AD-A043 051 ILLINOIS UNIV AT URBANA-CHAMPAIGN CENTER FOR ADVANCED--ETC F/G 9/2
RESEARCH IN NETWORK DATA MANAGEMENT RESOURCE SHARING. MULTI-COP--ETC(U)
MAY 76 P A ALSBERG, G G BELFORD, J D DAY DCA100-75-C-0021
UNCLASSIFIED UIUC-CAC-DN-76-202 CCTC-WAD-6505 NL

1 OF 1
AD
A043051

END
DATE
FILMED
9-77
DDC

CAC Document Number 202

CCTC-WAD Document Number 6505

Research in

Network Data Management

Resource Sharing

Multi-Copy Resiliency Techniques

May 31, 1976

CAC Document Number 202
CCTC-WAD Document Number 6505

Research in Network Data Management Resource Sharing,

Multi-Copy Resiliency Techniques

by

Peter A. Alsberg
Geneva G. Belford
John D. Day
Enrique Grapa

Research rept.

Prepared for the
Command and Control Technical Center
WWMCCS ADP Directorate
of the
Defense Communication Agency
Washington, D.C.

UIUC-CAC-DN-76-202
CAC-202

DDC
RECEIVED
AUG 16 1977
A

under contract
DCA100-75-C-0021

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
DISTRIBUTION/AVAILABILITY CODES
SPECIAL
A

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

53p.

31 May 1976

Approved for release:
Peter A. Alsberg, Principal Investigator

1473
407 227
LB

Table of Contents

Page

Executive Summary 1

Distributed Resource Sharing 1

Resiliency 1

Feasibility 3

Conclusions 6

Introduction 8

Resiliency 9

Examples 10

Related Work in Distributed Systems 11

Techniques for a Resilient Service 13

The Chained Model 14

A Broadcast Model 23

n-Host Resiliency for n > 2 25

Adequacy of Two-Host Resiliency 27

Resilient Service Down Time 27

Removing the Two-Host Resiliency Criterion 31

Failure Detection and Recovery 33

Overhead Due to Resiliency 39

Comparison of the Models 41

Alternative Resource Sharing Strategies 44

Range of Application 46

Synchronization Primitives 46

Directories and Data Access 47

Load Sharing 47

References 48

Executive Summary

Distributed Resource Sharing

Current intercomputer networks like the ARPANET, PWIN, CYCLADES, and others provide the basic communications facilities necessary to permit access to remote resources. These communication networks and their existing protocols are a necessary component of a distributed resource sharing system. However, by themselves, the communications networks are not sufficient to permit the automated distributed sharing of resources. High level protocols must be developed to permit cooperation in other than an ad hoc manner and techniques must be developed to provide resilient service to a user community. Network technology at the present time might better be characterized as resource access technology than as a technology which facilitates automated resource sharing.

In this paper we consider an environment which requires the sharing of resources dispersed over a large geographic area on a large number of possibly heterogeneous host computers. Since the hosts in this environment are separated by large distances, there is a significant and unavoidable message delay between hosts. Hence, a major consideration when choosing a resource sharing strategy is to minimize the message delays required to support sharing.

Resiliency

The current protocols on the ARPANET (and similarly in its "copy" PWIN) operate under basic assumptions that are at best questionable in a production networking environment. It is assumed that all hosts correctly obey protocol. It is assumed that no host is malicious. It is assumed that messages are not lost in the network. It is assumed that when a host fails, it will fail at a "convenient" point in the execution

of a protocol sequence. In fact, all of these assumptions are commonly violated every day in the ARPANET environment. What is required for production networking are resource sharing strategies which are as resilient as possible to protocol violations, malicious attack, communication failures, and host failures.

In this paper we describe resiliency techniques to support extreme reliability and serviceability. The type of resilient service addressed has four major attributes.

1. It is is able to detect and recover from a given maximum number of errors.
2. It is reliable to a sufficiently high degree that a user of the resilient service can ignore the possibility of service failure.
3. If the service provides perfect detection and recovery from n errors the (n+1)st error is not catastrophic. A "best effort" is made to continue service.
4. The abuse of the service by a single user should have a negligible effect on other users of the service.

These four points are a careful way of saying that the user of a resilient service should not have to consider the failure of the service in his design. He should be able to assume that the system will make a "best effort" to continue service in the event that perfect service cannot be supported. Finally, the system will not fall apart when he does something he is not supposed to.

Resiliency cannot be perfect in the large network environments we are considering. It is, for instance, possible but not likely that all of the hosts on a large computer network will simultaneously fail

and all services will be disrupted. Thus, it is important to establish criteria for acceptable resiliency in this environment. We introduce the concept of n-host resiliency. In order for service to be disrupted, n hosts must simultaneously fail in a critical phase of service. It may be possible for n or more hosts to fail outside of such a critical phase without disrputing service.

Feasibility

Techniques are described for supporting services that are two-host resilient to communication system and host failure. Furthermore, it is shown that greater than two-host resiliency is unnecessary. A two-host resiliency criterion is sufficient to achieve failure intervals measured in centuries. Furthermore, if implemented as described in this paper, this basic failure interval can be increased by many orders of magnitude.

The resiliency criterion is a question of service integrity. If the criterion is met, the service will almost certainly be functioning correctly. Unfortunately, large service hosts are down for substantial periods of time during the day for both scheduled maintenance and unscheduled failures. In order to guarantee that a service is available, a possibly large number of service sites may be required simply to have a reasonable expectation that at any time at least two of them will be up. This raises an important issue related to resiliency - the availability of resilient service.

Figure 7 and Tables 1 and 2 in this paper show the expected service down time under a wide range of possible host down times and number of service hosts. For example, a typical down time per day for large service hosts like those found on the PWIN tends to be in the range of two to four hours due to all causes (both scheduled and unscheduled

down time). If the down time averages four hours per day then a three host service would have interruptions averaging slightly more than 1½ hours per day. A four host service would have interruptions averaging slightly less than ½ hour per day. If the average server down time were a more optimistic two hours per day, then the resilient service would be down approximately ½ hour per day if three server hosts were involved and only three minutes per day if four server hosts were supported. In crisis situations it is possible temporarily to reduce the average host down time. For example, all but essential preventive maintenance can be eliminated. Also, qualified repair personnel can be stationed in the computer room around the clock for the duration of the crisis. It is conceivable that average down time per host may be reduced to one hour per day. Under those conditions a two host resilient service would still have almost two hours of interruption per day - clearly unacceptable in many crisis situations. However, three host resilient service would only be interrupted approximately seven minutes per day. A four host resilient service, however, would have an average interruption rate of slightly less than ½ minute per day.

When resilient service must be supported in a WWMCCS-like network, it appears that on the order of three or four service hosts must be supplied. This will support a service availability in an acceptable range during non-crisis periods. In a crisis situation, by judicious management of down time parameters, the same three or four service hosts should be capable of providing significantly improved availability for the duration of the crisis.

In some applications it may be desirable to maintain a two-host resilient strategy whenever two or more hosts are available.

However, if only one service host is available, it would be possible to discard the resiliency scheme and to operate with only the single host. A note of caution is important here: Two-host resiliency is a question of service integrity. The requirement for a large number of service hosts is imposed not so much by the two-host resiliency criterion as it is by the desired availability of service. Thus, if one discards the two-host resiliency criterion, service availability increases at the expense of service integrity.

In a single host environment it is possible to have undetected errors creep into distributed resources like data bases. This situation has been analyzed (see Table 3). The probability of service errors is greatly dependent on the frequency of the errors that would normally be detected by the resiliency scheme. If, for example, these normally detected errors occurred at the rate of one every one to four hours (a reasonable assumption based upon ARPANET experience) then a two host service which was permitted to operate in degraded one host mode would experience undetected errors at the rate of one every day or two. If three service hosts were permitted to degrade to single host service, then the errors could occur every three to nine days. If four service hosts were permitted to degrade to single host service, then undetected service errors would occur approximately every month. Note that the reduction of the service error rate to a multi-day period requires three or four service hosts. However, this is the number of hosts that already gives acceptable two-host resilient service without concern for service errors. Hence application of a single host degradation strategy currently appears limited to those two host services (for example, a master copy of a data base at one host and a single backup at a second host) where higher probabilities of service errors are acceptable.

The resiliency techniques described are more complex to implement than alternative strategies that would be allowable in a non-resilient service. However, little or no additional overhead is incurred due to the resiliency scheme. For example, the number of messages required by a resilient scheme is identical to that required by non-resilient schemes. However, the flow of messages is dramatically different and, in fact, there are fewer bottlenecks in the resilient than in a non-resilient scheme. In the case of storage and processing costs, the resilient scheme imposes a trivial additional burden to maintain some space for status tables and to periodically update those tables.

Conclusions

The most obvious immediate application for the results described in this paper is in WWMCCS data base problems. Here several questions have to be addressed before the analysis is complete. First, is single site service acceptable or are backups required? If single site service is acceptable then there is no need for a backup or resilient service.

If backups are required then the backup scheme should be made resilient. There is an initial one-time cost to properly program the resilient scheme. However, once programmed, the resilient scheme eliminates the concern for errors caused by network failures or host failures in any of the data base copies. At the same time, the resilient scheme does not increase network traffic, host storage requirements, or host processing requirements. Thus, once it has been decided to backup a data base, a two-host resilient scheme to maintain the primary and backup copies should be employed.

There is a special case if it is critical to maximize the availability of the primary and backup data base and if only one backup is permitted and if a high probability of daily errors creeping into the

master or backup data base is acceptable. Then the availability of the data base can be significantly improved by permitting a single host to continue service when its partner host is down without regard to the two-host resiliency scheme.

