A PRINCIPLE FOR RESILIENT SHARING OF DISTRIBUTED RESOURCES*

Peter A. Alsberg and John D. Day
Center for Advanced Computation
University of Illinois at Urbana-Champaign

Keywords: resilient protocols, resource sharing, distributed control, distributed computer systems, resilient resource sharing

A technique is described which permits distributed resources to be shared (services to be offered) in a resilient manner. The essence of the technique is to a priori declare one of the server hosts primary and the others backups. Any of the servers can perform the primary duties. Thus the role of primary can migrate around the set of servers. The concept of n-host resiliency is introduced and the error detection and recovery schemes for two-host resiliency are presented. The single primary, multiple backup technique for resource sharing is shown to have minimal delay. In the general case, this is superior to multiple primary techniques.

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 catastropic 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.

Resiliency Criteria. 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. We introduce the

* This work was performed as part of Contract DCA100-75-C-0021 with the Command and Control Technical Center - WWMCCS ADP Directorate of the Defense Communications Agency.

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 ntework 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 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]. Johnson proposed that updates to a 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 then 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. This system was primarily intended for an accounting file, in which updates are restricted to assignments of values to single fields. From the resiliency standpoint, it is difficult to ensure that the n-host criterion has been met and 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 additional problem of introducing a potential bottleneck. Grapa [1975] was able to 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 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. It provides the minimum response time allowed by the n-host resiliency criterion but requires a somewhat more complex mechanism.

A Technique for a Resilient Service

Consider synchronization on a network. The pacing item in a network synchronization operation is network message delay time. Network message delay is on the order of 100 milliseconds. The execution of a process synchronization primitive in the typical single site environment is on the order of .1 to 1 milliseconds. The processing incurred at the site is expected to be the same for both network and local operation. As a result, an appropriate measure of the efficiency of a network scheme is the number of message delays incurred.

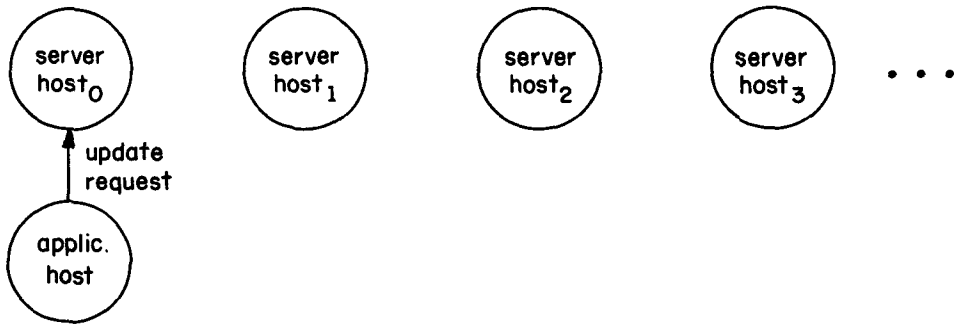
As we have indicated above, what we are interested in is a method by which we can provide resilient support for some distributed resource sharing activity. For purposes of illustration, let us assume we have some sort of data base (in the general sense) which is being read and modified by a group of network users. Let us consider, at least for purposes of description, that there is a set of server hosts which do nothing but perform the updates and mediate the synchronization of these 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 how the resiliency scheme works without failures.

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

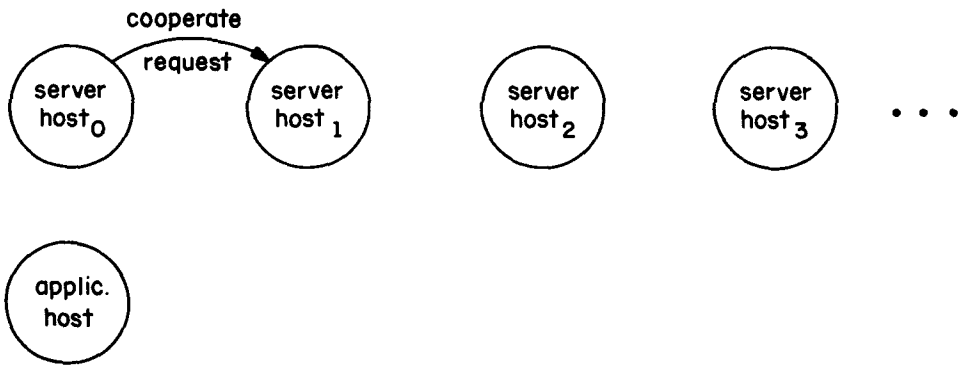
For the purposes of this discussion we will ignore to some extent the details of the end-to-end transmission. Some of the ACK's and timeouts mentioned below may be provided by an end-to-end protocol such as those described in Cerf and Kahn [1974] and Cerf et al. [1975]. In addition, the communication 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.

