Technical Description of the DECsafe Available Server Environment

Lawrence S. Cohen
John H. Williams

The DECsafe Available Server Environment (ASE) was designed to satisfy the high-availability requirements of mission-critical applications running on the Digital UNIX operating system. By supplying failure detection and failover procedures for redundant hardware and software subsystems, ASE provides services that can tolerate a single point of failure. In addition, ASE supports standard SCSI hardware in shared storage configurations. ASE uses several mechanisms to maintain continuous operation and to prevent data corruption resulting from network partitions.

The advent of shared storage interconnect support such as the small computer system interface (SCSI) in the Digital UNIX operating system provided the opportunity to make existing disk-based services more available. Since high availability is an important feature to mission-critical applications such as database and file system servers, we started to explore high-availability solutions for the UNIX operating system environment. The outcome of this effort is the DECsafe Available Server Environment (ASE), an integrated organization of computer systems and external disks connected to one or more shared SCSI buses.

In the first section of this paper, we review the many product requirements that needed to be explored. We then define the ASE concepts. In the next section, we discuss the design of the ASE components. In subsequent sections, we describe some of the issues that needed to be overcome during the product's design and development: relocating client-server applications, event monitoring and notification, network partitioning, and management of available services. Further, we explain how ASE deals with problems concerning multihost SCSI; the cross-organizational logistical issues of developing specialized SCSI hardware and firmware features on high-volume, low-priced standard commodity hardware; and modifications to the Network File System (NFS) to be both highly available and backward compatible.[1]

Requirements of High-availability Software

The availability concept is simple. If two hosts can access the same data and one host fails, the other host should be able to access the data, thus making the applications that use the data more available. This notion of loosely connecting hosts on a shared storage interconnect is called high availability. High availability lies in the middle of the spectrum of availability solutions, somewhere between expensive fault-tolerant systems and a well-managed, relatively inexpensive, single computer system.[2]

By eliminating hardware single points of failure, the environment becomes more available. The goal of the

ASE project was to achieve a product that could be configured for no single point of failure with respect to the availability of services. Thus we designed ASE to detect and dynamically reconfigure around host, storage device, and network failures.

Many requirements influenced the ASE design. The most overriding requirement was to eliminate the possibility for data corruption. Existing single-system applications implicitly assumed that no other instance was running on another node that could also access the same data. If concurrent access did happen, the data would likely be corrupted. Therefore the preeminent challenge for ASE was to ensure that the application was run only once on only one node.

Another requirement of ASE was to use industry-standard storage and interconnects to perform its function. This essentially meant the use of SCSI storage components, and this did pose some challenges for the project. In a later section, we discuss the challenge of ensuring data integrity in a multihosted SCSI environment. Also, the limitation of eight SCSI devices per SCSI storage bus confined the scaling potential of ASE to relatively small environments of two to four nodes.

Less obvious requirements affected the design. ASE would be a layered product with minimal impact on the base operating system. This decision was made for maintainability reasons. This is not to say we did not make changes to the base operating system to support ASE; however, we made changes only when necessary.

ASE was required to support multiple service types (applications). Originally, it was proposed that ASE support only the Network File System (NFS), as does the HANFS product from International Business Machines Corporation.[3] Customers, however, required support for other, primarily database applications as well. As a result, the ASE design had to evolve to be more general with respect to application availability support.

ASE was also required to allow multiple service types to run concurrently on all nodes. Other high-availability products, e.g., Digital's DECsafe Failover and Hewlett-Packard's SwitchOver UX, are "hot-standby" solutions. They require customers to purchase additional systems that could be idle during normal operation. We felt it was important to allow all members of the ASE to run highly available applications as well as the traditional, hot-standby configuration.

The remaining requirement was time to market. IBM's HA/6000 and Sun Microsystems' SPARCcluster1 products were in the market, offering cluster-like high availability. We wanted to bring out ASE quickly and to follow with a true UNIX cluster product.

One last note for readers who might try to compare ASE with the VMScluster, a fully functional cluster product. ASE addresses the availability of single-threaded applications that require access to storage. For example, it does not address parallel applications that might need a distributed lock manager and concurrent access to data. Another effort was started to address the requirements of clusters in the UNIX environment.[4]

ASE Concepts

To understand the description of the ASE design, the reader needs to be familiar with certain availability concepts and terms. In this section, we define the ASE concepts.

Storage Availability Domain

A storage availability domain (SAD) is the collection of nodes that can access common or shared storage devices in an ASE. Figure 1 shows an example of a SAD. The SAD also includes the hardware that connects the nodes such as network devices and the storage interconnects. The network device can be any standard network interface that supports broadcast. This usually implies either Ethernet or a fiber distributed data interface (FDDI). Although the SAD may include many networks, only one is used for communicating the ASE protocols in the version 1.0 product. To remove this single point of failure, future versions of ASE will allow for communication over multiple networks. Other networks can be used by clients to access ASE services. The storage interconnect is either a single-ended or a fast, wide-differential SCSI. The shared devices are SCSI disks or SCSI storage products like HSZ40 controllers.

Symmetric versus Asymmetric SADs

There are many ways a SAD may be configured with respect to nodes and storage. In a symmetric configuration (see Figure 1), all nodes are connected

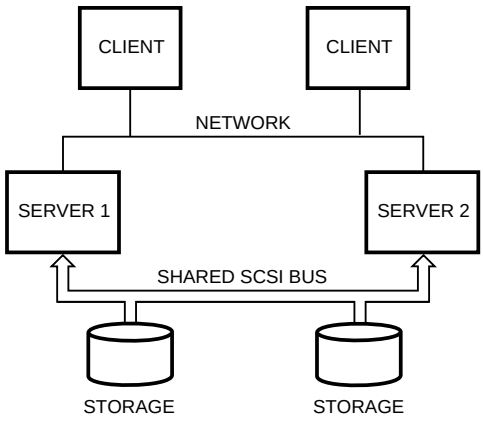


Figure 1
Simple Available Server Environment

to all storage. An asymmetric configuration exists when all nodes are not connected to all the storage devices. Figure 2 shows an asymmetric configuration.

