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 accept-

* 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.

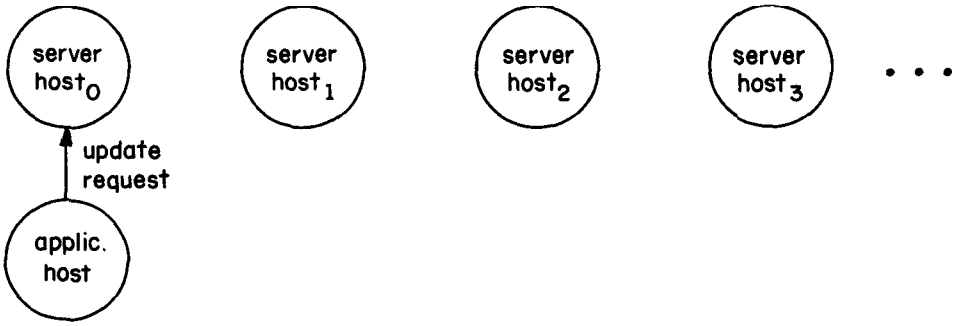
As we have indicated above, what we are interested in is a method by which we can provide resilient support for some distributed resource sharing activity. For purposes of illustration, let us assume we have some sort of data base (in the general sense) which is being read and modified by a group of 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.

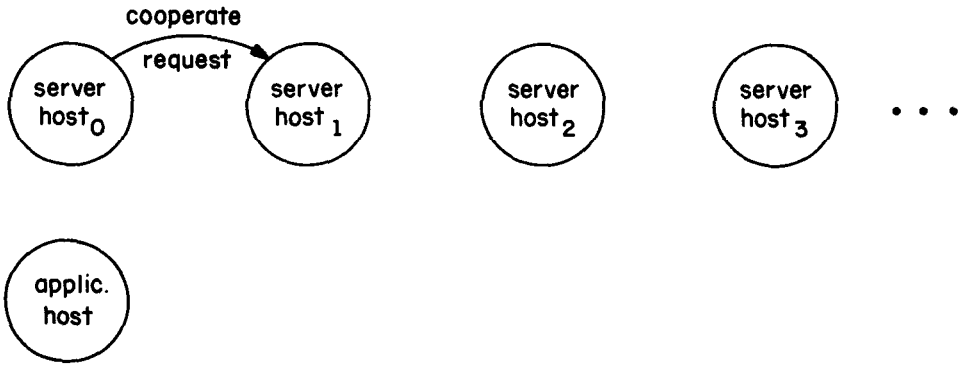
For the purposes of this discussion we will ignore to some extent the details of the end-to-end transmission. Some of the ACK's and timeouts mentioned below may be provided by an end-to-end protocol such as those described in Cerf and Kahn [1974] and Cerf et al. [1975]. In addition, the communication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

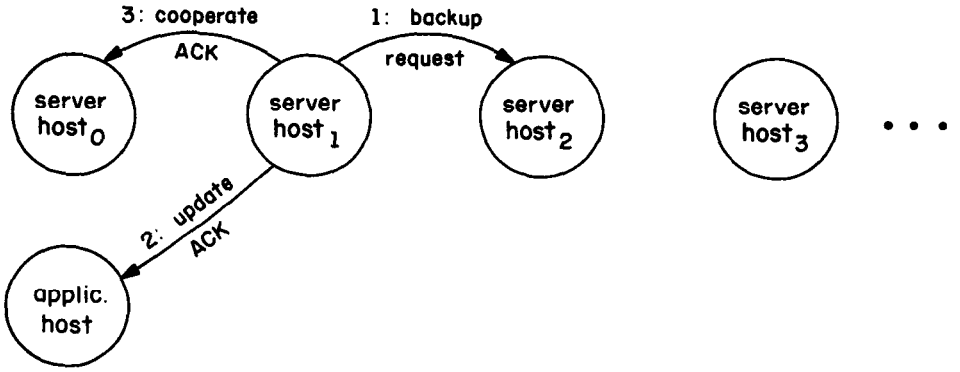
The second network message delay is incurred in figure 1b. The primary server host requests cooperation in executing the update operation from the first backup server host. The primary server host has already updated its data base. The first backup synchronization



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 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.