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.

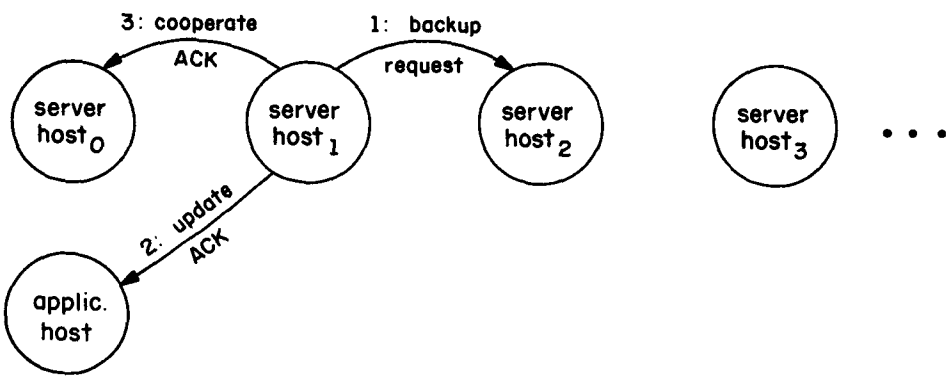
The second network message delay is incurred in figure 1b. The primary server host requests cooperation in executing the update operation from the first backup server host. The primary server host has already updated its data base. The first backup synchronization



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.



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

Update request sent to a primary server host

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 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. 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.

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 message 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 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 criterion has now been met. Third, a forward message acknowledgment is transmitted to the forwarding backup host. The message flow 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.

Alternative Backup Architectures. The backup servers have been arranged in a linear, ordered string. This is not essential. We have used the linear architecture in this paper for several reasons. It is easy to describe. It is one example of the single primary, multiple backup strategy for resilient resource sharing. It is also a minimum delay scheme for two-host resiliency. An example of a non-linearly ordered backup scheme is a broadcast scheme. In this scheme the primary broadcasts backup messages simultaneously to all backups. The broadcast scheme also has minimum delay. It requires fewer total messages than the linearly ordered scheme, but error recovery is more complex. Grapa is currently investigating the range of feasible backup architectures.

Summary. Resiliency is achieved in this scheme by a combination of techniques. The basic organization of the resiliency 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 passed down the chain, 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.

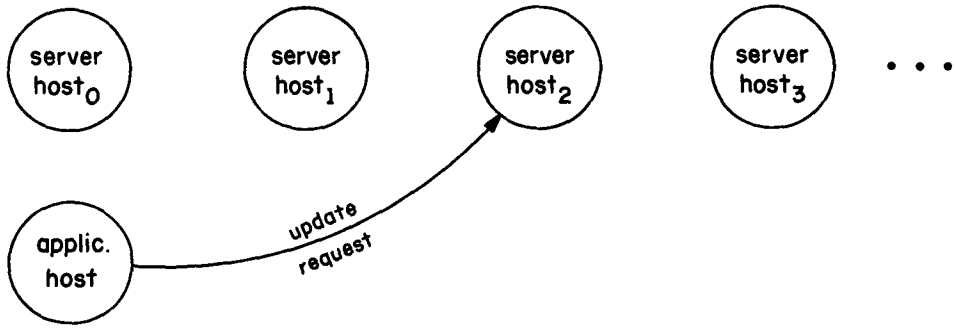
Also, for most applications one sequence number scheme can be applied to the messages to detect lost or duplicate messages. A second sequence number scheme can be applied to the requests themselves. This allows proper recovery in the event of failures. It also defines the order in which requests will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

There are two properties of this scheme that should be noted. First, regardless of where the user process sends the update request, he will get a response in three message delay times. (If the synchronizing scheme is moved into the application hosts, this delay can be cut to two message times.) Second, two nearly simultaneous host falures during a small critical interval are required to disrupt the scheme.

