Block-Level Consistency of Replicated Files

John L. Carroll *Darrell D. E. Long* *Jehan-François Pâris*

Computer Science Division
Department of Mathematical Sciences
San Diego State University
San Diego, California 92182

Computer Systems Research Group
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093

ABSTRACT

We investigate the construction of a *reliable device.* Such a device appears to the file system as an ordinary block-structured device, but is implemented as a set of server processes on several sites. This allows for replication while leaving the operating system kernel and the file system unchanged.

The regular structure of the block-level replication environment allows the use of consistency control algorithms that are simpler and less network intensive. We present three algorithms for maintaining file consistency in a block-level replication environment. The first is a *majority consensus voting* algorithm that recovers blocks only when required for data access; the second is a variant of the *available copy* scheme modified for replication at the block level; the third is a naive version of the available copy scheme that does not maintain any failure information.

Each scheme is evaluated in terms of availability and network traffic. While block-level replication is shown to allow improvements in the network traffic burden incurred by voting, available copy schemes are shown to have better availability and require significantly less traffic than voting schemes. The naive available copy variant proposed here is shown to be the algorithm of choice.

1. Introduction

A method often employed to increase *availability* and *reliability* of files is to replicate data at several sites. In this way, if a site fails it is likely that the other sites will continue to operate and to provide access to the file. Availability and reliability of a file can be made arbitrarily high by increasing the order of replication.

A *replicated file* is an abstract data object with the same semantics as an ordinary file, but which has greater availability and reliability because data are replicated. A common method for implementing a replicated file [2] is to replicate on a per file basis. While conceptually attractive, file-level replication can lead to unnecessary complications for the implementor in trying to preserve file system semantics.

We are constructing a *reliable device* which appears to the file system as an ordinary block-structured device, but which is implemented as a set of server processes on several sites. Because it presents the same interface as an ordinary device, the file system requires no modification and normal file system semantics are preserved.

This work was supported in part by a grant from the NCR Corporation and the University of California MICRO program.

We consider three policies for maintaining consistency in a block-level replication environment. The first is a variant on *majority consensus voting,* which takes advantage of the block-level replication and recovers out-of-date blocks only when they are required by the file system. This decreases network traffic and simplifies the recovery process. We also present two *available copy* algorithms modified for the block-level environment which provide extremely high availability in the absence of network partitions.

The amount of network traffic generated by these policies is also analyzed. The best overall scheme is shown to be the *naive available copy* algorithm. Its availability and low network traffic make it the algorithm of choice for implementing a reliable device.

This paper is organized into five sections: Section 2 describes our system model; Section 3 describes the consistency control and access algorithms; Section 4 presents an analysis of the performance of our schemes in terms of availability; Section 5 presents an analysis of network traffic; Section 6 has our conclusions.

2. The Model

A common method for implementing a replicated file system is at the file level [2]. The file is treated as a logical entity that is replicated on a per file basis. While conceptually simple, making the file the unit of replication leads to complications for the implementor when trying to preserve file system semantics. If the file system is part of the operating system kernel, as in most systems, the implementor has a choice: provide a replicated file system on top of the operating system as a set of library procedures, or move the replication into the operating system kernel.

The first choice is unsatisfactory because the implementor must provide an interface that preserves the semantics of the original file system using only extant system services. In essence, the implementor must build an entire replicated file system on top of the original file system. The second choice is also less than satisfactory because it requires modification to the operating system kernel.

A *reliable device* is implemented by a set of server processes on several sites and appears to the file system as an ordinary block-structured device. Because it presents the same simple interface as an ordinary device, the file system requires no modification and normal semantics are preserved. This approach has the advantage that extant programs can operate on replicated files without modification.

In the case of a conventional operating system such as UNIX† where the file system is part of the operating sys-

†UNIX is a Trademark of Bell Laboratories.

tem kernel, we would install a device driver stub which would receive requests for block access from the file system and would forward those requests to a user-state server which would perform the data access and consistency control algorithms. Such a scheme is illustrated in Figure 1.

In the UNIX model, a user-state process makes a file system request to the operating system kernel. The file system consults internal data structures to ascertain if it has the requested block in the buffer cache. If the block is not present then the file system requests the device driver to fetch the block. The device driver stub then communicates this request to the user-state server which executes the consistency control and data access algorithms.

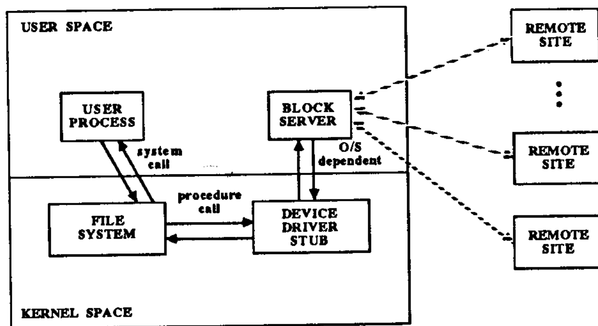

Figure 1: The UNIX Model

In a system design such as MACH [1], the reliable device would be implemented as a user-state server process which communicates with the file system via the interprocess communication mechanism as is illustrated in Figure 2.