The use of asymmetric configurations improves performance and increases scalability. Performance is better because fewer nodes share the same bus and have less opportunity to saturate a bus with I/O. Scalability is greater because an asymmetric configuration allows for more storage capacity. On the other hand, asymmetric configurations add significant implementation issues that are not present with symmetric configurations. Symmetric configurations allow for simplifying assumptions in device naming, detecting network partitions, and preventing data corruption. By assuming fully connected configurations, we were able to simplify the ASE design and increase the software's reliability. For these reasons, we chose to support only symmetric configurations in version 1.0 of ASE.

Service

We use the term *service* to describe the program (or programs) that is made highly available. The service model provides network access to shared storage through its own client-server protocols. Examples of ASE services are NFS and the ORACLE7 database. Usually, a set of programs or processing steps needs to be executed sequentially to start up or stop the service. If any of the steps cannot be executed successfully, the service either cannot be provided or cannot be stopped. Obviously, if the shared storage is not accessible, the service cannot begin. ASE provides a general infrastructure for specifying the processing steps and the storage dependencies of each service.

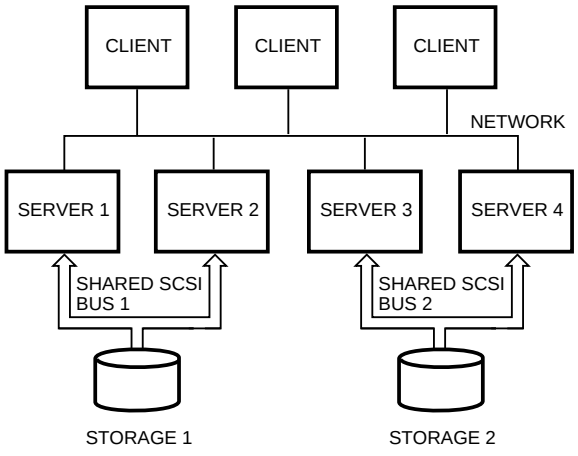


Figure 2
Asymmetric Configuration of ASE

Events and Failure Modes

ASE monitors its hardware and software to determine the status of the environment. A change in status is reported as an event notification to the ASE software. Examples of events include a host failure and recovery, a failed network or disk device, or a command from the ASE management utility.

Service Failover

The ASE software responds to events by relocating services from one node to another. A relocation due to a hardware failure is referred to as *service failover.* There are reasons other than failures to relocate a service. For example, a system manager may relocate a service for load-balancing reasons or may bring down a node to perform maintenance.

Service Relocation Policy

Whenever a service must be relocated, ASE uses configurable policies to determine which node is best suited to run the service. The policy is a function of the event and the installed system-management preferences for each service. For example, a service must be relocated if the node on which it is running goes down or if a SCSI cable is disconnected. The system manager may specify the node to which the service should be relocated. Preferences can also be provided for node recovery behavior. For example, the system manager can specify that a service always returns to a specified node if that node is up. For services that take a long time to start up, the system manager may specify that a service relocate only if its node should fail. Additional service policy choices are built into ASE.

Centralized versus Distributed Control

The ASE software is a collection of daemons (user-level independent processes run in the background) and kernel code that run on all nodes in a SAD. When we were designing the service relocation policy, we could have chosen a distributed design in which the software on each node participated in determining where a service was located. Instead, we chose a centralized design in which only one of the members was responsible for implementing the policy. We preferred a simple design since there was little benefit and much risk to developing a set of complex distributed algorithms.

Detectable Network Partition versus Undetectable Full Partition

A detectable network partition occurs when two or more nodes cannot communicate over their networks but can still access the shared storage. This condition could lead to data corruption if every node reported that all other nodes were down. Each node could try to acquire the service. The service could run

concurrently on multiple nodes and possibly corrupt the shared storage. ASE uses several mechanisms to prevent data corruption resulting from network partitions. First, it relies on the ability to communicate status over the SCSI bus. In this way, it can detect network partitions and prevent multiple instances of the service. When communication cannot occur over the SCSI bus, ASE relies on the disjoint electrical connectivity property of the SCSI bus. That is, if Server 1 and Server 2 cannot contact each other on the SCSI bus, it is impossible for both servers to access the same storage on that bus.

As a safeguard to this assumption, ASE also applies device reservations (hard locks) on the disks. The hard lock is an extreme failsafe mechanism that should rarely (if ever) be needed. As a result, ASE is able to adopt a nonquorum approach to network partition handling. In essence, if an application can access the storage it needs to run, it is allowed to run. Quorum approaches require a percentage (usually more than half) of the nodes to be available for proper operation. For two-node configurations, a tiebreaker would be required: if one node failed, the other could continue to operate. In the OpenVMS system, for example, a disk is used as a tiebreaker. We chose the nonquorum approach for ASE because it provides a higher degree of availability.

Although extremely unlikely to occur, there is one situation in which data could become corrupted: a full partition could occur during shadowed storage. Shadowing transparently replicates data on one or more disk storage devices. In a full partition, two nodes cannot communicate via a network, and the SCSI buses are disconnected in a way that the first node sees one set of disks and the second node sees another set. Figure 3 shows an undetectable full partition.