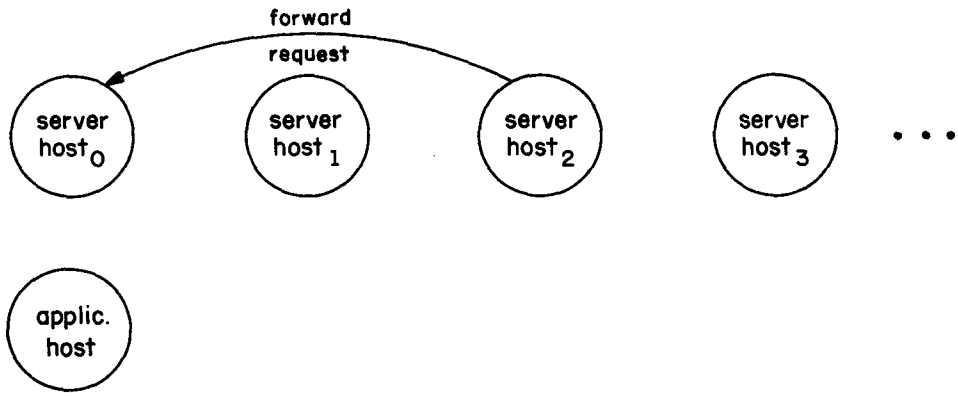
Failure Detection and Recovery

Failure Detection. The detection of failures may be accomplished in a variety of ways. Clearly, the time-outs associated with the ACK's will allow the system to detect a failure during the course of performing a request. If there are relatively long idle periods between requests, and if one wants to avoid the delays required to recover from a failure, it may be useful, for some applications, to have a low level "are you alive" protocol among the members of the chain. Otherwise, the error will not be detected until the next request is sent.

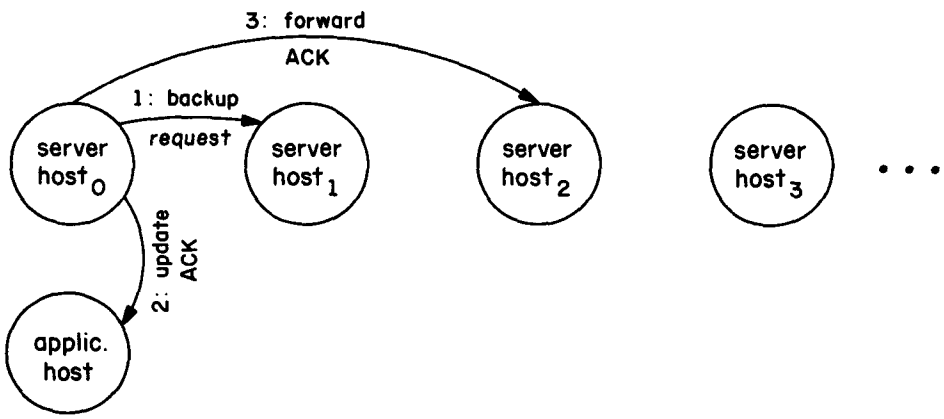
There are basically two kinds of failures which must be handled: 1) host failure and 2) network partition. Recovery from a host failure is relatively



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



2b: Backup host forwards update request to the primary 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