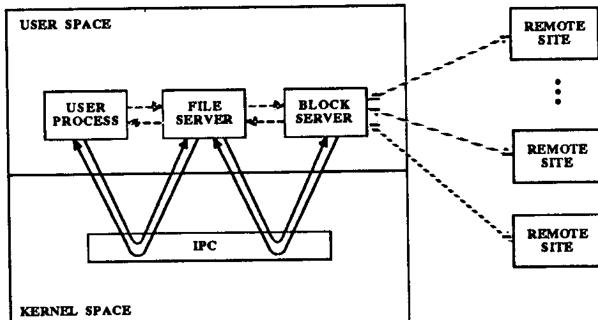

Figure 2: The MACH Model

Since the server is a user-state process there is no reason to require it to reside on the same site as the device driver stub, in the case of UNIX, or on the same site as the file system manager in the case of MACH. Because of this, the reliable device model operates easily in the context of diskless workstations.

3. Consistency Control Algorithms

We present three algorithms for maintaining file consistency in a block-level replication environment. Our schemes are less expensive than schemes that replicate at the file level because they recover only those blocks which have been modified during the time that the site was under repair. The savings in recovery time and network traffic can be significant in the context of large, long-lived files.

3.1. Majority Consensus Voting

Majority consensus voting schemes [3,4,5,6,10,12] insure the consistency of replicated files by honoring read and write requests only when an appropriate quorum of the sites holding copies of the file can be accessed.

In its simplest form voting assumes that the correct state of a replicated file is the state of the majority of its copies. Ascertaining the state of a replicated file requires collecting a quorum of the copies. Should this be prevented by one or more site failures, the file is considered unavailable.

The algorithm for reading first collects votes from all operational sites. These votes contain the version number of the requested block along with any weight assigned to the site. If a quorum is present access to the data block can proceed. If the local copy of the data block is out-of-date, then it is requested from the site which presented the highest version number. The restrictions on quorum composition insure that any quorum must contain the most current copy, so it is never necessary to do any recovery when a read operation is requested. However, it is more efficient to keep the local copy up to date as we do in our algorithm. This algorithm is illustrated in Figure 3.

function READ(s : site, k : index, B : block) : boolean
begin
  let Q be the set of all reachable sites
  if $\sum_{i \in Q} w_i > read\ quorum$ **then**
    let t be a site such that $v_{t,k} = \max_{i \in Q}\{v_{i,k}\}$
    if $v_{s,k} < v_{t,k}$ **then**
      request_block(t, k, B)
      write_block(k, B)
      $v_{s,k} \leftarrow v_{t,k}$
    end if
    read_block(k, B)
    return TRUE
  else
    return FALSE
  end if
end READ

Figure 3: Weighted Voting Read Algorithm

The algorithm for writing is even simpler. Votes are collected from all of the operational sites. These votes contain the version number of the requested block and any weight assigned to the site. If a quorum is present then the maximum version number is found and incremented. The restrictions on quorum composition insure that a site with the highest version number must be present in any quorum. The incremented version number is sent along with the data block to all sites in the quorum. This repairs all out-of-date copies that are operational. This algorithm is illustrated in Figure 4.

```
function WRITE(s : site, k : index, B : block) : boolean
begin
  let Q be the set of all reachable sites
  if ∑_{i∈Q} w_i > write quorum then
    v_{s,k} ← max_{i∈Q}{v_{i,k}} + 1
    send_block(Q, k, B, v_{s,k})
    write_block(k, B)
    return TRUE
  else
    return FALSE
  end if
end WRITE
```

Figure 4: Weighted Voting Write Algorithm

3.2. Available Copy

When network partitions are known to be impossible, *available copy* [7,8] schemes provide a simple means for maintaining file consistency. Available copy schemes are based on the observation that so long as at least one site has been continuously available it is known to hold the most recent version of the data blocks making up the files. An available copy scheme provides higher availability than voting schemes because it can continue to operate when all but one site have failed, while voting schemes require a majority of the sites to be operational in order to function.

The rule for writing a block when using an available copy scheme is simple: *write to all available copies.* Since all available copies receive each write request, they are kept in a consistent state: data can then be read from *any* available copy. If there is a copy of the data block on the local site, then the read operation can be done locally, avoiding any network traffic.

When a site recovers following a failure, if another site holds the most recent version of the data blocks the recovering site can repair immediately. In order to speed recovery, it is desirable to ascertain as quickly as possible the last site, or set of sites, that failed.

We consider a method which requires only that the availability information be brought up to date when a data block is modified or when a repair operation occurs. Our scheme assumes a fixed set of sites participating in the replication which are connected via a network which provides reliable message delivery and is free of partitions. We assume *clean* failure; if a site fails it simply halts, malevolent failures are not tolerated. This fail-stop [11] behavior can be simulated by an appropriate software layer.

Definition 3.1. The *was-available* set, denoted W_s, for a site s is the set of all sites that received the most recent write request and all of those sites which have repaired from site s.