If the integrity of a data base is an important issue and if data base availability must be high, then on the order of three or four service hosts (i.e., a primary and two or three backups) are required to support that service. Under normal conditions, a three or four host scheme can expect anywhere from several minutes to one hour or more of service interruption in a day. The variability depends upon the average down time per host per day. In a crisis situation it appears possible to temporarily reduce the down time per host for a multi-day period. Then the data base will be available on the same three or four hosts all but a few minutes or less per day.

Introduction

The development of large packet switched networks servicing wide geographic areas has generated a great deal of interest in distributed resource sharing. A communications network is a necessary but, by itself, is not a sufficient basis to make automated distributed resource sharing facilities generally available. High-level protocols must be provided to allow cooperation in other than an ad hoc manner and techniques must be developed to provide resilient service to the user. This paper discusses one means by which resilient service may be provided to the user for a wide variety of situations, e.g., synchronization, data base access, and load sharing.

For our purposes we will consider a distributed resource sharing environment which requires the sharing of resources dispersed over a large number of possibly heterogeneous host computers. Large packet switched computer networks like the ARPANET, CYCLADES, and EIN represent examples of this environment. Since the hosts in this environment may be separated by very large distances, there is a significant and unavoidable message delay between hosts. Hence, a major consideration when choosing a resource sharing strategy is to reduce, as much as possible, the number of message delays required to effect the sharing of resources.

In these networks, some of the resources to be shared will be identical (e.g. duplicate copies of data bases may be maintained for reliability). Others will be completely dissimilar (e.g., weather data may be stored on the ARPANET datacomputer and processed on the ILLIAC IV). Between these two extremes lie the resource sharing concerns of interest to most users.

The user expects a tolerable, as well as tolerant, resource sharing environment. The user we are interested in wants a maximum degree of automation and transparency in his resource sharing. He wishes the resource sharing to be resilient to host failures and, when catastrophic failures occur, he would like a "best effort" recovery to be automatically initiated by the resource sharing system.

Resiliency

The concept of resiliency applies to the use of a resource as a service. A resilient service has four major attributes.

1. It is able to detect and recover from a given maximum number of errors.
2. It is reliable to a sufficiently high degree that a user of the resilient service can ignore the possibility of service failure.
3. If the service provides perfect detection and recovery from n errors, the (n+1)st error is not catastrophic. A "best effort" is made to continue service.
4. The abuse of the service by a single user should have negligible effect on other users of the service.

What we are trying to describe here are concepts of extreme reliability and serviceability. The user of a resilient service should not have to consider the failure of the service in his design. He should be able to assume that the system will make a "best-effort" to continue service in the event that perfect service cannot be supported; and that the system will not fall apart when he does something he is not supposed to.

In this paper we discuss a technique for providing resilient services. This technique is resilient to communication system and host

failures. Host failures include not only complete failure (e.g., a major hardware failure, but also partial failure (e.g., a malfunctioning host operating system). Resiliency cannot be perfect in the large network environments we are considering. It is, for instance, possible but not likely that all 50 of the hosts on a large computer network will simultaneously fail and all services will be disrupted. What is of interest is the establishment of criteria for acceptable resiliency in this environment. To this end, we introduce the concept of n-host resiliency. In order for service to be disrupted, n hosts must simultaneously fail in a critical phase of service. We point out that it may be possible for n or more hosts to fail outside of such a critical phase without disrupting service. The resiliency techniques discussed in this paper assume a two-host resiliency criterion. Expansion of the techniques to treat three-host or greater resiliency is straightforward. A two-host resiliency criterion has been used because it appears sufficient to provide an adequate level of service in most situations and to illustrate the principle.

Examples

Examples of the kind of resilient services we envision are network synchronization primitives or a network virtual file system. The techniques discussed below can support synchronization primitives like P and V, lock and unlock, and block and wakeup in a resilient fashion on a network. Network virtual file systems which provide directory services and data access services can also be provided in an automated and resilient fashion. The network virtual file system would appear to be a single file system to the user, but would in fact be dispersed over a large number of possibly heterogeneous hosts on a packet switched network.

Related Work in Distributed Systems

There are two main problems that are addressed by the technique we are presenting here: synchronization of the users of the service and the resiliency of the service. Other researchers have proposed techniques to achieve the synchronization but haven't treated the resiliency issue carefully.

Perhaps the first work in this area was by Johnson [1974]. This work concerned the updating of data bases simultaneously maintained at several ARPANET hosts. Johnson proposed that updates to an accounting data base be timestamped by the host which generates the update. The updates are then broadcast to the copies of the data base. The data base managers at each host apply the updates in chronological order, as determined by timestamps. (Ties are broken by an arbitrary ordering of the hosts.) Johnson's model introduces the problem that during some time interval the copies may be mutually inconsistent due to message delays, etc. Since this system is intended for an accounting file, sharp restrictions in the number and kind of operations can be made. For example, updates are restricted to increments and decrements of single fields. The support of arbitrary operations on multiple fields is not addressed. From the resiliency standpoint, it is difficult to ensure that an n-host criterion has been met or that all copies of the data base will eventually receive all of the updates.

Bunch [1975] attempted to avoid some of the difficulties of Johnson's scheme by introducing a central name (sequence number) generator. This approach has the problem of introducing a potential bottleneck. Grapa [1975] was able to partially avoid this problem in his "reservation center" model. Grapa's model is somewhat more general than either the Bunch or Johnson model and in a sense includes them as limiting cases.

Despite the fact that none of these models treat the resiliency issues (they were never really intended to), there are also several problems that might be encountered in more general data base environments. We have already mentioned the problem that for some time interval the data base may be inconsistent. This may cause problems for some applications. Also, an update operation on one field may use values of other fields to compute the new value (in an irreversible manner). In this case, the Johnson and Grapa models must include a time delay before applying the updates to guarantee that there are no delayed updates with earlier timestamps than those already received. Similarly, it is difficult for these models to provide a quick response time for updates that modify multiple fields. The technique we describe here avoids these problems and provides the minimum response time allowed by the n-host resiliency criterion while requiring a somewhat more complex mechanism.

Techniques for a Resilient Service

As was indicated above, we are interested in a method by which we can provide resilient support for some distributed resource sharing activity. For purposes of illustration, assume some sort of data base (in the general sense) which is being read and modified by a group of network users. (Examples of network data bases would include data bases as small as a single integer variable used for sequencing network-wide operations to a queue of jobs for an automated load sharing service to a multi-million byte JOPS data base being simultaneously shared by several commands in a crisis.) Consider, at least for purposes of description, that there is a set of server hosts which do nothing but perform updates and mediate the synchronization of updates generated by user processes. (This may appear to be somewhat excessive for the practical case; but if one is really concerned about having a reliable service, it is unwise to make it susceptible to the kind of environment found in the typical application host. However, there is nothing about this scheme that requires that the synchronizing function be in a devoted host.) One of the hosts of this set is designated as the primary and the rest are backups. The backups are ordered in a linear fashion. We will discuss recovery schemes in a subsequent section. For now, let us consider the resilient message flow when the set of server hosts are functioning correctly.

Update operations may be sent to the primary or to any backup. The user process then blocks. It waits for either a response from the service or a timeout indicating that the message has been lost and should be retransmitted.

For the purposes of this discussion we will ignore, to some extent, the details of the end-to-end transmission. Some of the ACK's

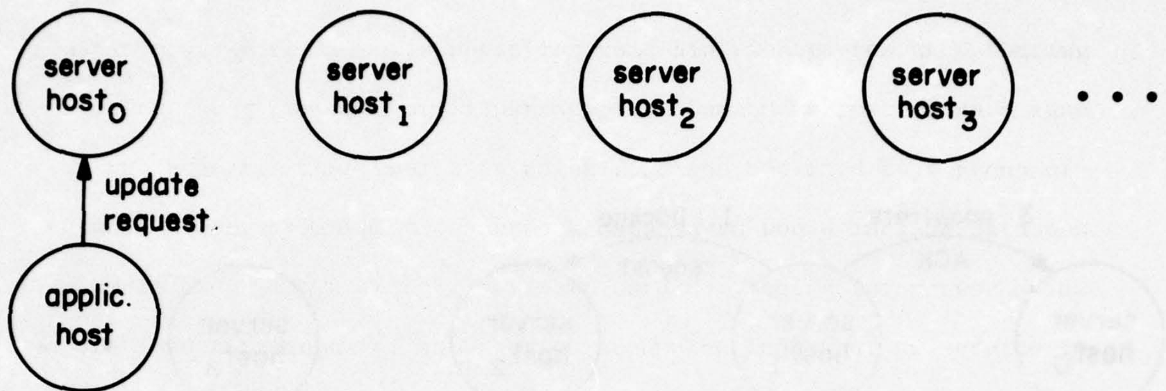
and timeouts mentioned below may be provided by an end-to-end protocol such as those described in Cerf and Kahn [1974] and Cerf et al. [1975]. In addition, the communication between the user and the service could be a single message connection to the service. Such a connection would take more than one message to convince both sides that no messages have been lost or duplicated [Belsnes, 1975]. However, for our purposes we are mainly interested in the delays incurred. Although multiple parallel messages may be generated, the number of sequential message delays will be inherent to any system performing this service.