Update request sent to a backup server host

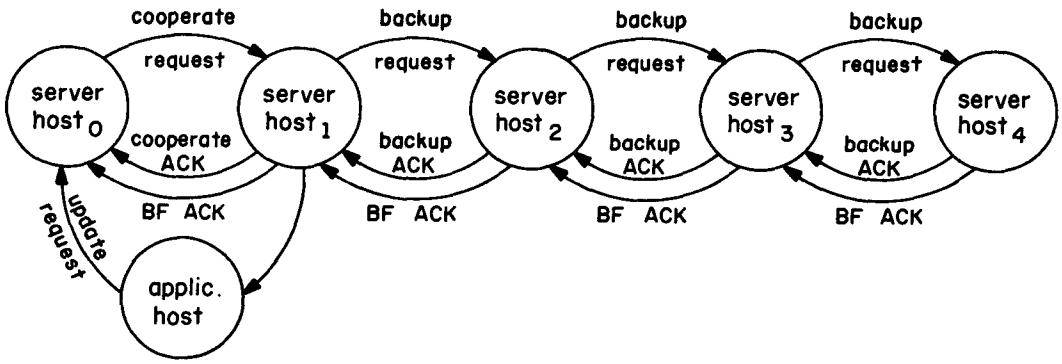
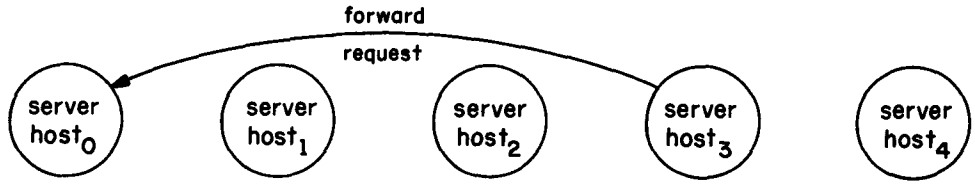
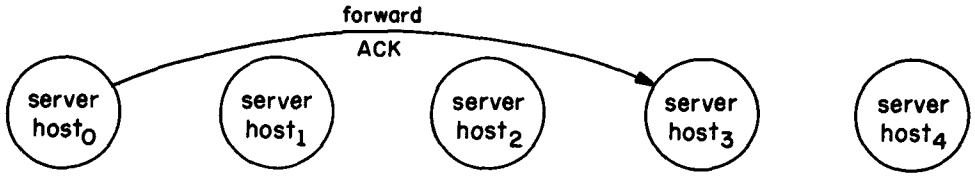


Figure 3

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

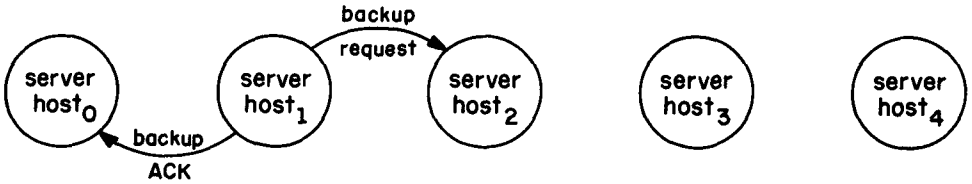


4a: An update request generated by the third host journalizes the request and sends a forward request message to the primary.



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

straightforward and will be discussed in the next section. However, operation during a network partition is much more difficult to handle and for a majority of applications 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 and have a way to rectify the existence of the two primaries when the partition is repaired. 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 by 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 the communications were restored and it was determined whether or not a partition had actually occurred.

Host Failure Recovery. Although much of the detail for failure recovery will depend heavily on the application and the degree of resiliency desired, it is possible to describe the basic mechanism by which host failures and network partitions are recovered. Let us consider the case of a host failure first (see figure 5). Assume that the subnet has notified a service host that messages to that host cannot be delivered because the host is dead. In figure 5 the dead host is a backup host. (If the primary dies a new primary must be elected. There are a variety of criteria that could be used. A simple algorithm would be to designate the first backup host as the new primary.)

In figure 5 we assume that the adjacent upstream host is notified of the failure. It will notify the primary of the failure so that the primary may delete the host from the backup table. The primary will then pass a "structure modification" message along the chain. (The "restructuring host", the one that started this recovery, will set a time-out waiting for the "structure modification" message to ripple down the chain.) After the "structure modification" has rippled back to the "restructuring host", it will attempt to establish communication with the next live downstream host and continue the propagation of the "structure modification message". The "restructuring host" will set a time-out and wait for the return ACK's to propagate. Once the "restructuring host" has received the restructuring ACK from the downstream host, it will then send any "backup requests" it has been holding on down the chain and normal operation has resumed.

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 data base.