The *was-available* sets represent those sites which received the most recent change to a data block. We posit the existence of an atomic broadcast mechanism. This condition can be relaxed by ascertaining which sites are operational when the sites first communicate and by sending this information along with the first write request. The second write request will contain the set of sites which received the first write request and so forth. By delaying the information in this way, communication costs are minimized at the expense of some small increase in recovery time.

Definition 3.2. Let $S=\{s_1, \cdots, s_n\}$ be the set of sites which hold copies of the data blocks; then the *closure* of a was-available set W_s is denoted by $C^*(W_s)$.

The closure of the was-available set has been described in an earlier paper [8].

The sites making up the system where the data blocks are replicated can be in any one of three states: *failed, comatose* or *available.* A *failed* site is one that has ceased to function due to hardware or software failure. A *comatose* site is one that has been repaired but the current state of the data blocks is not known. Sites enter this state following a total failure and remain there until the most recent version of the data blocks are found by examining the version numbers of the other sites. A site that has been continuously operational or that has been repaired and holds the most recent version of the data blocks is said to be *available.*

When a site recovers from a failure, it must communicate with other sites to determine if it holds the most recent version of the data blocks. In all cases where a site s repairs from another site t, it does so by first sending to t a version vector **v** containing what it believes to be the correct version number for each of the data blocks. Site t returns another version vector **v'** which contains the correct version number for each data block along with copies of those blocks which have been modified while s was not operational. Site s replaces those blocks which are out-of-date with current copies and **v'** becomes the new **v**. The recovery algorithm is illustrated in Figure 5.

```
procedure RECOVERY(s : site)
begin
  state(s) ← comatose
  select
    when all sites in C*(W_s) have recovered
      let t ∈ C*(W_s) : ∀u ∈ C*(W_s), version(t) ≥ version(u)
  or
    when ∃u ∈ S : state(u)=available
      let t be any such u
  end select
  if s≠t then
    send(t v)
    request(t, (v', {blocks}))
    repair those blocks that differ in v'
    v ← v'
    W_s ← W_t ∪ {s}
    send(t, W_s)
  end if
  state(s) ← available
end RECOVERY
```

Figure 5: Available Copy Recovery Algorithm

3.3. Naive Available Copy

A *naive available copy* [8] scheme does not attempt to detect the last site to fail. Because it does not maintain availability information about sites holding copies of the data block, network traffic is reduced at the cost of introducing poor worst-case behavior.

The naive scheme operates as the previous scheme would if the was-available sets were fixed so that $\forall s \in S, W_s=S$, where S is the set of all sites. Sites recover as in the previous scheme except that no site availability information is kept. The algorithm for such a scheme is

illustrated in Figure 6.

```
procedure SIMPLE_RECOVERY(s : site)
begin
   state(s)←comatose
   select
      when all sites have recovered
         let t∈S : ∀u∈S, version(t)≥version(u)
   or
      when ∃u∈S : state(u)=available
         let t be any such u
   end select
   if s≠t then
      send(t, v)
      request(t, (v', {blocks}))
      repair those blocks that differ in v'
      v←v'
   end if
   state(s)←available
end SIMPLE_RECOVERY
```

Figure 6: Naive Available Copy Recovery Algorithm

4. Availability Analysis

In this section we compare the availabilities of replicated blocks managed by majority consensus voting, available copy and naive available copy. In all three cases, we assume that the copies of the replicated block reside on distinct *sites* of a computer network. Sites are subject to failure. When a site fails, a repair process is immediately initiated. Should several sites fail, the repair process will be performed in parallel on these failed sites. We also assume that the repair process will attempt to bring up to date all the copies that might have become obsolete during the time the site under repair was not operational. Such attempts will not always be successful since they depend on the availability of up-to-date copies of the replicated block. Since the available copy algorithm does not operate correctly in the presence of partitions, we assume that the communications network linking the several sites where the physical copies of the replicated blocks reside cannot fail.

We assume that individual site failures and that individual site repairs are independent events distributed according to a Poisson law. In other words, the probability that a given site will experience no failure during a time interval of duration t is $e^{-\lambda t}$ where λ is the *failure rate,* and the probability that a given site will be repaired in less than t time units is $1-e^{-\mu t}$ where μ is the *repair rate.*

The availability A of a system is the limiting value of the probability $p(t)$ that system will be operating correctly at time t.

$$A = \lim_{t \to \infty} p(t)$$

4.1. Majority Consensus Voting

We will restrict our analysis to the case where all sites containing copies have equal failure rates λ and equal repair rates μ. Under these conditions, it is common to assign equal weights to all copies. Equal weights cause a particular problem for replicated blocks with an *even* number of copies. Draw conditions will occur every time an equal number of copies are up and down. To solve these ties, we will need to adjust by a small quantity the weight of one of the copies. The *availability* $A_V(n)$ of replicated bock with n copies will be given by [10]

$$A_V(n) = \sum_{j=n}^{\lceil n/2 \rceil} \frac{\begin{bmatrix} n \\ n-j \end{bmatrix} \rho^{n-j}}{(1+\rho)^n} \qquad \text{for } n \text{ odd (1.a)}$$