The Chained Model

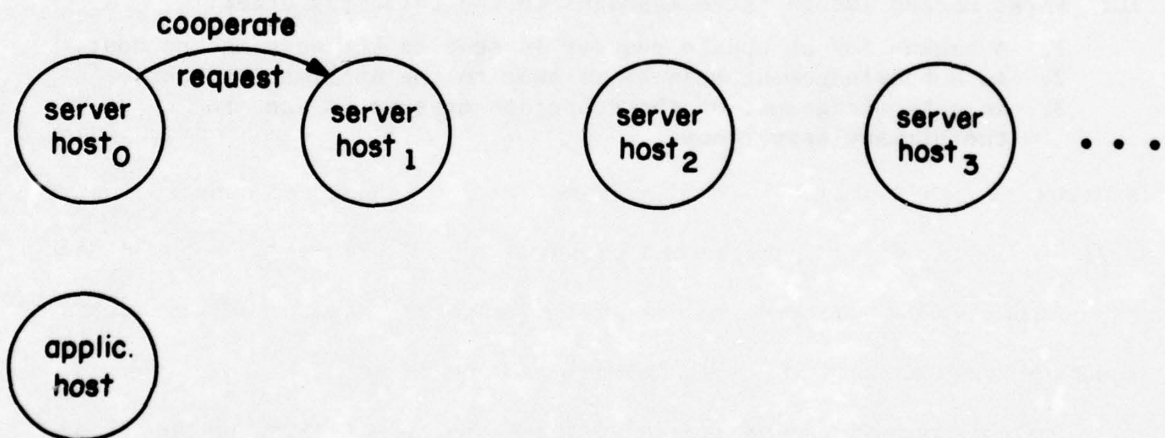
Dedicated servers. Figure 1 shows the message flow for an update operation which has been transmitted to the primary server host of a data base. The first network message delay is incurred in figure 1a. The application host transmits the update to the primary server host.

The second network message delay is incurred in figure 1b. The primary server host requests cooperation in executing the update operation from the first backup server host. The first backup server host will perform the same update. The backup host will be expected to issue the update ACK message to the application host.

In figure 1c the third network message delay is incurred. Three messages are transmitted by the first backup server host. In terms of network delay, these messages are essentially simultaneously transmitted. Small improvements in resiliency can be achieved by issuing them in the designated order. First, the backup server host passes a backup update message to the next backup server host. At this time only two server hosts, the primary and the first backup, have



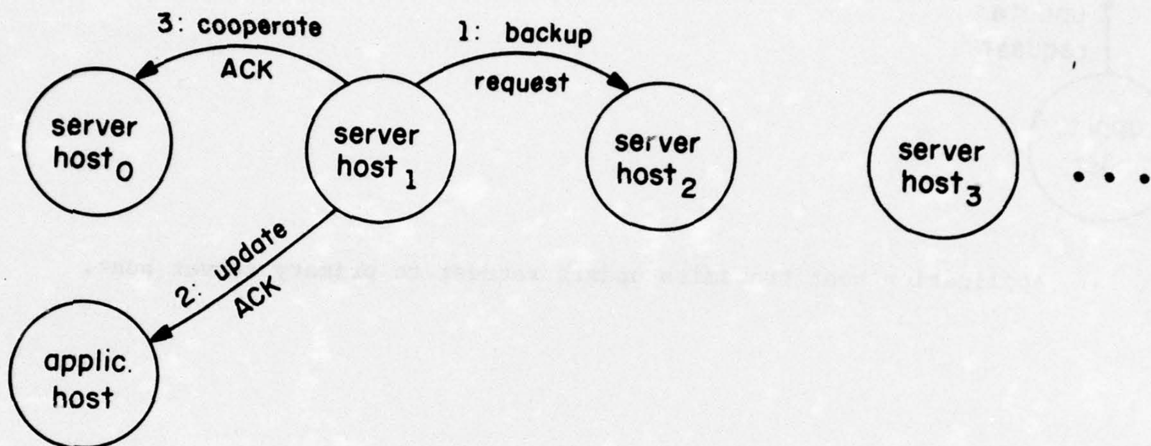
1a: Application host transmits update request to primary server host.



1b: Primary server host requests cooperation from the first backup in executing the update request.

Figure 1

Update request sent to a primary server host



1C: First backup issues three messages in the following order:

1. A backup for an update request is sent to the next backup host.
2. An acknowledgement message is sent to the application host.
3. An acknowledgement of the cooperate message is sent to the primary server host.

Figure 1 (continued)

Update request sent to a primary server host

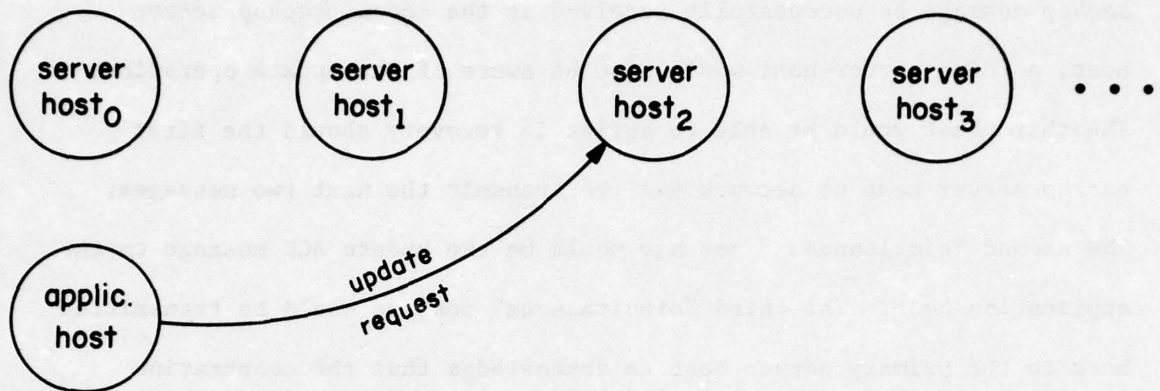
positive knowledge of the existence of the update operation. Should the backup message be successfully received at the second backup server host, a third server host would also be aware of the update operation. The third host would be able to assist in recovery should the first backup server host or network fail to transmit the next two messages. The second "simultaneous" message would be the update ACK message to the application host. The third "simultaneous" message would be transmitted back to the primary server host to acknowledge that the cooperation request on an update operation has been received.

Once the primary server host has received the cooperation acknowledgement, it is certain that the two-host resiliency criterion has been met. The primary may now apply the update to his copy of the data base. Similarly, once the application host has received the update ACK message it is also certain that the two-host resiliency criterion has been met. Should the primary server host fail to receive the cooperation acknowledgement, appropriate retry and recovery techniques will be initiated.

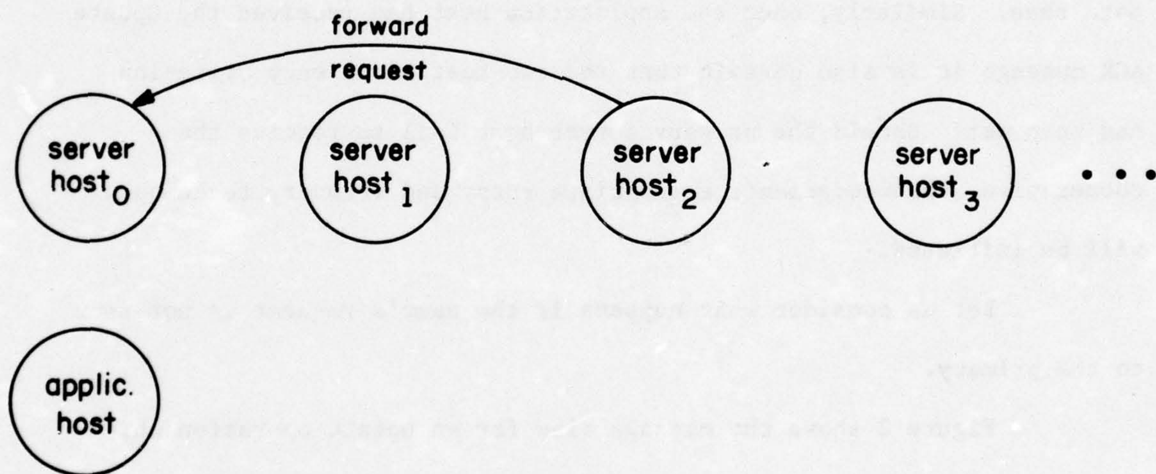
Let us consider what happens if the user's request is not sent to the primary.

Figure 2 shows the message flow for an update operation which has been transmitted to a backup server host. The first network message delay is incurred in figure 2a. The application host transmits the update to a backup server host.

The second network delay is incurred in figure 2b. The backup server host forwards the update operation to the primary server host. The application hosts have no knowledge of the ordering of server hosts. However, each of the server hosts is assumed to have explicit knowledge



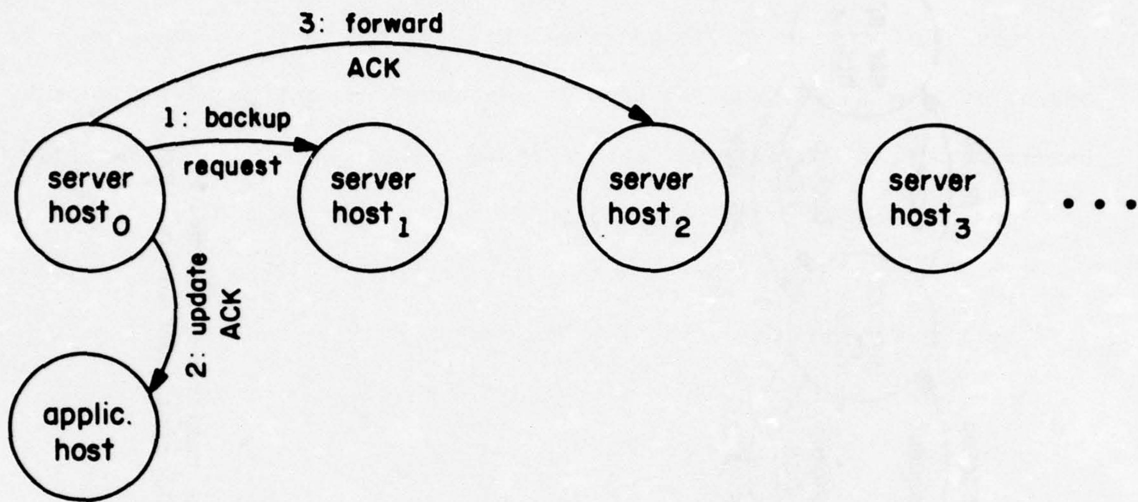
2a: Application host transmits update request to a backup server.