Even though this scenario does not allow for common access to disks, it is possible that storage that is replicated or shadowed across two disks and buses could be corrupted. Each node believes the other is down because there is no communication path. If one node has access to half of the shadowed disk set and the other node has access to the other half, the service may be run on both nodes. The shadowed set would become out of sync, causing data corruption when its halves were merged back together. Because the possibility of getting three faults of this nature is infinitesimal, we provide an optional policy for running a service when less than a full shadowed set is available.

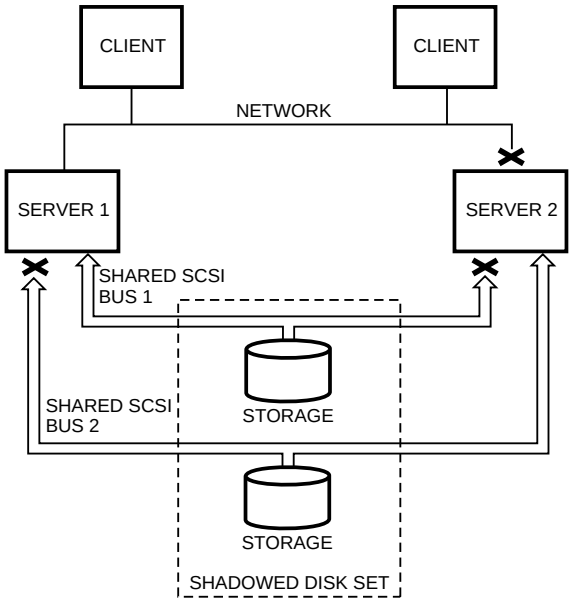


Figure 3
Full Partition

Service Management

ASE service management provides three functions: service setup, SAD monitoring, and service relocation. The management program assists in the creation of services by prompting for information such as the type of service, the disks and file systems that are required, and shadowing requirements. ASE gathers the requirements and creates the command sequences that will start the service. It thus integrates complex subsystems such as the local file systems, logical storage management (LSM), and NFS into a single service.

ASE version 1.0 supports three service types: user, disk, and NFS. A *user service* requires no disks and simply allows a user-supplied script to be executed whenever a node goes up or down. The *disk service* is a user service that also requires disk access, that is, disk and file system information. The disk service, for example, would be used for the creation of a highly available database. The *NFS service* is a specialized version of the disk service; it prompts for the additional information that is germane to NFS, for example, export information.

The monitoring feature provides the status of a service, indicating whether the service is running or not and where. It also provides the status of each node.

The service location feature allows system managers to move services manually by simply specifying the new location.

Software Mirroring

Software mirroring (shadowing) is a mechanism to replicate data across two or more disks. If one disk fails, the data is available on another disk. ASE relies on Digital's LSM product to provide this feature.

ASE Component Design

The ASE product components perform distinct operations that correspond to one of the following categories:

1. Configuring the availability environment and services
2. Monitoring the status of the availability environment
3. Controlling and synchronizing service relocation
4. Controlling and performing single-system ASE management operations
5. Logging events for the availability environment

The configuration of ASE is divided into the installation and ongoing configuration tasks. The ASE installation process ensures that all the members are running ASE-compliant kernels and the required daemons (independent processes) for monitoring the environment and performing single-system ASE operations. Figure 4 illustrates these components. The shared networks and distributed time services must also be configured on each member to guarantee connectivity and synchronized time. The most current ASE configuration information is determined from time stamps. Configuration information that uses time stamps does not change often or frequently and is protected by a distributed lock.

The ASE configuration begins by running the ASE administrative command (ASEMGR) to establish the membership list. All the participating hosts and daemons must be available and operational to complete this task successfully. ASE remains in the install state until the membership list has been successfully processed. As part of the ASE membership processing, an ASE configuration database (ASECDB) is created, and the ASE member with the highest Internet Protocol (IP) address on the primary network is designated to run the ASE director daemon (ASEDIRECTOR). The ASE director provides distributed control across the ASE members. Once an ASE director is running, the ASEMGR command is used to configure and control individual services on the ASE members. The ASE agent daemon (ASEAGENT) is responsible for performing all the single-system ASE operations required to manage the ASE and related services. This local system management is usually accomplished by executing scripts in a specific order to control the start, stop, add, delete, or check of a service or set of services.

The ASE director is responsible for controlling and synchronizing the ASE and the available services dependent on the ASE. All distributed decisions are made by the ASE director. It is necessary that only one ASE director be running and controlling an ASE to provide a centralized point of control across the ASE. The ASE director provides the distributed orchestration of service operations to effect the desired recovery or load-balancing scenarios. The ASE director controls the availability services by issuing sets of service actions to the ASE agents running on each member. The ASE director implements all failover strategy and control.