and

$$A_V(n) = \sum_{j=n}^{n/2+1} \frac{\begin{bmatrix} n \\ n-j \end{bmatrix} \rho^{n-j}}{(1+\rho)^n} + \frac{\begin{bmatrix} n \\ n/2 \end{bmatrix} \rho^{n/2}}{2(1+\rho)^n} \qquad \text{for } n \text{ even (1.b)}$$

with $\rho=\lambda/\mu$. This latter expression can be rewritten as

$$A_V(2k) = \frac{1}{(1+\rho)^{2k}} \left[\begin{bmatrix} 2k \\ 0 \end{bmatrix} + \cdots + \begin{bmatrix} 2k \\ k-1 \end{bmatrix} \rho^{k-1} + \frac{1}{2} \begin{bmatrix} 2k \\ k \end{bmatrix} \rho^k \right]$$

$$= \sum_{j=0}^{k-1} \frac{\begin{bmatrix} 2k-1 \\ j \end{bmatrix} \rho^j}{(1+\rho)^{2k-1}} = A_V(2k-1)$$

4.2. Available Copy

The state-transition-rate diagram for a replicated block having n copies has $2n$ states. The first n states labelled from S_1 to S_n represent the states of the block when 1 to n copies are available; n new states labelled from S'_0 to S'_{n-1} represent the states of the block when all copies of the block have failed and 0 to $n-1$ copies not including the copy that failed last have recovered but remain comatose. As seen in Figure 7, all states S_j with $j=1, \cdots, n-1$ have one outward transition leading to state S_{j-1} (S'_0 for S_1) and corresponding to the failure of one of the j available copies and one outward transition to state S_{j+1} corresponding to the recovery of one of the $n-j$ failed copies. State S_n has only one outward transition leading to S_{n-1}. Once *all* copies of the block have failed, the block is in state S'_0 and will return to state S_1 if and only if the last available copy recovers. If any of the $n-1$ other copies recovers, that copy will remain *comatose* and the block will be in state S'_1. As a result, state S'_0 has one outward transition with rate μ leading to state S_1 and another one with rate $(n-1)\mu$ leading to state S'_1.

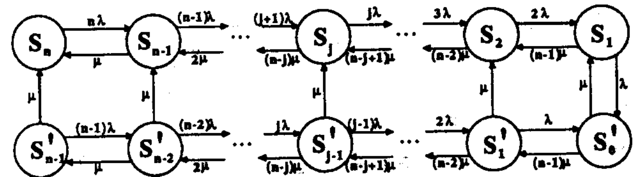

Figure 7: State Diagram for Available Copy

All states S'_j with $j=1,...,n-2$ have three outward transitions: one leading to state S'_{j-1} corresponding to the failure of one of j the comatose copies, another one with rate μ leading to state S_{j+1} corresponding to the recovery of the last available copy, and a third one with rate $(n-j-1)\mu$ leading to state S'_{j+1} corresponding to the recovery of one of the other $n-j-1$ failed copies. State S'_{n-1} has no third outward transition since the only failed

copy is necessarily the last available copy.

One can easily derive from the equilibrium conditions for our system the availability $A_A(n)$ of the replicated block

$$A_A(n) = \sum_{i=1}^{n} p_i$$

where p_i denotes the probability that the block is in state S_i. In particular, we have [8]

$$A_A(2) = \frac{1+3\rho+\rho^2}{(1+\rho)^3} \quad (2)$$

$$A_A(3) = \frac{2+9\rho+17\rho^2+11\rho^3+2\rho^4}{(1+\rho)^3(2+3\rho+2\rho^2)} \quad (3)$$

$$A_A(4) = \frac{6+37\rho+99\rho^2+152\rho^3+124\rho^4+47\rho^5+6\rho^6}{(1+\rho)^4(6+13\rho+11\rho^2+6\rho^3)} \quad (4)$$

where $\rho = \lambda/\mu$.

A more general lower bound for the availability of available copy algorithms can be derived from the equilibrium of flows between states $S_n, S_{n-1}, \cdots, S_2, S_1$ and states $S'_{n-1}, S'_{n-2}, \cdots, S'_1, S'_0$. Since we have

$$\mu(p'_{n-1} + p'_{n-2} + \cdots + p'_1 + p'_0) = \lambda p_1$$

and

$$p_1 + p'_1 = \frac{n\rho^{n-1}}{(1+\rho)^n}$$

we can obtain a lower bound for the probability of being in any of the non-available states:

$$p'_{n-1} + p'_{n-2} + \cdots + p'_1 + p'_0 < \frac{n\rho^n}{(1+\rho)^n}$$

Hence,

$$A_A(n) = 1-(p'_{n-1} + p'_{n-2} + \cdots + p'_1 + p'_0) < 1-\frac{n\rho^n}{(1+\rho)^n} \quad (5)$$

Theorem 4.1. The availability $A_A(n)$ of a replicated block with n identical copies managed by an available copy consistency algorithm is greater than the availability $A_V(n)$ of a block with $2n-1$ or $2n$ identical copies managed by a voting algorithm as long as the failure-to-repair rate ratio ρ remains less than or equal to one.