2b: Backup host forwards update request to the primary host.

Figure 2

Update request sent to a backup server host



2c: Primary host issues three messages in the following order:

1. A backup for an update request is sent to the first backup host.
2. An acknowledgement message is sent to the application host.
3. An acknowledgement of the forwarding message is sent to the forwarding backup host.

Figure 2 (continued)

Update request sent to a backup server host

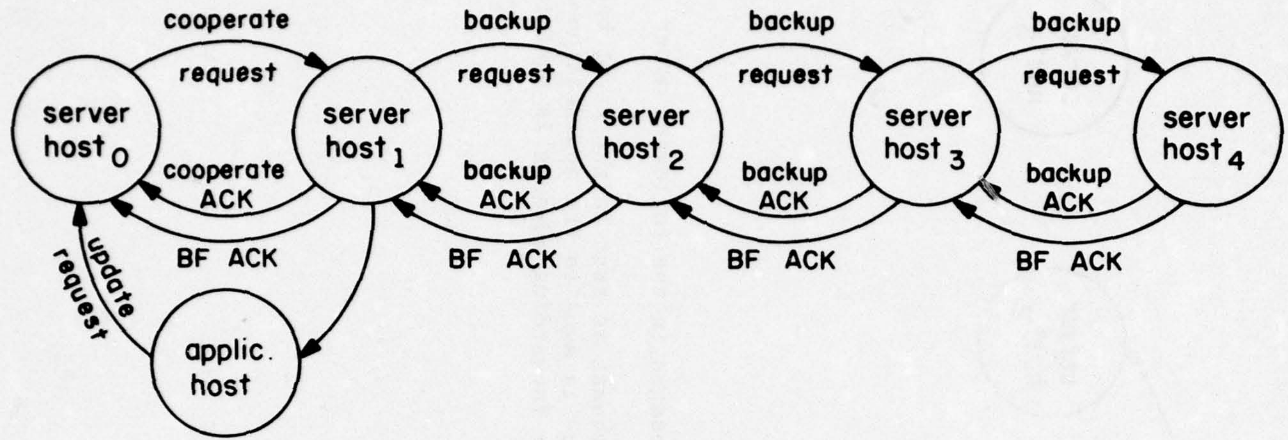


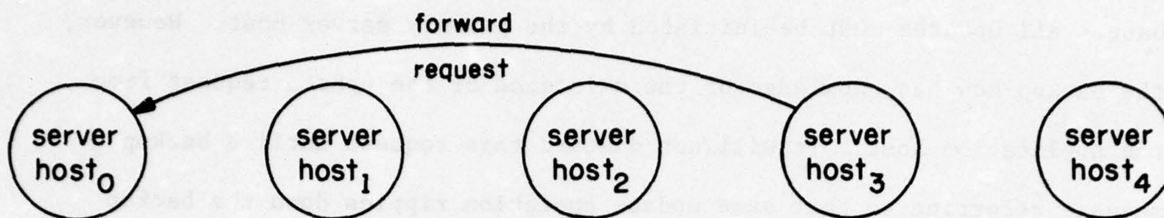
Figure 3

Summary of the message flow for the resiliency scheme.
(BF ACK refers to the Backup Forwarded ACK mentioned in the text.)

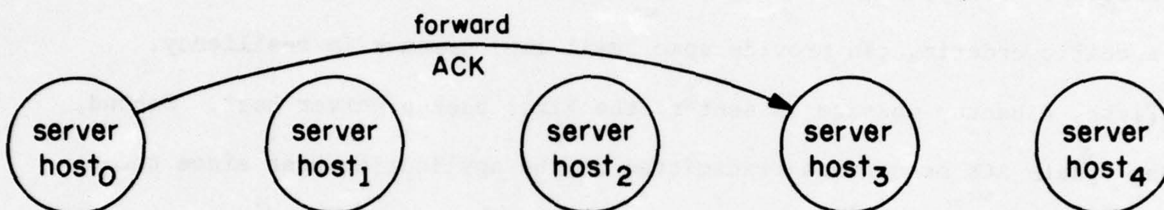
of the ordering. The backup server host performs no updates on the data base. All updates must be initiated by the primary server host. However, the backup now has knowledge of the existence of the update request from the application host. It will not discard this request until a backup message referring to that same update operation ripples down the backup chain and through it.

In figure 2c the third network message delay is incurred. Three messages are transmitted by the primary server host. As was the case previously, these messages are essentially simultaneous but a specific ordering can provide some small improvements in resiliency. First, a backup message is sent to the first backup server host. Second, an update ACK message is transmitted to the application host since the two-host criteria has now been met. Third, a forward message acknowledgment is transmitted to the forwarding backup host. The message flow for the chained scheme is summarized in figure 3.

Participating servers. In a service environment where there is no special set of hosts dedicated to the service, updates from a user on one of the hosts participating in the service will only experience two network delays as opposed to the three found in the dedicated host case. Figure 4 shows that the first delay is generated when the host in which the update was generated sends the update to the primary as a forward request. (Note that since members of the service will most likely maintain the necessary connections among each other, many of the single message connection difficulties can be avoided in this case.) The second delay is incurred when the primary responds with a forward ACK message to the originating backup host. The primary also sends the backup request to the first backup server. From this point on, the procedure is identical to the dedicated server scheme.

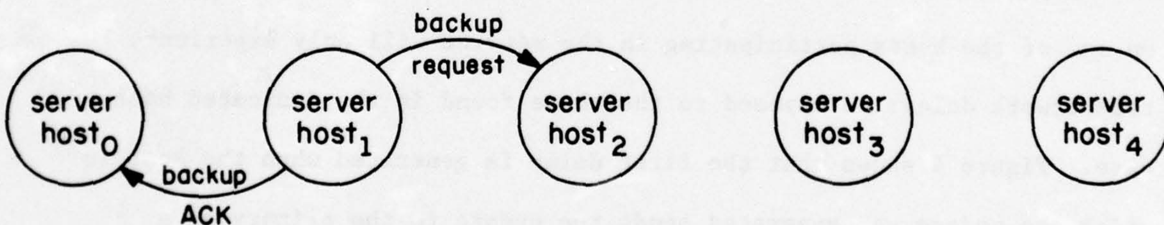


4a: An update request generated by the third host journalizes the request and sends a forward request message to the primary.



4b: The primary records the update. The two host criteria has now been fulfilled. The primary sends two messages:

1. a forward acknowledgement back to the third host.
2. a backup request to next backup host.



4c: The next backup host records the update and sends a backup request to next server and acknowledges the one he received.

Figure 4

Application of the Resiliency Scheme for Undedicated Hosts

A Broadcast Model

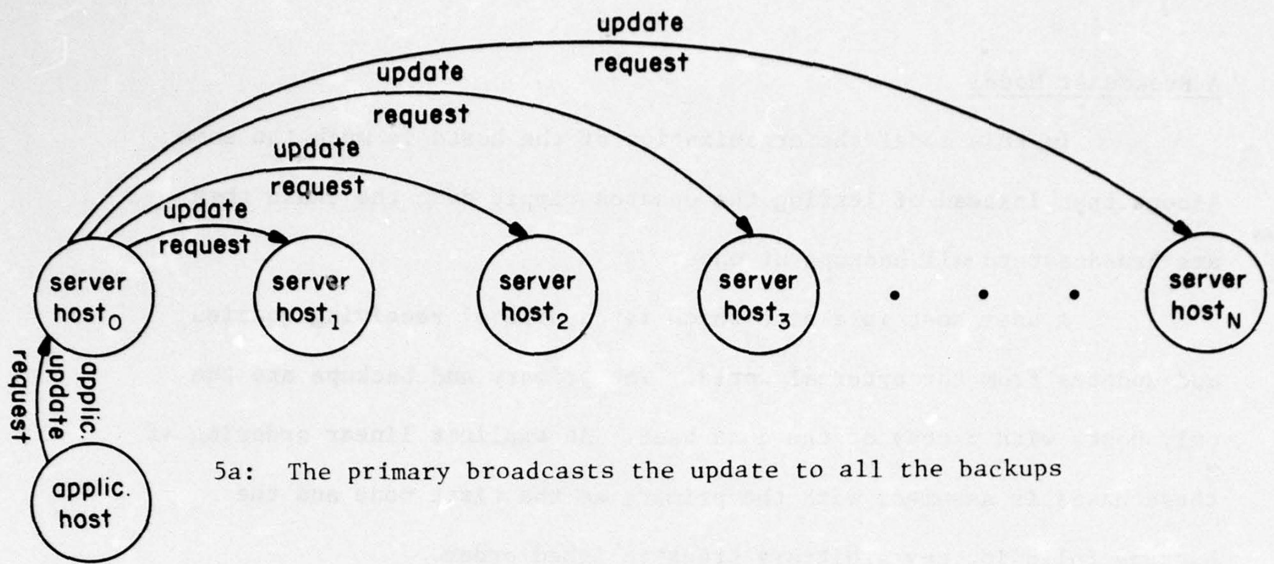
In this model the organization of the hosts is much the same except that instead of letting the updates ripple down the chain they are broadcast to all backups at once.

A user host is a site which is capable of receiving queries and updates from the external world. The primary and backups are the only hosts with a copy of the data base. An explicit linear ordering of these hosts is assumed, with the primary as the first node and the backups following any arbitrary preestablished order.