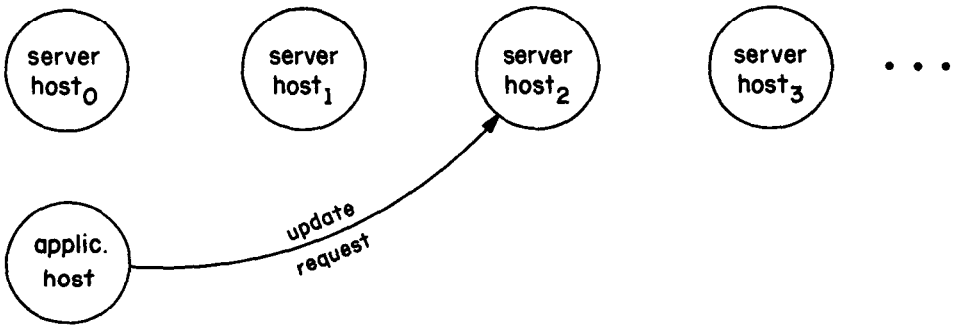
Also, for most applications one sequence number scheme can be applied to the messages to detect lost or duplicate messages. A second sequence number scheme can be applied to the requests themselves. This allows proper recovery in the event of failures. It also defines the order in which requests will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

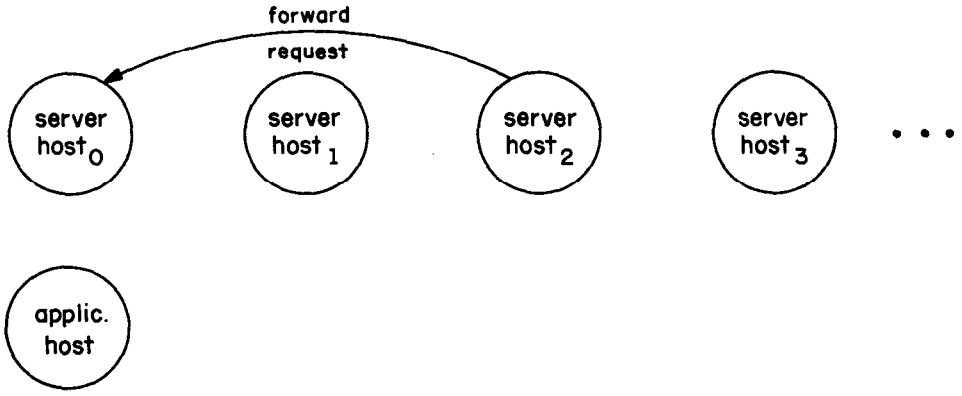
Failure Detection and Recovery

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.

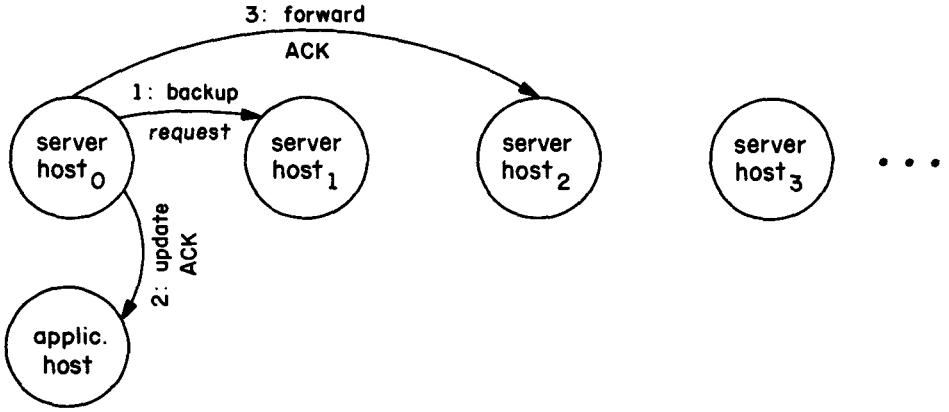
There are basically two kinds of failures which must be handled: 1) host failure and 2) network parti-



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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.