Proof: Since $A_V(2n-1) = A_V(2n)$, we only need to prove that $A_A(n) > A_V(2n-1)$ for all $\rho \le 1$.

i. From equations (1.a) and (3), we know that $A_A(3) > A_V(5)$.

ii. For $k \ge 4$, let us compare the lower bound for $A_A(n)$ given by inequality (5) with the upper bound for $A_V(2n-1)$

$$A_V(2n-1) < 1-\frac{\begin{bmatrix}2n-1\\ n\end{bmatrix}\rho^n}{(1+\rho)^{2n-1}}$$

A sufficient condition for $A_A(n) > A_V(2n-1)$ is then given by

$$\frac{\begin{bmatrix}2n-1\\ n\end{bmatrix}}{n} > (1+\rho)^{n-1} \quad (6)$$

This inequality holds for $n = 4$ and any $\rho \le 1$. Since

$$\frac{\begin{bmatrix}2n+1\\ n+1\end{bmatrix}}{n}+1 = 2n+\frac{1}{n}+1\frac{2n}{n}+1\frac{\begin{bmatrix}2n-1\\ n\end{bmatrix}}{n} > 2\frac{\begin{bmatrix}2n-1\\ n\end{bmatrix}}{n}$$

for all $n > 1$, it also holds by recurrence for all $n > 4$ and any $\rho \le 1$.

4.3. Naive Available Copy

In the naive available copy algorithm, no record is kept of which copy failed last. Once all the copies of a block have failed, the recovery algorithm will then have to wait until *all* copies of the block have recovered. It will then select the copy with the highest version number, mark it as being available and use it to bring all other copies of the block up-to-date.

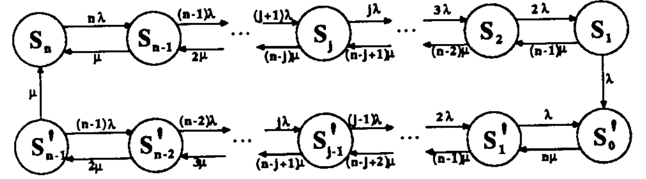

Figure 8: State Diagram for Naive Available Copy

As seen in Figure 8, the state-transition-rate diagram for a replicated block of n copies managed by a naive available copy algorithm has the same $2n$ states as if the block was managed by a conventional available copy algorithm. Transitions between states will be quite similar to those observed for a conventional available copy algorithm with the exception that there will be no transitions from state S'_j with $j \le n-2$ to an available state.

From the state transition diagram, we have

$$k\lambda p_k = (n-k+1)\mu p_{k-1} + \lambda p_1 \qquad k = 2, 3, \ldots, n \quad (7)$$

$$k\mu p'_{n-k} = (n-k+1)\lambda p'_{n-k+1} + \mu p'_{n-1} \qquad k = 2, 3, \ldots, n \quad (8)$$

$$\lambda p_1 = \mu p'_{n-1} \quad (9)$$

From equation (7), we obtain

$$p_k = \sum_{j=1}^{k} \frac{(n-j)!\,(j-1)!}{(n-k)!\,k!}\rho^{j-k} p_1$$

and from equation (8)

$$p'_{n-k} = \sum_{j=1}^{k} \frac{(n-j)!\,(j-1)!}{(n-k)!\,k!}\rho^{k-j} p'_{n-1}$$

Since the sum of the probabilities of being in any given state must be equal to one,

$$p_1 = \frac{1}{B(n;\rho) + \rho B(n;\frac{1}{\rho})}$$

where

$$B(n;\rho) = \sum_{k=1}^{n}\sum_{j=1}^{k} \frac{(n-j)!\,(j-1)!}{(n-k)!\,k!}\rho^{j-k}$$

The *availability* $A_{NA}(n)$ of a replicated block with n copies managed by a naive available copy consistency algorithm is then given by

$$A_{NA}(n) = \sum_{k=1}^{n} p_k = \frac{B(n;\rho)}{B(n;\rho) + \rho B(n;\frac{1}{\rho})}$$

Note that $A_{NA}(2) = A_V(3)$, which means that *two* copies managed by our naive available copy algorithm have the same availability as *three* copies managed by a voting algorithm.

4.4. Discussion

We have already shown that the conventional available copy algorithm with n copies performed better than voting with $2n$ copies. The figures 9 and 10 contain the availabilities of replicated blocks with three and four available copies respectively compared with the availabilities of replicated blocks with six and eight voting copies. In all three graphs, ρ varies between 0 and 0.20; the first value corresponding to perfectly reliable copies and the latter to copies that are repaired five times faster than they fail and have an individual availability of 83.33%.