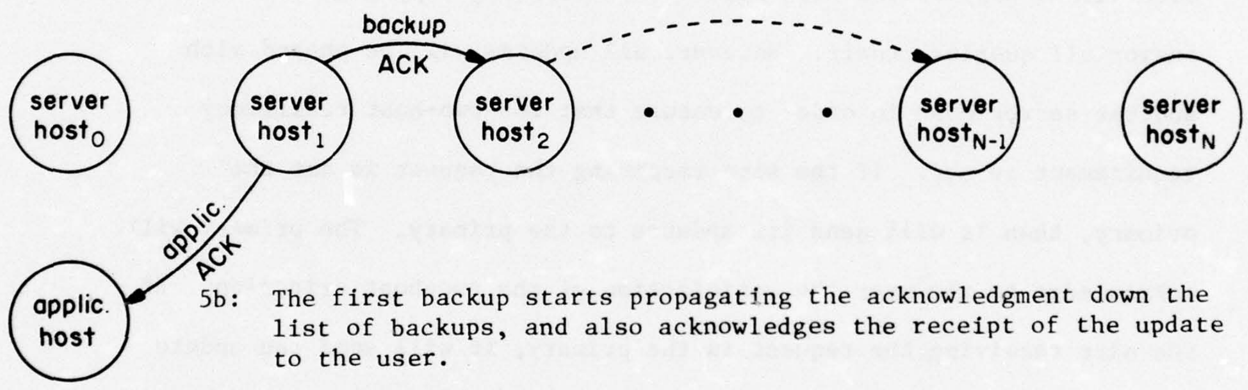
Communication with the user host takes place just as in the chained model above. The user host sends its updates or queries to any site with a copy of the data base. The receiving copy will try to answer all queries itself. However, all updates will be shared with another server host in order to ensure that the two-host resiliency requirement is met. If the site receiving the request is not the primary, then it will send its updates to the primary. The primary will acknowledge to the user the satisfaction of the two-host criterion. If the site receiving the request is the primary, it will send the update to one of the backups (probably the next one in the linear order) asking it to cooperate and send the acknowledgment to the user.

In either of these cases, the primary ends up with the update. Then a second stage of the update is initiated: its actual application to all data base copies.

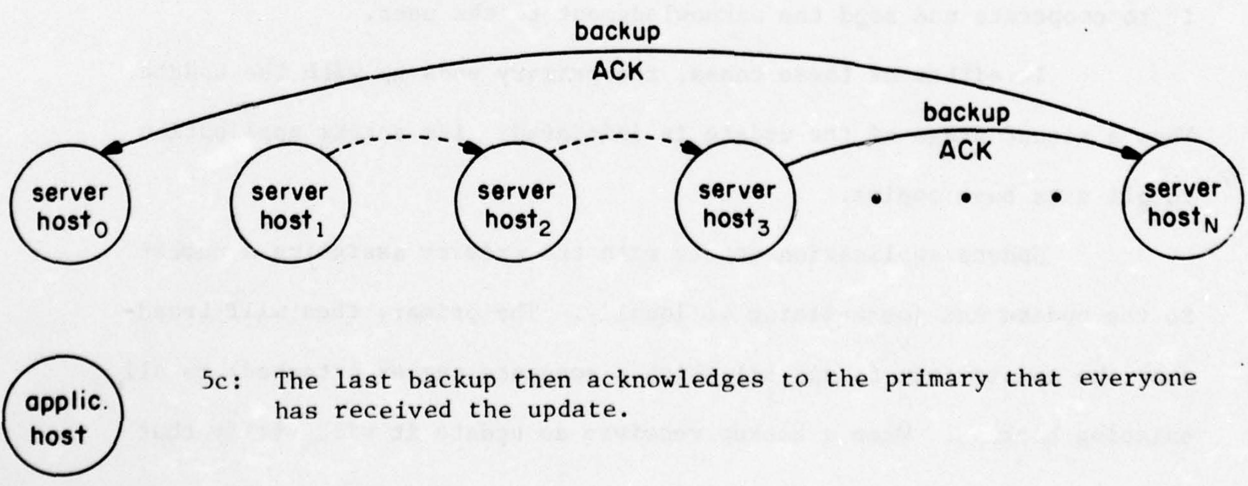
Update application starts with the primary assigning a number to the update and journalizing it locally. The primary then will broadcast the update (see figure 5a) (with a sequence number attached) to all existing backups. When a backup receives an update it will verify that



5a: The primary broadcasts the update to all the backups



5b: The first backup starts propagating the acknowledgment down the list of backups, and also acknowledges the receipt of the update to the user.



5c: The last backup then acknowledges to the primary that everyone has received the update.

Figure 5. The Broadcast Model

the update number is sequentially correct. If this is the case it will initiate the local application.

Upon receipt of an update, the first backup in the linear order is expected to send an acknowledgment (which includes the update's number) to the host that follows it in the given "linear" order. (See figure 5b) All other backups enter a passive state until the acknowledgment gets to them. At that time they will simply pass the acknowledgment to the next backup. The last backup sends an acknowledgement to the primary. This last acknowledgment indicates that the update has been received at all copies of the data base and the operation is complete (figure 5c).

n-Host Resiliency for n > 2

Consider n-host resiliency for arbitrary n. In this case backup n-2 will be responsible for sending the acknowledgment to the user if one user request was forwarded to the primary by backup n-1 or greater. Otherwise, backup n-1 will send the user acknowledgment.

There are two properties of these schemes that should be noted. First, n nearly simultaneous host failures during a small critical interval are required to disrupt the scheme. Second, regardless of where the user process sends the update request in an n-host resilient system, he will get a response in n+1 message delay times. If the synchronizing scheme is moved into the application hosts, this delay can be cut to n message times. (For large n, there are m-way branching strategies that can reduce the delay to the order of log m^n. This issue is of no practical importance since, as will be shown, n=2 is sufficient for extraordinarily high resiliency!)

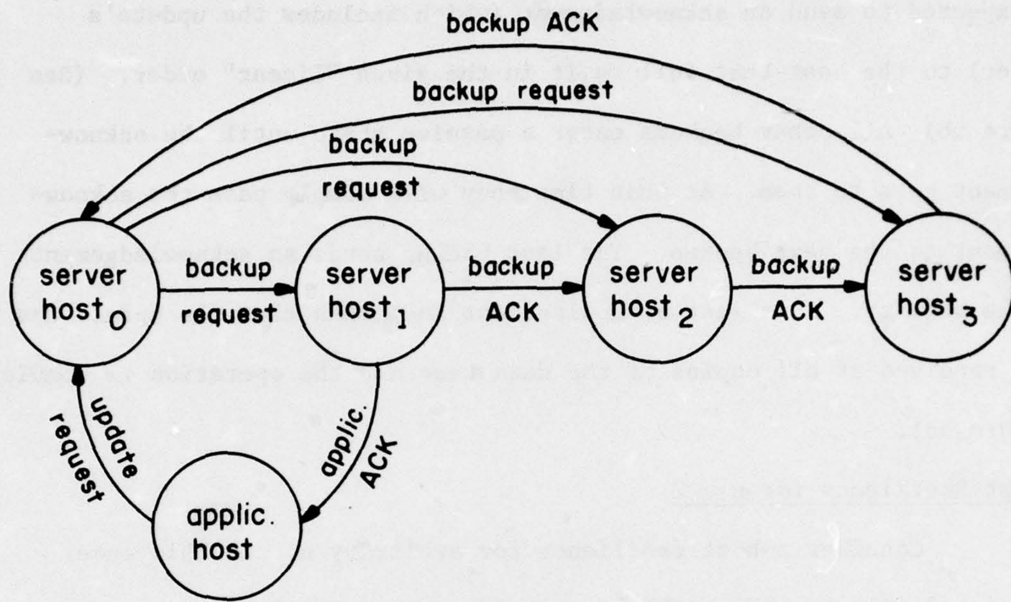


Figure 6. Summary of the message flow for the broadcast resiliency scheme.

Adequacy of Two-Host Resiliency

The two-host criterion does not, of course, guarantee resiliency if two critical hosts fail more or less simultaneously. Suppose, for example, that a user request is sent to the primary, which sends it to the first backup. Then, after the acknowledgment to the user is sent, but before the second backup successfully receives the message, both primary and first backup fail. The probability that this will happen is very small. To quantify this, suppose that host failures are random with mean time between failures F. Then, if one site fails at a critical time, the probability that the second site will also fail within the next t seconds is

$$P_f = 1 - \exp(-t/F),$$

where F must also be expressed in seconds. Thus, if F is 24 hours and t is 1 second, $P_f = 10^{-5}$, and if the primary fails, say, only once a day at a critical time, such double failures will occur only about every 10^5 days, or about 300 years. Another point must be noted. To destroy resiliency, not only must both sites fail, but because of the message ordering (figures 1 and 2) the message to the second backup must also be lost. It is clear that the likelihood of these nearly simultaneous, multiple failures is essentially negligible. Thus three-host or greater resiliencies are not required for any realistic applications. The main problem in maintaining the two-host resiliency scheme is that of providing a sufficient number of server hosts in the scheme so that the user has a good chance of finding at least two in service.

Resilient Service Down Time

In order to get some idea of how many server hosts may be required to achieve a given degree of reliability, let us consider the two-host resiliency case in more detail.

In this scheme, at least two server hosts must be up when the user request is entered into the system. We wish to estimate the probability that the scheme fails because two hosts are not available. Suppose that host failures occur independently and randomly. Let a denote the availability of any one server host. (Availability is the fraction of time that the host is available; alternatively, it can be considered as the probability that a user request finds the host up.) If there are N server hosts, the probability that none are up is

$$P_o = (1 - a)^N,$$

The probability that exactly one is up is

$$P_1 = Na(1 - a)^{N-1}.$$

Therefore the probability that the scheme fails because the requisite two hosts are not available is

$$P = P_o + P_1 = (1-a)^{N-1}(1-a + Na).$$

A family of service availability curves is plotted in figure 7. Individual host availability is plotted on the x-axis as hours of down time per day (1-(hours down/24) = a). P is plotted on the y-axis as service downtime per day (24xP = hours down per day, 24x60xP = minutes down per day, etc.). Average host downtime can be read as resilient service down time by using the lines plotted for the number of server hosts (N=2 to 10). For example, if the average down time per host per day is two hours, then table 1 shows resilient service down time as a function of N.