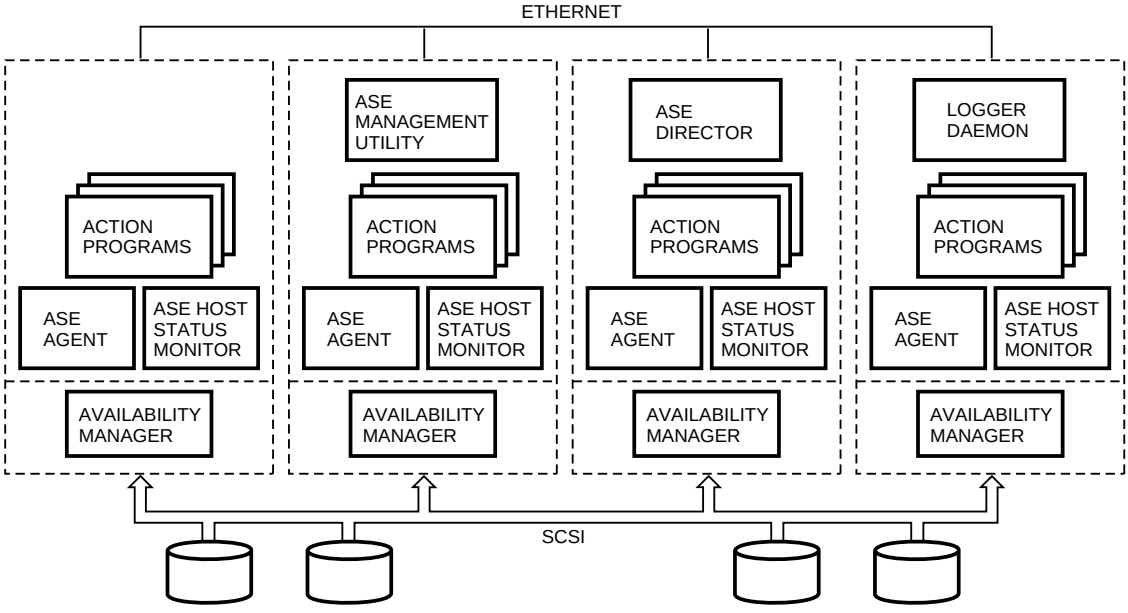


Figure 4
ASE Component Configuration

The ASE agent and the ASE director work as a team, reacting to component faults and performing failure recovery for services. The ASE events are generated by the ASE host status monitor and the availability manager (AM), a kernel subsystem. The ASE agents use the AM to detect device failures that pertain to ASE services. When a device failure is detected, the AM informs the ASE agent of the problem. The ASE agent then reports the problem to the ASE director if the failure results in service stoppage. For example, if the failed disk is part of an LSM mirrored set, the service is not affected by a single disk failure.

The ASE host status monitor sends host- or member-state change events to the ASE director. The ASE host status monitor uses both the networks and shared storage buses, SCSI buses, configured between the ASE members to determine the state of each member. This monitor uses the AM to provide periodic SCSI bus messaging through SCSI target-mode technology to hosts on the shared SCSI bus.

The ASE agent also uses the AM to provide device reservation control and device events. The ASE host status monitor repeatedly sends short messages, pings, to all other members and awaits a reply. If no reply is received within a prescribed time-out, the monitor moves to another interconnect until all paths have been exhausted without receiving a reply. If no reply on the shared network or any shared SCSI is received, the monitor presumes that the member is down and reports this to the ASE director. If any of the pings is successful and the member was previously down, the monitor reports that the member replying is up. If the only successful pings are SCSI-based, the ASE host status monitor reports that the members are experiencing a network partition. During a network partition, the ASE configuration and current service locations are frozen until the partition is resolved.

All ASE operations performed across the members use a common distributed logging facility. The logger daemon has the ability to generate multiple logs on each ASE member. The administrator uses the log to determine more detail about a particular service failover or configuration problem.

ASE Static and Dynamic States

As with most distributed applications, the ASE product must control and distribute state across a set of processes that can span several systems. This state takes two forms: static and dynamic. The static state is distributed in the ASE configuration database. This state is used to provide service availability configuration information and the ASE system membership list. Although most changes to the ASE configuration database are gathered through the ASE administrative command, all changes to the database are passed through a single point of control and distribution, the ASE director. The dynamic state includes changes in status of the base availability environment components and services. The state of a particular service, where and whether it is running, is also dynamic state that is held and controlled by the ASE director. Figure 5 depicts the flow of control through the ASE components.

ASE Director Creation

The ASE agents are responsible for controlling the placement and execution of the ASE director. Whenever an ASE member boots, it starts up the ASE agent to determine whether an ASE director needs to be started. This determination is based on whether an ASE director is already running on some member.

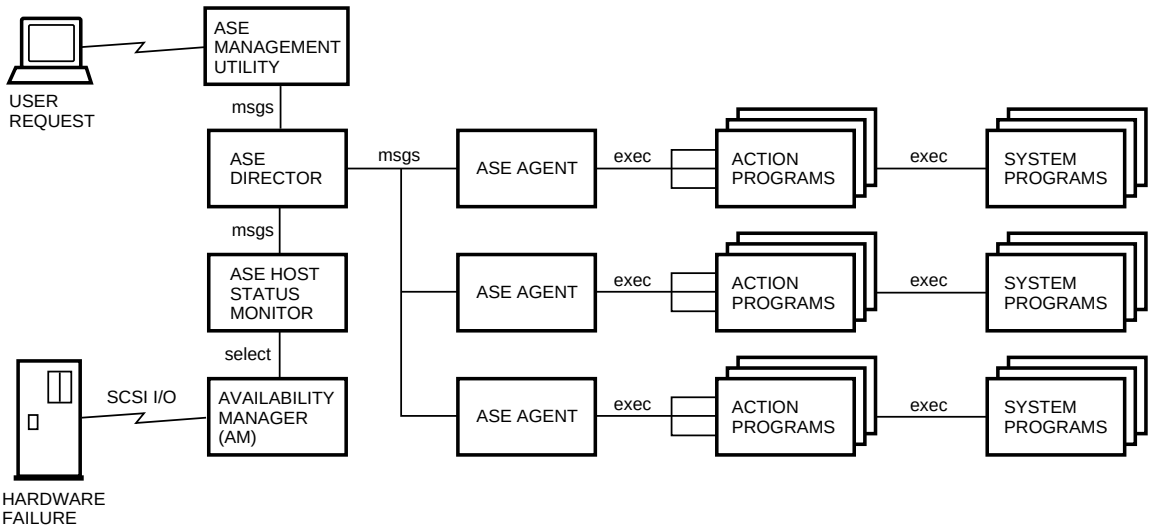


Figure 5
ASE Control Flow

If no ASE director is running and the ASE host status monitor is reporting that no other members are up, the ASE agent forks and executes the ASE director. Due to intermittent failures and the parallel initialization of members, an ASE configuration could find two ASE directors running on two different systems. As soon as the second director is discovered, the younger director is killed by the ASE agent on that system. The IP address of the primary network is used to determine which member should start a director when none is running.