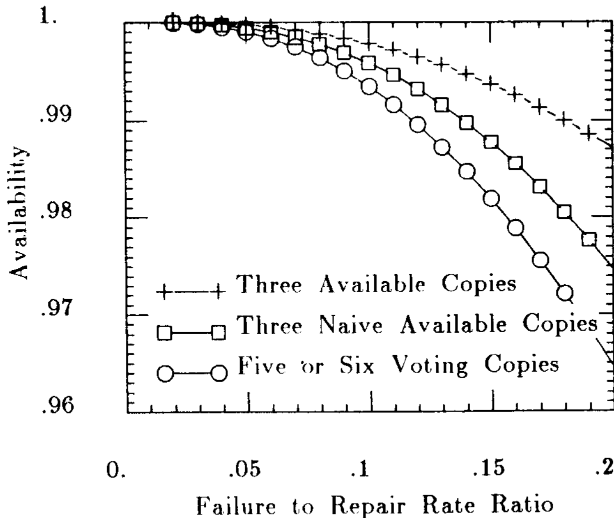

Figure 9: Availabilities for Three Available Copies and Six Voting Copies

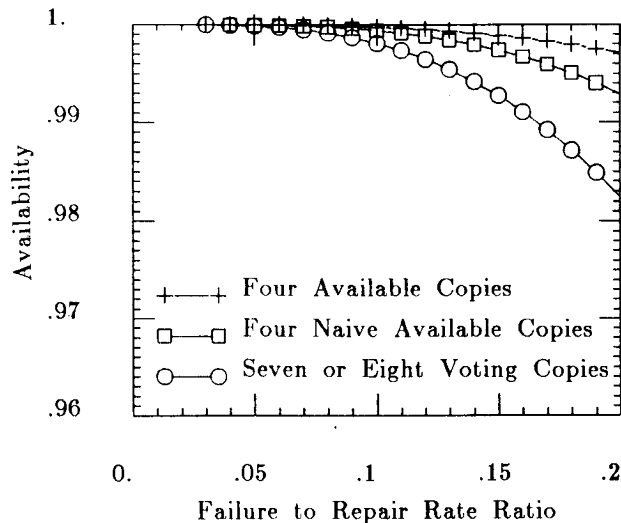

Figure 10: Availabilities for Four Available Copies and Eight Voting Copies

These graphs clearly indicate that both the traditional and the naive available copy algorithms produce much higher availabilities than voting. And, they fail to show any significant difference between the two available copy algorithms under investigation for values of ρ less than 0.10. Most of today's computers are characterized by availabilities well above 0.95 and by values of ρ well below 0.05, which could lead us to the conclusion that the naive recovery algorithm would perform as well as the conventional algorithm. Besides, observed repair time distributions are characterized by coefficients of variation less than one. Under such conditions, sites will tend to recover in the same order as they failed. The last site to recover after a total failure will often be the last one that failed. When this happens, the conventional available copy algorithm will be unable to recover faster than our naive algorithm as it will have to wait for the last copy to recover in order to get the last copy that failed.

5. Network Traffic Analysis

We compare the cost in network traffic of majority consensus voting, available copy, and naive available copy by analyzing the number of transmissions required by each scheme, since network congestion is influenced mainly by the number of messages rather than the size of the messages. While it is possible to instead focus on the sizes of the messages by estimating the total number of actual blocks transferred by each scheme, the differences are similar to the results obtained below, though slightly less pronounced.

This analysis will focus on the number of high-level transmissions that occur, such as requests for version vectors, block transfers, and the like. The details of the network implementation will determine the actual number of messages generated by a high-level request. While these low-level transmissions may vary with different networks, their number should be proportional to the number of high-level requests. Consequently, this analysis will focus on the number of high-level transmissions. We do not attempt to model systems which guard against concurrent access of files; each of the consistency schemes would then require further message traffic to implement appropriate commit protocols.

The number of messages generated by a given operation often depends on the average number of sites participating in the operation, which in voting depends on the average number of operational sites and in the available copy schemes involves the average number of available sites. The average number of sites responding to a query from some local site, given, of course, that the local site is operational and available, is:

$$U = \frac{\sum_{i=1}^{n} i p_i}{\sum_{i=1}^{n} p_i}$$

Since the values for p_i vary for each of the three schemes, this formula is dependent on the balance equations corresponding to the scheme being used. Let U_V^n, U_A^n and U_N^n denote the average number of participating sites in an n-site network using majority consensus voting, available copy and naive available copy, respectively. The formula for voting participation, for example, is

$$U_V^n = \frac{n(1+\rho)^{n-1}}{(1+\rho)^n - \rho^n}$$

From this it follows that $U_V^n = n(1-\rho)+O(\rho^2)$. U_V^n, U_A^n and U_N^n all agree to within $O(\rho^2)$, which is negligible for values of ρ in the range typical for computer systems.

We consider two types of networks in turn: multi-cast mechanisms in which a single transmission may be received by several sites, and networks which require transmissions to be addressed to an individual site. The three schemes retain their relative advantages in either type of network, though the differences are amplified in a single destination network.

5.1. The Multi-cast Environment

In a multi-cast network, the naive available copy scheme need only broadcast one message when a write is performed, and the reliable delivery assumption is sufficient to guarantee that this write is successful. It has been shown [8] that consistency can be guaranteed without postulating an atomic broadcast mechanism.

A write using the available copy scheme will also always be successful. However, the local site now also receives responses from each of the other operational sites, and thereby determines a more current was-available set. Since the local site broadcasts the write and the remaining operational sites each reply, the average number of messages caused by a write in an available copy scheme using this mechanism in an n-site network is therefore U_A^n. By contrast, we will consider voting to require one message querying the existence of a quorum, which provokes return messages from each of the other operational sites before the block update is actually broadcast. The message traffic for successful writes in voting would then be $1+U_V^n$. The expected message traffic associated with the modification of a single replicated block in the majority consensus voting scheme described here is $1+n(1-\rho)+O(\rho^2)$ network transmissions, as compared to $n(1-\rho)+O(\rho^2)$ for available copy and 1 for naive available copy.