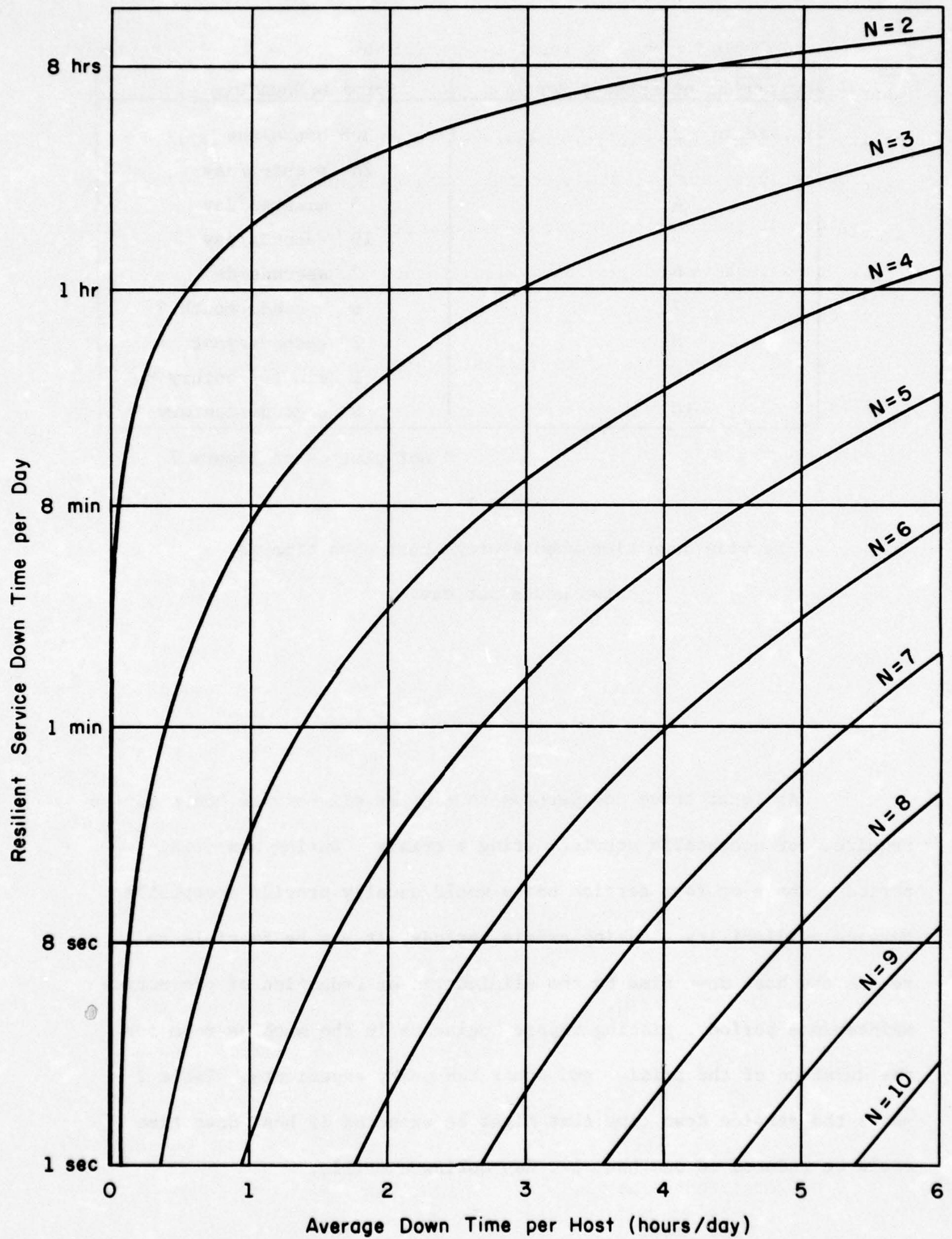


Figure 7

Service Down Time vs. Host Down Time and Number of Hosts

Number of Service Hosts	Service Down Time
2	3.8 hours/day
3	28 minutes/day
4	3 minutes/day
5	19 seconds/day
6	2 seconds/day
7	6 seconds/month *
8	7 seconds/year *
9	1 minute/century *
10	6 seconds/century *

* not plotted on figure 7

Table 1

Service down time when average host down time is two hours per day.

At least three and perhaps as many as six service hosts may be required for acceptable service during a crisis. During non-crisis periods, three or four service hosts would usually provide acceptable service availability. During crisis periods, it may be feasible to reduce the host down time by the elimination or reduction of preventive maintenance periods, placing repair engineers in the machine room for the duration of the crisis, and other temporary expedients. Table 2 shows the service down time that might be expected if host down time could be reduced to one hour per day during a crisis.

Number of Server Hosts	Service Down Time
2	2 hours/day
3	7 minutes/day
4	24 seconds/day
5	1 second/day
6	23 seconds/year *
7	1 second/year *
8	5 seconds/century *
9	2 seconds/thousand years *
10	1 second/ten thousand years *

* Not plotted in figure 7

Table 2

Service down time when average host down time is one hour per day

Using tables 1 and 2, it appears that three or four service hosts could provide both acceptable non-crisis performance and good crisis performance for those applications which require resiliency. Judicious management of deferable down time is required during a crisis to reduce the need for servers.

Removing the Two-Host Resiliency Criterion

Note that the probabilities computed above do not necessarily reflect total system reliability. The system may be kept working (but without the safeguard of the 2-host resiliency criterion) when only one server site is up. To get some feel for how often an undetected error is really likely to occur, the rate of occurrence of errors must then be taken into account as well as the expected down time of the resiliency scheme. For example, suppose that the resiliency safeguard is inoperative for one T-minute period per day. P_1, the probability that exactly one host is up, is used to compute T. Assume that the errors that would

normally be detected by the resiliency scheme occur randomly. (Thus the occurrence of errors forms a Poisson process. See CAC Document Number 181, CCTC-WAD Document Number 6501 for further discussion and references on Poisson processes.) Let the mean time between these errors be m. Then the probability P_e that an error occurs during the downtime is

$$P_e = 1 - \exp(-T/m).$$

Table 3 shows P_e for m values of 15 min., 1 hr., 4 hrs., 1 day, 1 week, and 1 month between errors. At present we do not have a good idea of what is an appropriate value for m. Hence we have tabulated a broad range of conceivable values.

Number of service hosts	P_e					
	m = 15 min	1 hr	4 hrs	1 day	1 wk	1 mo
2	1.0	.97	.60	.14	.02	.0007
3	.84	.36	.11	.02	.003	.000 09
4	.18	.05	.01	.002	.0003	.000 01
5	.02	.005	.001	.000 2	.000 03	.000 001
6	.002	.000 5	.000 1	.000 02	.000 003	.
7	.0002	.000 05	.000 01	.000 002	.	.
8	.000 02	.000 005	.000 001	.	.	.
9	.000 002	.	.	.	.	.
10	.	.	.	.	.	.

Table 3

P_e for average host down time of two hours per day and six values of m

Note in table 3 that if m is one hour and there are only two hosts, it is virtually certain that an error occurs on any given day. This corresponds to a single master and single backup strategy. If two backups are provided (N=3), an error occurs about once every three days. If undetected

errors are acceptable on the order of once a year (about the reliability of ARPANET communications technology), then four (m = 1 day), five (m = 1 to 4 hrs), or six (m = 15 min) service hosts must be supplied. As a rough rule of thumb, for any fixed value of m and host down time, each server host added decreases the rate of occurrence of undetected errors by a factor 10. (Note: experience with production use of the ARPANET indicates that m < 1 day is a reasonable assumption. Lost messages and hung connections, for example, have been frequently observed at several per day. These errors at least bound the value of m from above. Improvements in the resiliency of end-to-end protocols (a host problem - not a subnet problem) in ARPA-like networks are needed. These could significantly reduce the contribution of lost message and similar errors to m.)

Failure Detection and Recovery

Resiliency is achieved in these schemes by a combination of techniques. The basic organization of the scheme provides the skeleton on which to construct the resilient service. The additional mechanisms used for a particular application will depend heavily on the degree of resiliency required. This additional resiliency is gained by applying a combination of sequence numbering schemes and ACK and time-out mechanisms. For instance, to get two-host resiliency for updates being propagated in the chained model, a "Backup forwarded ACK" is used in the following way:

When a backup server host receives the "backup ACK" corresponding to the backup message sent to its right-hand neighbor (see figure 3), it sends a "backup forwarded ACK" to its left-hand neighbor. This assures that neighbor that the update has progressed to at least the second backup beyond itself.

Lost and duplicate message detection. For most applications, one sequence number scheme can be applied to the messages to detect lost or duplicate messages. A second sequence number scheme can be applied to the requests themselves. This allows proper recovery in the event of failures. It also defines the order in which requests will be applied to the data base.

As an example, consider how the sequence numbers can be used to detect lost messages in the broadcast model. Upon arrival of message m at a backup, the backup will first check whether all prior messages have been successfully received. If this is not the case, then the backup may assume that a message has been lost and ask for the retransmission of missing messages (probably by the primary). In the meantime, update m is stored in a "wait list" until the problem can be cleared up. If, on the other hand, all prior messages have been received, then the backup can proceed with the application of update m and is ready for the acknowledgment propagation. If the backup is the first host in the linear list, then it initiates this process by sending the acknowledgment shown in figure 5b. Otherwise a waiting period (controlled by a timeout) is started.