ASE Director Design

The ASE director consists of four major components: the event manager, the strategist, the environment data manager, and the event controller. Figure 6 shows the relationship of the components of the ASE director.

The event manager component handles all incoming events and determines which subcomponent should service the event. The strategist component processes the event if it results in service relocation. The strategist creates an action plan to relocate the service. An action plan is a set of command lists designed to try all possible choices for processing the event. For example, if the event is to start a particular service, the generated plan orders the start attempts from the most desired member to the least desired member according to the service policy.

The environment data manager component is responsible for maintaining the current state of the ASE. The strategist will view the current state before creating an action plan. The event controller component oversees the execution of the action plan. Each of the command lists within the action plan is processed in parallel, whereas each command within a command list is processed serially. Functionally, this means that services can be started in parallel, and each service start-up can consist of a set of serially executed steps.

ASE Agent Design

The ASE agent is composed of the environment manager, the service manager, a second availability manager (AVMAN), and the configuration database manager. Figure 7 shows the ASE agent components.

All the ASE agent components use the message library as a common socket communications layer that allows the mixture of many outstanding requests and replies across several sockets. The environment manager component is responsible for the maintenance and initialization of the communications channels used by the ASE agent and the start-up of the ASE host status monitor and the ASE director. The environment manager is also responsible for handling all host-status events. For example, if the ASE host status monitor reports that the local node has lost connection to the network, the environment manager issues stop service actions on all services currently being served by the local node. This forced stop policy is based on the assumption that the services are being provided to clients on the network. A network that is down implies that no services are being provided; therefore, the service will be relocated to a member with healthy network connections.

If the ASE agent cannot make a connection to the ASE host status monitor during its initialization, the ASE host status monitor is started. The start-up of the ASE director is more complex because the ASE agent must ensure that only one ASE director is running in the ASE. This is accomplished by first obtaining the status of all the running ASE members. After the member status is commonly known across all ASE agents, the member with the highest IP address on the primary network is chosen to start up the ASE director. If two ASE directors are started, they must both make connections to all ASE agents in the ASE. In those rare cases when an ASE agent discovers two directors attempting to make connections, it will send

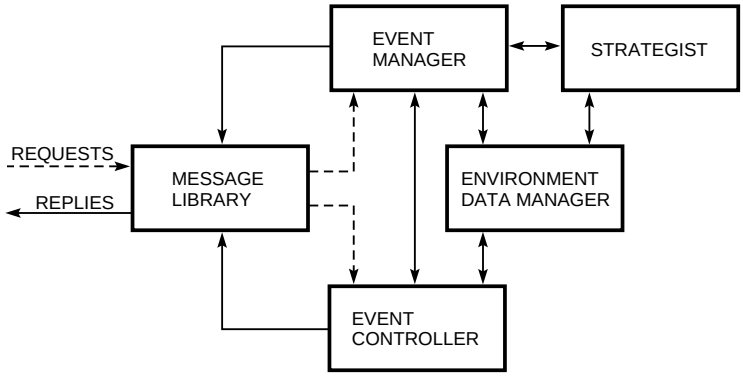


Figure 6
ASE Director

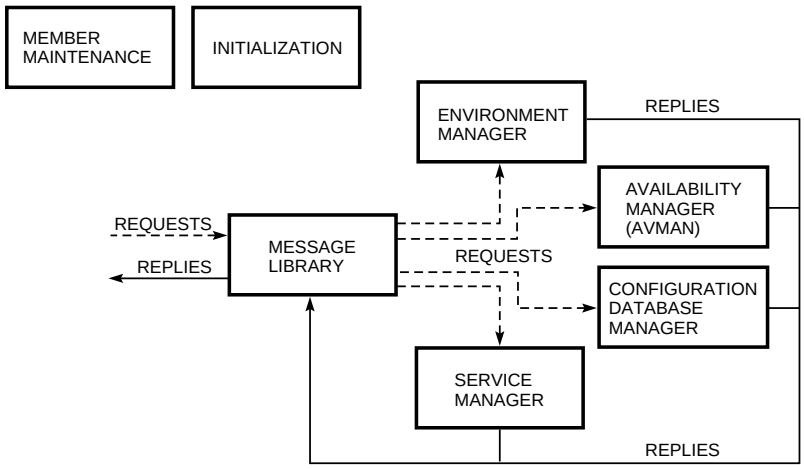


Figure 7
ASE Agent

an exit request to the younger director, the one with the newer start time.

The service manager component is responsible for performing operations on ASE services. The service manager performs operations that use specific service action programs or that determine and report status on services and their respective devices. The service actions are forked and executed as separate processes, children of the agent. This allows the ASE agent to continue handling other parallel actions or requests. The ASE agent is aware of only the general stop, start, add, delete, query, or check nature of the action. It is not aware of the specific application details required to implement these base availability functions. A more detailed description of the ASE service interfaces can be found in the section ASE Service Definition. When the service manager executes a stop or start service action that has device dependencies, the ASE agent provides the associated device reserves or unreserves to gain or release access to the device. Services and devices must be configured such that one device may be associated with only one service. A device may not belong to more than one service.