Read access generates no network traffic in the available copy schemes since all available sites contain a local copy of the most recent version of each block. Voting again requires the collection of a quorum before a read is permitted, resulting in at least U_V^n messages, and at most U_V^n+1 if the local version is not up to date. A lower bound on the expected message traffic associated with the access of a single replicated block in majority consensus voting is $n(1-\rho)+O(\rho^2)$ network transmissions, as compared to zero for the available copy schemes.

There is one other situation in which the consistency schemes might provoke network traffic: after site repair. Block-level replication makes it possible for voting to dispense with recovery upon repair, without degrading user access or availability. The voting algorithm presented in this paper incurs no traffic upon recovery.

A site in an available copy network must attempt recovery by broadcasting one request for information from the remote sites, which will generate responses from each of the operational sites. The recovery of this site will require two more messages: a request for the transmission of a version vector, and the response to that request, though these two last transmissions may be delayed until the remaining sites in the was-available set are repaired. The average total number of messages upon recovery for both available copy and naive available copy is therefore U_A^n+2 and U_N^n+2 respectively, represented by $2+n(1-\rho)+O(\rho^2)$.

While both available copy schemes incur no network traffic for reads, naive available copy requires fewer messages for writing, and has recovery costs comparable to those of the available copy scheme. In voting, reads are almost as expensive as writes, and voting requires significantly more messages to write than available copy.

Since the block-level replication allows voting to dispense with recovery traffic, the margin by which available copy outperforms voting is dependent on the frequency of reads compared to site failures. The relative scarcity of site failures suggests that recovery traffic be discounted entirely; it is interesting to note that site failures would have to be more frequent than disk accesses in order for the voting schemes to begin to compare favorably to the available copy schemes.

The two available copy schemes have identical read and recovery costs; naive available copy gains its advantage by employing a less complex write protocol. The advantage generated by the naive available copy scheme is therefore dependent on the relative frequency of reads to writes. Research on observed access patterns of typical computer systems show the read to write ratio to be in the neighborhood of 2.5:1 [9].

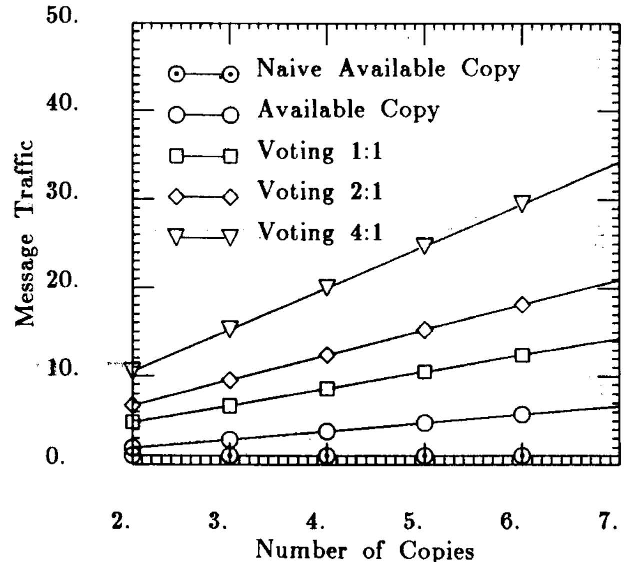

Figure 11: Multi-cast Results

Figure 11 illustrates the comparisons for several read to write ratios for a typical value of ρ (ρ=0.05). The dependent axis reflects the number of high-level network transmissions generated by one write and x reads, where x is the expected number of reads associated with a single write. Varying x does not affect the performance of the available copy schemes since their read traffic cost is zero. Voting costs are illustrated for values of x from 1 to 4, reflecting read to write ratios of 1:1, 2:1, 4:1. The graph compares schemes employing the same number of sites. A comparison of schemes with equal availabilities would result in much steeper voting traffic costs.

5.2. The Unique Addressing Environment

In the absence of a multi-cast network, separate messages must be individually addressed to each destination site. While this increases the message traffic incurred by each of

the three schemes, their relative differences remain intact.

A write using the available copy schemes now account for $n-1$ messages, since the local site must send individual messages to each of the remote sites. Available copy, on the other hand, incurs added traffic since each of these sites now respond to the local site, accounting for $n+U_A^n-2$ messages. A write under the voting scheme incurs even more traffic, since the local site must request quorum information and version numbers from all $n-1$ remote sites, receive responses from U_V^n-1 sites, and then send the updated block to each of those sites, accounting for $n+2U_V^n-3$ messages.

Reading a block still generates no message traffic in the available copy schemes, but voting must again ascertain a quorum, leading to a traffic burden of $n+U_V^n-2$, or, if the most recent version is not local, $n+U_V^n-1$. Voting again has no recovery costs, while the available copy schemes must ask each possible site for updates, receive version vectors, and at some point request and receive from a specific remote site the updated blocks, accounting for $n+U_A^n$ or $n+U_N^n$ messages.