When an acknowledgment with sequence number m reaches a backup, it first checks to see if the acknowledgment corresponds to a known, processed message. If so, it simply adds its own acknowledgment by propagating the acknowledgment to the next host in the linear list. However, if the acknowledgment corresponds to an unknown message, the backup can assume that message m is lost and ask for the retransmission of that message (from the primary or probably from any of the backups upstream in the list if the backups journalize messages). In this case

the propagation of the acknowledgment should be suspended, but a first-to-acknowledge flag should be set to indicate to the backup that it should restart the propagation of the acknowledgment as soon as message m arrives. A similar action is taken when an acknowledgment for a known but unprocessed message arrives (i.e. one waiting in the waitlist for the intermediate updates). In this case the backup will also set the first-to-acknowledge flag and suspend the propagation of the acknowledgment until the update can be extracted from the waitlist.

Multiple failures could occur; for example, a message asking for the retransmission of a lost message could be lost. Therefore, a pair of timeouts are also included. Whenever the retransmission of an update is solicited, retransmission timeout is set. Most likely the action that should be taken when retransmission timeout expires is to repeat the appeal for retransmission.

As mentioned before, acknowledgment timeout is set to wait for the acknowledgment of an accepted (correct sequence) message. All but the first backup in the list will have a similar acknowledgment timeout mechanism. However, the length of the timeout could be larger for backups further from the primary. When an acknowledgment timeout expires, a search for the acknowledgment will be started by sending an "ACK SEARCH" message to the next backup in the linear order. Reception of an "ACK SEARCH" message for an unknown update will also help to detect lost messages. Reception of an ACK SEARCH for the known message will produce one of the following responses:

1) the retransmission of the acknowledgment if the acknowledgment was sent and lost in transmission,
2) propagation of the "ACK SEARCH" message in the direction contrary to that of the propagation of acknowledgments, or

3) no action because the corresponding message is either in the wait list or was lost and is already going to be retransmitted.

The detection of failures may be accomplished in a variety of ways. Clearly, the same timeouts that were discussed above to detect lost messages may be used to detect a host failure during the course of performing a request. There may be relatively long idle periods between requests. As a result, long delays may be incurred by a request which discovers a failure and must wait for recovery to be initiated and completed. For some applications, it may be useful to have a low level "are you alive" protocol among the members of the chain. Otherwise, the failure will not be detected until the next request is sent.

Host failure recovery. Although some applications may require slight variations, the basic mechanism for recovering from host failures is quite simple. Most of the variations will center around how a host determines that another host has failed and the degree of resiliency desired. Let us assume that one of the backup hosts has determined that one of his neighbors has failed. What needs to be done? Clearly, the dead host must be removed from the set of server hosts by notifying the rest of the server hosts of the change. Note that this notification is identical to updating a data base using the participating host scheme. Only in this case the data base being operated on contains the state of the system as a whole.

Therefore, the host who detects the failure formulates an update request and forwards it to the primary. The sequence of events is then identical to that for the participating servers (as shown in figure 4) with one exception. When the update indicating the change in structure ripples down to the host who initiated the update, it will

attempt to establish communication with the next active host in the chain, and then forward the update on. (If desired, the host could attempt to establish communication while it was waiting for the update to ripple down.)

When a host comes up after a crash it will send an "initialization request" to some host in the service. If that host is not the primary, it will forward the request on to the primary. The primary will add the new host to its tables and pass a message down the chain indicating that the other hosts in the service should add the new host to their tables. The primary will also assign one host (possibly the last one in the chain) to bring the newcomer up to date. How the new host is brought up to date depends on the application. It may be done by transferring to that host the journal of all updates since the host went down. It may require transferring the entire data base.

Recovery from apparent network partition. Operation during a network partition is difficult to handle. For a majority of applications it will probably consist of providing very degraded service. To give the reader an idea of the complexity of providing service across partitions, let us consider the case where as close to full service as possible is provided. First, each side must organize itself into a resilient system with a primary. When the partition is repaired, each side must have a way to rectify the existence of the two primaries. It must be possible to restore the data base to the state it was just before the partition and to journalize all updates made during the partition. When the partition is repaired, the update journals of both sides must be merged according to the chronological order in which the updates were generated. If the same event has been observed and entered by groups on

both sides of the partition, the journals may contain duplicate entries. Duplicates must be recognized and all but one discarded. It should be further noted that answers to queries submitted to a partitioned subset may be inconsistent with answers given queries after the partition has been repaired. Since network partitions are so difficult to handle, it is highly desirable that they be very infrequent.

It may appear on the surface that this problem is easily solved by proper network topology. To a degree this is the case. But the solution is also highly dependent on how much information the subnet returns about failures. Suppose the communications subnet only indicated whether or not it could deliver a message. Then every apparent failure would have to be treated as a possible partition, and the rather expensive partitioned mode of operation would have to be initiated. However, if the subnet distinguishes between "I was unable to deliver the message" and "I got the message to the destination node, but the host is not servicing the interface"; it would be possible to classify many of the failures as host failures and take a less expensive recovery procedure. There would still be a small group of failures that would have to be treated as partitions until communications were restored and it was determined whether or not a partition had actually occurred.

Let us now consider the problem of an apparent network partition. In this case the subnet has notified the host that it could not deliver a message. For some reason the message did not get as far as the destination node. Perhaps, after some number of retries, this host has a reasonable suspicion that the network has partitioned. It will then broadcast "are you alive" messages to everyone in the service. After some time period, it will assume that all responses that can arrive have arrived. It will then modify its structure tables according

to the responses, and send messages to the other members with which it can communicate to do the same. If this fragment of the partition has the old primary in it, the primary will coordinate partitioned mode operation. If not, a new primary may be chosen by an appropriate algorithm, depending on the level of partitioned service that is desired. The service then enters partitioned operation mode. As mentioned above, what the service does in this case will depend heavily on the application, the degree of resiliency desired, and the frequency of partition. In the general case, two or more partitions can produce incompatible states that cannot be joined later. Thus, the operation of a service, while the network is partitioned, can easily become quite complicated and expensive.

Let us consider what kinds of partitioned serviced can be provided without requiring the considerable overhead of re-joining the data bases after a partition has been repaired. If the application is not very critical, then the simplest solution is to not allow any updates to be processed at all. A less severe scheme would be to allow the side of the partition with a majority (possibly weighted) of the servers to continue operation, and the minority, in essence, to be dormant. It would be possible to allow the minority side to continue processing queries with the warning that the data may not be current.

Numerous variations on these kinds of schemes can easily be generated to provide almost any level of service commensurate with the constraints of cost and reliability.

Overhead Due to Resiliency

Resiliency adds negligible overhead to a distributed backup scheme in terms of network traffic, storage costs, and host processing.

Network traffic. In order to estimate the traffic overhead of resiliency, we will compare a multiple backup scheme without resiliency to one with resiliency. Assume that N server hosts are involved and that one is a known primary which receives all queries. The absolute minimum traffic required is N + 1 messages. The user will send an update message to the primary and will receive a result message (two messages). The primary could just blindly broadcast the update to the N-1 backups without acknowledgment (N-1 messages). A no-acknowledgment strategy is strongly contraindicated. Without an acknowledgment, frequently occurring situations may cause complete failure to the backup system without notice. For example, the service program which supports the backup may not have been initiated in the backup host. The service program may have been temporarily preempted. The operating system may have jumbled the data (an observed GCOS phenomenon). The service program may have exhausted its journalization space. The network may have dropped the message. There are many other examples. Thus, it appears that a production service should always require acknowledgments to assure that the backup system is functioning. If acknowledgments are included then the total number of messages required is 2N. The message traffic in the resilient broadcast scheme (figure 6) is also 2N. Resiliency is obtained by restructuring the message flow. (A side benefit: the resilient message flow scheme reduces the traffic at the primary host.) Hence there is no traffic burden added to a production system by the resiliency scheme.

Storage costs. It is obvious that the service induced storage required by a non-resilient scheme is the same as for a resilient scheme. If, for example, the non-resilient backup scheme stores a data base off-line and only journalizes updates, the same could be done in a resilient

scheme. The only additional data needed by the resilient scheme are a few state tables which monitor the status of the service hosts. These tables should have negligible storage impact.

Processing costs. Like storage costs, the processing costs associated with handling an update message are identical whether the system is resilient or not. Some additional processing is required to update state tables, to maintain timeouts, and to check sequence numbers. This processing is negligible compared to the cost of update processing. The major resiliency cost is increased complexity of the processing algorithm rather than increased use of the processor.

Comparison of the Models

The nature of the differences in the chained and broadcast models are such that no clear cut choice between them can be made without first considering the application and the service requirements on that application. In this section we describe the trade-offs and how they affect the service.

Complexity. Although the chained model is probably the simpler of the two, the increased complexity of the broadcast model (primarily in fault situations) is not severe. However, the number of messages exchanged to complete the same operation is 50% less in the broadcast model. This is because this model uses one acknowledgment while the chained scheme uses two (the backup ACK and the backup forwarded ACK). However, if the implementation of the chained model used the message acknowledgments of an end-to-end protocol as the backup ACK's then the chained model would be in line with the broadcast model.

Timeliness. In general, the backups in the broadcast model will be more up to date. Broadcasting the updates means that all backups are updated with the same delay experienced by the first backup

in the chained model. However, this advantage is small. The update message propagation time down even a long chain in an ARPA-like network is only a few seconds.