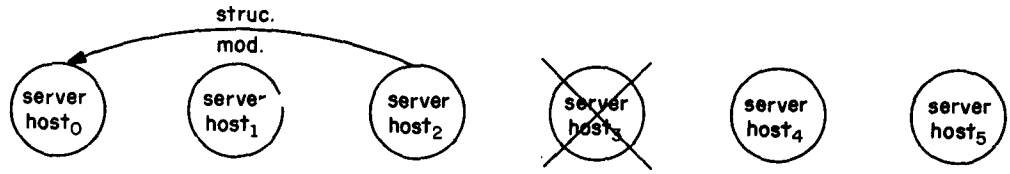
Note that there is no protection from a malicious backup server declaring itself to be the new primary. While malicious users can be addressed by this resiliency approach, servers must be benevolent. The approach to primary recovery discussed here is not resilient to single host error when, for example, the single host declares itself primary. Further work is required to make host failure recovery two-host resilient.

Apparent Network Partition Recovery. 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 whatever algorithm is fitting, 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 span the entire spectrum from doing nothing to the rather complex scheme described above.

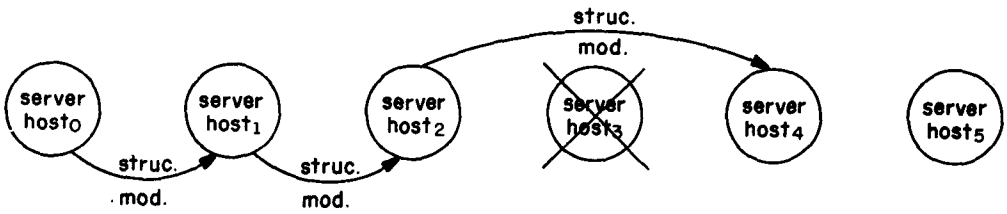
Alternative Resource Sharing Strategies

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.

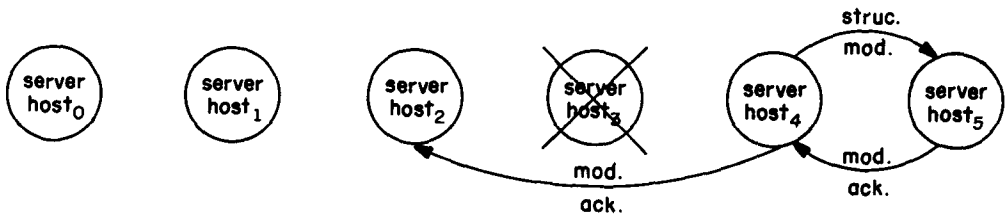
Let us consider the case where there is more than one primary. In the general case the primary which receives a service request must synchronize the execution of that service request with all other primaries. Otherwise, the system cannot guarantee that service requests are executed in the same order at all resource sites. (This requirement is essential in the general



5a: $Host_2$ sends the primary a structure modification message to notify it that $host_3$ has failed.



5b: The modification propagates back down to the instigating host who then establishes a connection with the next available host and notifies it of the change.



5c: The modification propagates to the end of the chain while acknowledgements are used guarantee that the messages arrived safely.

Figure 5

Restructuring after a Host Failure
($Host_3$ has failed and it has been detected by $host_2$.)

case. There may be specific applications 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 strategy 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 prevented from 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.

Summary

A single primary, multiple backup strategy has been proposed to support resilient distributed resource sharing. The flow of messages and service requests has been described for one possible architecture, a single primary with linearly ordered backup service hosts. Other architectures for structuring the backup service hosts are feasible and under investigation.

In the case of the linearly ordered backup architecture, the detection of host failure and recovery from that failure was discussed. The use of the resiliency technique to support a wide range of distributed networking services was also discussed. These included resource directories, network virtual file access, load sharing, and synchronization primitives.

The single primary, multiple backup strategy can support resilient resource sharing service requirements in the most general case. At the same time it achieves the minimum theoretical message delay. In the general case, this strategy is less resource consumptive, less complex, and no less resilient than a strategy that would employ more than one primary with or without backup service sites. In that sense a single primary strategy is always preferred over a multiple primary strategy for general applications.

References

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.