The agents' availability manager (AVMAN) component is called by the service manager to process a reserve or unreserve of a particular device for a service stop or start action. The AVMAN uses ioctl() calls to the AM to reserve the device, to invoke SCSI device pinging, and to register or unregister for the following AM events:

1. Device path failure—an I/O attempt failed on a reserved device due to a connectivity failure or bad device.
2. Device reservation failure—an I/O attempt failed on a reserve device because another node had reserved it.
3. Reservation reset—the SCSI reservation was lost on a particular device due to a bus reset.

A reservation reset occurs periodically as members reboot and join the ASE. The ASE agent reacts by rereserving the device and thereby continuing to provide the service. If the reservation reset persists, the ASE agent informs the ASE director. If a device path failure occurs, the ASE agent informs the ASE director of the device path failure so that another member can access the device and resume the service. The device reservation failure can occur only if another member has taken the reservation. This signifies to the ASE agent that an ASE director has decided to run this service on another member without first stopping it here.

The configuration database manager component handles requests that access the ASE configuration database. Working through the configuration database manager component, the ASE agent provides all access to the ASE configuration database for all other components of the ASE.

ASE Availability Manager Design

The availability manager (AM) is a kernel component of ASE that is responsible for providing SCSI device control and SCSI host pinging with target mode. The AM provides SCSI host pinging to the ASE host status monitor daemon through a set of ioctl() calls to the "/dev/am_host*" devices. As has been mentioned, the AM provides SCSI device control for pings and event notification to the ASE agent through ioctl() calls to the "/dev/ase" device. All ASE SCSI device controls for services and SCSI host pinging assume that all members are symmetrically configured with respect to SCSI storage bus addressing.

ASE Host Status Monitor Design

The ASE host status monitor (ASEHSM) component is responsible for sensing the status of members and interconnects used to communicate between members. As previously mentioned, this monitor is designed to provide periodic pinging of all network and SCSI interconnects that are symmetrically configured between ASE members. The ping rate is highest, 1 to 3 seconds per ping, on the first configured ASE network and SCSI bus. All other shared interconnects are pinged at a progressively slower rate to decrease the overhead while still providing some interconnectivity state. The ASE host status monitor provides member-state change events to both the ASE agent and the ASE director. The ASE agent initializes and updates the monitor when members are added or deleted from the ASE configuration database. The ASE host status monitor is designed to be flexible to new types of networks and storage buses as well as extensible to increased numbers of shared interconnects.

ASE Service Definition

ASE has provided an interface framework for available applications. This framework defines the availability configuration and failover processing stages to which an application must conform. The application interfaces consist of scripts that are used to start, stop, add, delete, query, check, and modify the particular service. Each script has the ability to order or stack a set of dependent scripts to suit a multilayered application. The NFS Service Failover section in this paper provides an example of a multilayered service that ASE supports "out of the box." ASE assumes that a service can be in one of the following states:

1. Nonexistent—not configured to run
2. Off-line—not to be run but configured to run
3. Unassigned—stopped and configured to run
4. Running—running on a member

At initialization, the ASE director presumes all configured services should be started except those in the off-line state. Whenever a new member joins the ASE, the add service action script is used to ensure that the new member has been configured to have the ability to run the service. The delete service script is used to remove the ability to run the service. The delete scripts are run whenever a service or member is deleted. The start service script is used to start the service on a particular member. The stop service is used to stop a service on a particular member. The check script is used to determine if a service is running on a particular member. The query script is used to determine if a particular device failure is sufficient to warrant failover.

ASE strives to keep a service in a known state. Consequently, if a start action script fails, ASE presumes that executing the stop action will return the service to an unassigned state. Likewise, if an add action fails, a delete action will return the service to a nonexistent state. If any action fails in the processing of an action list, the entire request has failed and is reported as such to the ASE director and in the log. For more details on ASE service action scripts, see the *Guide to the DECsafe Available Server*.[5]

NFS Service Failover

In this section, we present a walk-through of an NFS service failover. We presume that the reader is familiar with the workings of NFS.[1] The NFS service exports a file system that is remotely mounted by clients and locally mounted by the member that is providing the service. Other ASE members may also remotely mount the NFS file system to provide common access across all ASE members.

For this example, assume that we have set up an NFS service that is exporting a UNIX file system (UFS) named /foo_nfs. The UFS resides on an LSM disk group that is mirroring across two volumes that span four disks on two different SCSI buses. The NFS service is called foo_nfs and has been given its own IP address, 16.140.128.122. All remote clients who want to mount /foo_nfs will access the server using the service name foo_nfs and associated IP address 16.140.128.122. This network address information was distributed to the clients through the Berkeley Internet Name Domain (BIND) service or the network information service (NIS). If several NFS mount points are commonly used by all clients, they can be grouped into one service to reduce the number of IP addresses required. Although grouping directories exported from NFS into a single service reduces the management overhead, it also reduces flexibility for load balancing.

Further, assume that the NFS service foo_nfs has four clients. Two of the clients are the members of the ASE. The other two clients are non-Digital systems. For simplicity, the Sun and HP clients reside on the same network as the servers (but they need not). The ASE NFS service foo_nfs is currently running on the ASE member named MUNCH. The other ASE member is up and named ROLAIDS.