Lost update detection. When an update is lost in the chained model, there are three ways to detect it: 1) the arrival of the next update causes the recipient to notice a missing sequence number, or 2) and 3) the sender times out after it has not received a backup ACK or a backup forwarded ACK within some time limit. When an update is lost in the broadcast model, there are also three ways to detect it: 1) a missing sequence number will be discovered, 2) an ACK timeout will occur or 3) the arrival of an acknowledgment propagating down the list may cause a backup to notice a lost update. One of these schemes may be more resilient in some environments than the other one. Further investigation is necessary to determine the pros and cons of these two techniques.

Primary recovery. Recovery of the primary is simpler in the chained model, primarily because the re-organization is somewhat implied by the structure of the backups. The broadcast model must include some information about the latest updates received, to prevent duplicate assignment of sequence numbers. In the chained model it is only necessary to verify the updates of the first backup to find out what sequence number should be considered by the new primary as the last issued. In the broadcast model the last sequence number that should be considered by the new primary is not a function of the first backup only; it might well be that some other backup has gotten the broadcast of a message that the first backup did not.

Journaling. It is clear that updates should be journalized until an update has been recorded at a particular number of hosts. This

aids in recovery and insures that no messages are lost. However, it is not so clear when it is safe to strip back the journal. In the chained model, the obvious point is upon receipt of the backup forwarded ACK. This means that a host may delete entries from the journal when it is sure the update has propagated one host beyond itself. In the broadcast scheme, it is less obvious. The only "back traffic" is the ACK from the last backup to the primary. The broadcast model has one advantage over the chained model in that the primary does know when an update has propagated all the way down and it is safe to delete entries in the journal. All one would have to add to the broadcast model is a message that was sent down the list of hosts indicating how much of the journal could be deleted. If the same effect was desired in the chained model, this message and one to indicate to the primary that the update had made it to all backups would have to be added. There are still several unanswered questions about the maintenance of these journals, especially when hosts are dead or during network partitions. These need to be studied in more detail and will perhaps be dependent on the nature of the application.

Queries. Up to now we have emphasized the manipulation of updates. We will now say a few words about queries. Queries are expected to be sent to an arbitrary copy of the data base for processing. This copy can be either the primary or any of the backups. In some circumstances it is necessary that the query be answered after a given update has been processed. For applications of this nature, it is probably advisable to qualify the query by a given update that must be processed before it is answered. In general, the broadcast model will keep the backups slightly more up to date as we mentioned above.

Alternative Resource Sharing Strategies

We have proposed the use of a single primary with multiple backups to support resilient resource sharing. The alternative to this approach is to share primary duties among several members of the resource set. This can take the form of designating all members of the resource set as primary or some subset as the group of primaries and another subset as the group of backups. In the case of two-host resiliency, it has been shown that the single primary, multiple backup strategy produces the theoretically minimum message delay that ensures the resiliency criteria have been met.

Let us consider the case where there is more than one primary. In the general case, the primary which receives a service request must synchronize the execution of that service request with all other primaries. Otherwise, the system cannot guarantee that service requests are executed in the same order at all resource sites. (This requirement is essential in the general case. There may be specific applicatons where the nature of the service permits the out of order processing of requests. An example is an inventory system where only increments and decrements to data fields are permitted and where instantaneous consistency of the data base is not a requirement.) The synchronization of multiple processes reduces to the execution of an algorithm in each of the processes that will result in distinguishing one process. The distinguished process then establishes, for example, the order in which operations will be performed, notifies the other primaries of its decision and then relinquishes its distinguished role.

In the single primary case the distinguished resource is designated a priori. Hence, any additional message traffic, processing load, or protocol complexity to distinguish a primary is avoided.

Instead emphasis is placed on electing a new primary should the original primary fail.

An alternative strategy may require all members of a resource set to be primary or only some of those members to be primary. However, the requirement for synchronization tends to increase processing load at each host, message traffic in the communications subnet, and the complexity of the service protocols. At the same time, there is no increase in resiliency or decrease in delay. Thus a multiple primary strategy can never be superior, in the general case, to a single primary strategy. Hence, the single primary, multiple backup strategy, is, in a sense, fundamental to resilient, distributed resource sharing.

Range of Application

The resilient resource sharing strategies discussed above can be applied to a wide range of distributed system services. In particular, the authors have studied the questions of resilient network synchronization, resource directories, data access and load sharing. In all cases the resiliency technique seems to provide a convenient framework to support automated distributed resource sharing.

Synchronization Primitives

The application of the resiliency technique to the support of synchronization primitives is straightforward. Service requests are transmitted to the synchronization service host exactly as shown in figures 1 and 2. The synchronization primitives can be traditional P and V, block and wakeup, lock and unlock, and similar primitives. When a process requests synchronization service (e.g., a P, a lock, or a block) it transmits this primitive request to one of the synchronization service hosts. The acknowledgment returned by a synchronization host will be either a block or proceed message. This tells the requesting process whether it is forbidden to or permitted to enter its critical section. If the process blocks, it may choose to exercise a local system primitive to block its further progress. Alternatively the application process can go blocked waiting for a read on the communications network. In this latter case the read will not be satisfied until a proceed message is received from one of the synchronization service hosts. This proceed message is generated by a synchronization host following the execution of a V, unlock, or wakeup primitive by another process.

Directories and Data Access

In a distributed environment the problem of accessing and updating network virtual file systems and their associated directories is difficult. For example, consider the problem of a single network-wide tree-structured file directory scheme. Each host on the network must be able to determine, in some reasonably transparent fashion, where individual files are stored. If each site in a large network is required to keep the entire directory structure, the cost for updates and synchronization of access to all of those directories (whenever they are updated) would clearly be prohibitive. It is relatively straightforward to use a scheme where the very highest levels of the directory structure are fixed and replicated on all hosts. Alterable directories and files are at lower levels of the tree. A list of potential service hosts is stored at the point where the hierarchy becomes variable. These service hosts are coordinated via the resiliency technique to provide access to files below that point. This approach has the advantage of partitioning the hierarchy in such a way as to minimize the number of hosts required to cooperate in an update.

Load Sharing

Automated load sharing requires that multiple processors be controlled in a resilient and transparent fashion to provide processing services to requesting hosts. The resiliency technique can be applied in a straightforward way to coordinate the offering of that service. Any potential service site can receive a request for service and pass it on to the primary for determination of an optimum processor for the work. Once the task has been successfully forwarded to the primary it would not matter if one of the service hosts involved in the task were to die. Adequate information would be maintained to support the automatic recovery of the service host.

References

Alsberg, P.A. and Day, J.D.
"A Principle of Resilient Sharing of Distributed Resources", submitted to the 2nd International Conference on Software Engineering, October 1976.

Belsnes, Dag
"Single-Message Communication", IEEE Transactions on Communication, Vol. Com-24, No. 2, Feb 1976.

Bunch, Steve
"Automated Backup" in Preliminary Research Study Report, CAC Document 162, May 1975.

Cerf, V.G. and Kahn, R.E.
"A Protocol for Packet Network Intercommunication", IEEE Transactions on Communication, Vol. Com-22, No. 5, May 1974.

Cerf, V., McKenzie, A., Scantlebury, R., and Zimmerman, H.
"A Proposal for an Internetwork End to End Protocol", INWG Note 96, Jul 1975.

Grapa, Enrique
"Thinking Aloud about a Distributed Data Base Model", CAC Dileptus Project Internal Modeling Memo #5, Nov 1975.

Johnson, P.R. and Beeler, M.
"Notes on Distributed Data Bases", Draft, Aug 1974.

Johnson, P.R. and Thomas, R.H.
"The Maintenance of Duplicate Data Bases", RFC 677, Jan 1975.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUC-CAC-DN-76-202	2.	3. Recipient's Accession No.
4. Title and Subtitle Multi-Copy Resiliency Techniques			5. Report Date May 31, 1976 6.
7. Author(s) Peter A. Alsberg et al.			8. Performing Organization Rept. No. CAC #202
9. Performing Organization Name and Address Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. DCA100-75-C-0021
12. Sponsoring Organization Name and Address Command and Control Technical Center WWMCCS ADP Directorate 11440 Isaac Newton Sq., N., Reston, VA 22090			13. Type of Report & Period Covered Research 14.
15. Supplementary Notes			

16. Abstracts

In this paper, we describe a strategy which allows resources to be shared in a resilient manner while minimizing user delay. The strategy described supports two-host resiliency. That is, at least two of the cooperating hosts must simultaneously malfunction while in the process of cooperation, and the malfunction must be of a very restricted form in order for undetectable or unrecoverable failure to occur. Extension to n-host resiliency is also discussed.

17. Key Words and Document Analysis. 17a. Descriptors

network protocol resiliency
distributed data management
resource sharing

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
No Restriction on Distribution Available from the National Technical Information Service, Springfield, Virginia 22151	UNCLASSIFIED	52
	20. Security Class (This Page) UNCLASSIFIED	22. Price

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CAC Document Number 202 CCTC-WAD Document Number 6505	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Multi-Copy Resiliency Techniques		5. TYPE OF REPORT & PERIOD COVERED Research
		6. PERFORMING ORG. REPORT NUMBER CAC #202
7. AUTHOR(s) Peter A. Alsberg et al.		8. CONTRACT OR GRANT NUMBER(s) DCA100-75-C-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center WWMCCS ADP Directorate 11440 Isaac Newton Sq., N., Reston, VA 22090		12. REPORT DATE May 31, 1976
		13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Copies may be obtained from the
National Technical Information Service
Springfield, Virginia 22151

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No restriction on distribution

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

network protocol resiliency

distributed data management

resource sharing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper, we describe a strategy which allows resources to be shared in a resilient manner while minimizing user delay. The strategy described supports two-host resiliency. That is, at least two of the cooperating hosts must simultaneously malfunction while in the process of cooperation, and the malfunction must be of a very restricted form in order for undetectable or unrecoverable failure to occur. Extension to n-host resiliency is also discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)