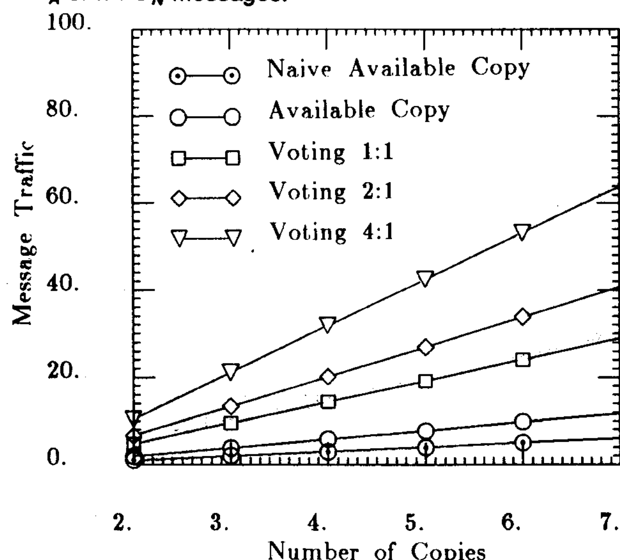

Figure 12: Unique Address Results

Figure 12 illustrates the traffic incurred by each of the three schemes for different values of n for a typical value of ρ. Typical read to write ratios are again used. Factoring in the overhead of unsuccessful writes in voting would produce an even less favorable comparison. Similarly, the contrast between schemes with equal availabilities rather than equal sites further amplifies the advantages of available copy over majority consensus voting.

6. Conclusions

To increase the availability and reliability of files the data are often replicated at several sites. The usual method is to treat files as logical entities and to replicate on a per file basis. This can lead to unnecessary complications for the implementor in trying to preserve file system semantics.

We have investigated the construction of a *reliable device*. Such a device appears to the file system as an ordinary block structured device, but is implemented as a set of server processes on several sites. This allows us to provide replication while leaving the operating system kernel and the file system unchanged. Since the file system is not modified, the file operation semantics remain the same.

The consistency control algorithms based on majority consensus voting were impressively overshadowed by the performance of the available copy schemes. A consistency control mechanism based on available copy had the availability of a voting scheme with twice the number of sites.

Available copy was shown to be significantly less network-intensive than voting when the two schemes employed the same number of sites, and the available copy mechanisms even further outperform voting when systems with similar availabilities are compared.

The voting schemes obviate the concern for network partitions. However, the available copy algorithms do not require the added burden of insuring reliable message delivery, nor do they have to deal with failed access attempts. An available site is not dependent on the existence of any quorum in order to successfully read or write a block.

The naive available copy scheme presented in this paper likewise eclipses the standard available copy algorithm. Its simplicity allows lower implementation costs and incurs fewer network transmissions without measurably sacrificing block availability.

Acknowledgements

We are grateful to Walter Burkhard and all of the members of the Gemini Research Group for their support and encouragement. We are also indebted to Robin Fishbaugh, Ernestine McKinney and Mary Long for their assistance.

This work has been done with the aid of MACSYMA, a large symbolic manipulation program developed at the Massachusetts Institute of Technology. MACSYMA is a trademark of Symbolics Incorporated.

References

[1] Baron, R. V., R. F. Rashid, A. Tevanian and M. W. Young, *MACH-1 Kernel Interface Manual*, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, 1986.

[2] Burkhard, W. A., B. Martin and J.-F. Pâris, "The Gemini Fault-Tolerant File System Test-Bed," *Proc. 3rd Int. Conf. on Data Engineering*, 1987, 441-448.

[3] Ellis, C. A., "Consistency and Correctness of Duplicate Database Systems," *Operating Systems Review*, 11, 1977.

[4] Garcia-Molina, H., "Elections in a Distributed Computer Systems," *IEEE TOC*, C-31, 1982, 48-59.

[5] Garcia-Molina, H. and D. Barbara, "Optimizing the Reliability Provided by Voting Mechanisms," *Proc. 4th DCS*, 1984, 340-346.

[6] Gifford, D. K. "Weighted Voting for Replicated Data," *Proc. 7th ACM SOSP*, 1979, 150-161.

[7] Goodman, N., D. Skeen, A. Chan, U. Dayal, R. Fox and D. Ries, "A Recovery Algorithm for a Distributed Database System," *Proc. 2nd ACM PODS*, 1983, 8-15.

[8] Long, D. and J.-F. Pâris, "On Improving the Availability of Replicated Files," *Proc. 6th SRDSDS*, 1987, 77-83.

[9] Ousterhout, J., H. Da Costa, D. Harrison, J. Kunze, M. Kupfer and J. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," *Proc. 10th ACM SOSP*, 1985, 15-24.

[10] Pâris, J.-F., "Voting with a Variable Number of Copies," *Proc. 16th FTCS*, 1986, 50-55.

[11] Schlichting, R. D. and F. B. Schneider, "Fail Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM TOCS*, 1983, 222-238.

[12] Thomas, R. H., "A Majority Consensus Approach to Concurrency Control," *ACM TODS* 4, 1979, 180-209.