Enter our system administrator with his afternoon Big Gulp Soda. He places the Big Gulp Soda on top of MUNCH to free his hands for some serious console typing. Oh! We forgot one small aspect of the scenario. This ASE site is located in California. A small tremor later, and MUNCH gets a good taste of the Big Gulp Soda. Seconds later, MUNCH is very upset and fails. The ASE host status monitor on ROLAIDS stops receiving pings from MUNCH and declares MUNCH to be down. If the ASE director had been running on

MUNCH, then a new director is started on ROLAIDS to provide the much-needed relief. The ASE director now running on ROLAIDS determines that the foo_nfs service is not currently being served and issues a start plan for the service. The start action is passed to the local ASE agent since no other member is available. The ASE agent first reserves the disks associated with the foo_nfs service and runs the start action scripts. The start action scripts must begin by setting up LSM to address the mirrored disk group. The next action is to have UFS check and mount the /foo_nfs file system on the ASE hidden mount point /var/ase/mnt/foo_nfs. The hidden mount point helps to ensure that applications rarely access the mount point directly. This safeguard prevents an unmounting, which would stop the service. The next action scripts to be run are related to NFS. The NFS exports files must be adjusted to include the foo_nfs file system entry. This addition to the exports files is accomplished by adding and switching exports include files.

The action scripts then configure the service address (ifconfig alias command), which results in a broadcast of an Address Resolution Protocol (ARP) redirection packet to all listening clients to redirect their IP address mapping for 16.140.128.122 from MUNCH to ROLAIDS.[6] After all the ARP and router tables have been updated, the clients can resume communications with ROLAIDS for service foo_nfs. This entire process usually completes within ten seconds. The storage recovery process often contributes the longest duration. Figure 8 summarizes the time-sequenced events for an NFS service failover.

This scenario works because NFS is a stateless service. The server keeps no state on the clients, and the clients are willing to retry forever to regain access to their NFS service. Through proper mounting operations, all writes are done synchronously to disk such that a client will retry a write if it never receives a successful response.

If ASE is used to fail over a service that requires state, a mechanism has to be used to relocate the required state in order to start the service. The ASE product recommends that this state be written to file systems synchronously in periodic checkpoints. In this manner, the failover process could begin operation at the last successful checkpoint at the time the state disk area was mounted on the new system. If a more dynamic failover is required, the services must synchronize their state between members through some type of network transactions. This type of synchronization usually requires major changes to the design of the application.

Implementation and Development

We solved many interesting and logistically difficult issues during the development of the ASE product. Some of them have been discussed, such as the asymmetric versus symmetric SAD and distributed versus centralized policy. Others are mentioned in this section.

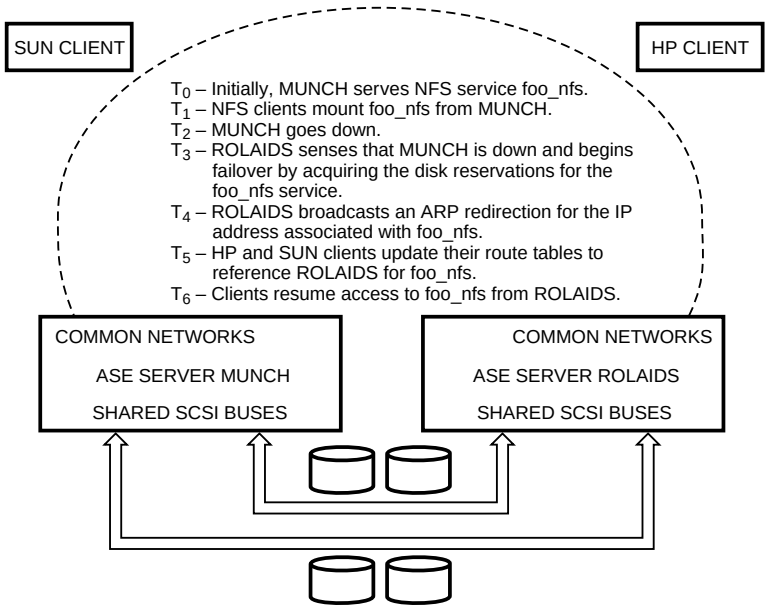


Figure 8
Time-sequenced Events for NFS Failover

The SCSI Standard and High-availability Requirements

The SCSI standard provides two levels of requirements: mandatory and optional. The ASE requirements fall into the optional domain and are not normally implemented in SCSI controllers. In particular, ASE requires that two or more initiators (host SCSI controllers) coexist on the same SCSI bus. This feature allows for common access to shared storage. Normally, there is only one host per SCSI, so very little testing is done to ensure the electrical integrity of the bus when more than one host resides. Furthermore, to make the hosts uniquely addressable, we needed to assign SCSI IDs and not hardwire them. Lastly, to support its host-sensing needs, ASE requires that SCSI controllers respond to commands from another controller. This SCSI feature is called target-mode operation.

In addition to meeting the basic functional SCSI requirements, we had to deal with testing and qualification issues. When new or revised components were used in ways for which they were not originally tested, they could break; and invariably when a controller was first inserted into an ASE environment, we found problems. Additional qualifications were required for the SCSI cables, disks, and optional SCSI equipment. ASE required very specific hardware (and revisions of that hardware); it would be difficult to support off-the-shelf components.

Note, however, when all was said and done, only one piece of hardware, the Y cable, was invented for ASE. The Y cable allows the SCSI termination to be placed on the bus and not in the system. As a result, a system can be removed without corrupting the bus.

The challenge for the project was to convince the hardware groups within Digital that it was worth the expense of all the above requirements and yet provide cost-competitive controllers. Fortunately, we did; but these issues are ongoing in the development of new controllers and disks. Our investigation continues on alternatives to the target mode design. We also need to develop ways to reduce the qualification time and expense, while improving the overall quality and availability of the hardware.

NFS Modifications to Support High Availability

The issues and design of NFS failover could consume this entire paper. We discuss only the prominent points in this section.

NFS Client Notification

The first challenge we faced was to determine how to tell NFS clients which host was serving their files both during the initial mount and after a service relocation. The ideal solution would have been to provide an IP address that all nodes in the SAD could respond to. If clients knew only one address, all NFS packets would be sent to that address and we would never have to tell the client the location had changed. The main problem with this solution is performance. Each node in the SAD would receive all NFS traffic destined for all nodes. The system overhead for deciding whether to drop or keep the packet is very high. Also the more nodes and NFS services, the more likely it would be to saturate individual nodes. Unfortunately, this solution had to be rejected.

The next best solution, in our minds, is per service IP addresses. Each NFS service is assigned an IP address (not the real host address). Now each node in the SAD could respond to its own address and to the addresses assigned to the NFS services that it is running. The main issues with this approach are the following: (1) It could use many IP addresses and (2) It is more difficult to manage because of its many addresses. However, there were no performance trade-offs, and we could move services to locations in a way that was transparent to the NFS clients. Notifying the clients after a relocation turned out to be easy because of a standard feature in the ARP that we could access through the ifconfig alias command of the Digital UNIX operating system.[6] Essentially, all clients have a cache of translations for IP addresses to physical addresses. The ARP feature, which we referred to as ARP redirection, allows us to invalidate a client-cached entry and replace it with a new one. The ifconfig command indirectly generates an ARP redirection message. As a result, the client NFS software believes it is sending to the same address, but the network layer is sending it to a different node.

Similar functionality could have been achieved by requiring multiple network controllers connected to a single network wire on the SAD nodes. This solution, however, requires more expense in hardware and is less flexible since there is only one address per board. Essentially, the latter means the granularity of NFS services would be much larger and could not be distributed among many SAD nodes without a great deal of hardware.

NFS Duplicate Request Cache

The NFS duplicate request cache improves the performance and correctness of an NFS server.[7] Although the duplicate request cache is not preserved across relocations, we did not view this as a significant problem because this cache is not preserved across reboots.

Other Modifications: Lock Daemons and mountd

We modified only two pieces of software related to NFS failover: the lock daemon and the mountd. We wanted the lock daemon to distinguish the locks associated with a specific service address so that only those

locks would be removed during a relocation. After the service is relocated, we rely on the existing lock reestablishment protocol. We modified the mountd to support NFS loopback mounting on the SAD, so that a file system could be accessed directly on the SAD (as opposed to a remote client) and yet be relocated transparently.

Future of ASE

Digital's ASE product was designed to address a small, symmetrically configured availability domain. The implementation of the ASE product was constrained by time, resources, and impact or change in the base system. Consequently, the ASE product lacks extensibility to larger asymmetric configurations and to more complex application availability requirements, e.g., support of concurrent distributed applications. The next-generation availability product must be designed to be extensible to varying hardware configurations and to be flexible to various application availability requirements.

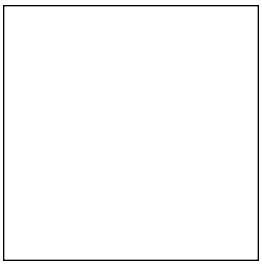
Acknowledgments

We thank the following people for contributing to this document through their consultation and artwork: Terry Linsey, Mark Longo, Sue Ries, Hai Huang, and Wayne Cardoza.

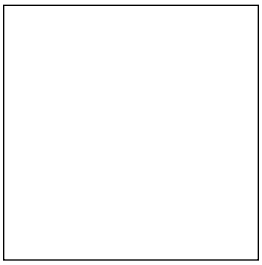
References

1. Sun Microsystems, Inc., *NFS Network File System Protocol Specification,* RFC 1094 (Network Information Center, SRI International, 1989).
2. J. Gray, "High Availability Computer Systems," *IEEE Computer* (September 1991).
3. A. Bhide, S. Morgan, and E. Elnozahy, "A Highly Available Network File Server," *Conference Proceedings from the Usenix Conference,* Dallas, Tex. (Winter 1991).
4. W. Cardoza, F. Glover, and W. Snaman, "Design of the Digital UNIX Cluster System," *Digital Technical Journal,* forthcoming 1996.
5. *Guide to the DECsafe Available Server* (Maynard, Mass.: Digital Equipment Corporation, 1995).
6. D. Plummer, *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48-bit Ethernet Address for Transmission on Ethernet Hardware,* RFC 0826 (Network Information Center, SRI International, 1982).
7. C. Juszczak, "Improving the Performance and Correctness of an NFS Server," *Conference Proceedings from the Usenix Conference,* San Diego, Calif. (Winter 1989).

Biographies



Lawrence S. Cohen
Larry Cohen led the Available Server Environment project. He is a principal software engineer in Digital's UNIX Engineering Group, where he is currently working on Digital's UNIX cluster products. Since joining Digital in 1983, he has written network and terminal device drivers and worked on the original ULTRIX port of BSD sockets and the TCP/IP implementation from BSD UNIX. Larry also participated in the implementation of Digital's I/O port architecture on ULTRIX and in the port of NFS version 2.0 to the DEC OSF/1 version of UNIX. Larry was previously employed at Bell Labs, where he worked on the UNIX to UNIX Copy Program (UUCP). He has a B.S. in math (1976) and an M.S. in computer science (1981), both from the University of Connecticut.



John H. Williams
John Williams is a principal software engineer in Digital's UNIX Engineering Group. John led the advanced development efforts for the UNIX cluster product and was the project leader for the DECsafe Available Server Environment version 1.1 and version 1.2. Before that, John designed and implemented the security interface architecture for the DEC OSF/1 operating system. Currently, John is responsible for the UNIX cluster management features. John received a B.S. in computer science from the Michigan Technological University in 1978.