The Design of an Advanced Storage Controller

James R. Bergsten
Founder, President and Chief Technical Officer
Ark Research Corporation

Introduction

This paper describes the design and implementation of an Advanced Storage Controller (*ASC*). This storage controller is used to enhance existing server and storage equipment. The ASC improves reliability with real-time remote data access and data mirroring, improves performance by caching, and enhances connectivity, management and security via Storage Area Networking (*SAN*) device zoning and routing.

This storage controller was implemented as a real-time application running under the *Ark Real Time Operating System* (*ARTS*) described in *The Design of a Reduced Function Operating System* by this author.

This document describes ASC features, capabilities, uses, and potential. It does not necessarily document specific command or configuration syntax. Please see the appropriate publications for precise product information and specifications for released products.

Table of Contents

Overview

The Advanced Storage Controller was developed to complement and supplement new and/or existing storage equipment, such as disk drives, redundant arrays of inexpensive disks (*RAID*), Compact Disk (*CD ROM's)*, and tapes. This was accomplished by designing and implementing architecture where:

1. Hosts (servers) attach to the ASC using existing physical connections, using SCSI or Fibre Channel protocols. These servers recognize the ASC as one or more industry-standard devices controlled and accessed using existing industry standard protocols. Consequently, there is no need to modify the server hardware or software.

2. The ASC connects to and controls physical device(s) and network paths. Each physical device appears to be attached to a standard server – requiring no device changes.

3. The ASC transfers data among the host(s), the ASC cache(s), the network, and the physical device(s).

4. The ASC enhances the operation with advanced functions, such as *n-way* data mirroring, and transparent access to remote devices. The ASC controls multiple network physical connections, using TCP/IP and other associated Internet protocols.

5. The ASC is controlled, configured, and monitored via additional network connections, implementing the File Transfer Protocol (*FTP*), Telnet, Hypertext Transmission Protocol (*HTTP*), and Simple Network Monitoring Protocol (*SNMP*).

Because the ASC interfaces at the SCSI (as opposed to file system) level, it is possible to attach the ASC to any host or device that supports SCSI protocol. An ASC can concurrently support many simultaneously attached clients or servers, such as those running AIX, HP/UX, Linux, Mac OS, NT, SCO, Solaris, Sun OS, and so on. The ASC enables data sharing amongst these environments (assuming that one has host software to accomplish this).

The ASC implements server-transparent *n-way* data mirroring. This allows multiple, real-time copies of user data for performance, reliability, disaster recovery, business continuation, and backup.

The ASC is implemented as a software application running under a Real Time Operating System (*RTOS*). As such, it does not depend on any particular hardware platform, and can be ported to different hardware architectures. The ASC application and RTOS support multitasking and either symmetric or asymmetric multiprocessing.

Both the ASC application and the RTOS were developed completely in-house. They are not based on any existing OS (such as UNIX – sorry). There is neither licensed nor public domain code in the ASC. All of the ASC code are written in *C* (not *C++*).

The RTOS and ASC software are currently running on a number of Intel Pentium-based multiprocessing platforms using one or more PCI busses for host, device, and network communication. As of this writing,

ASC is imbedded in a number of hardware platforms that are in production at a number of Ark customer sites around the world for about a year.

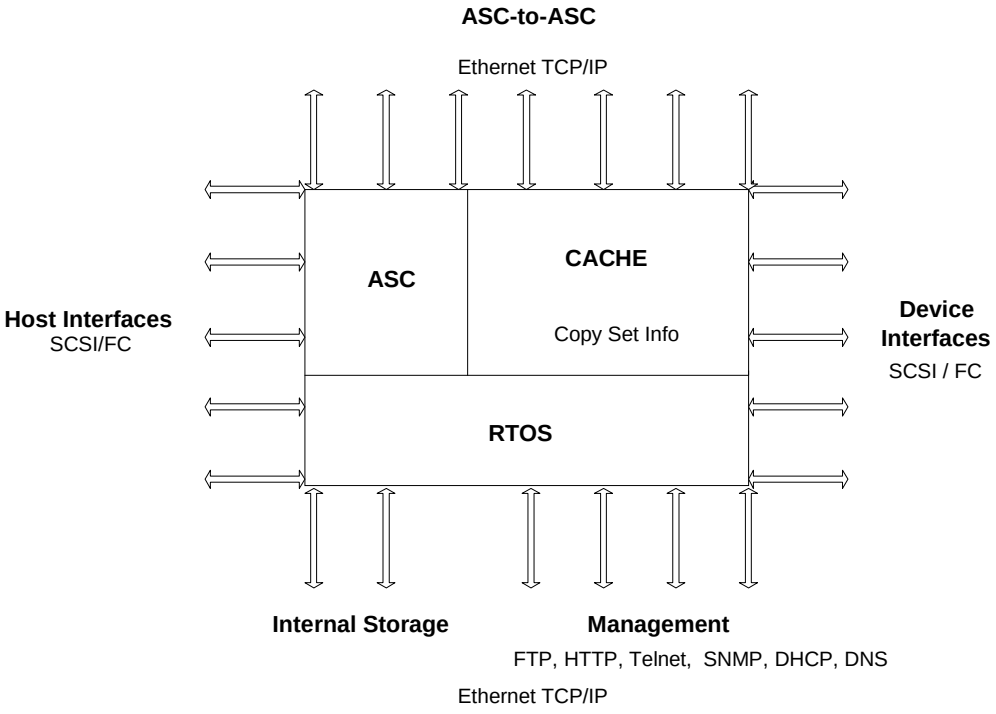

Figure 1: ASC Block Diagram

Definitions

This document uses conventional, generally accepted meanings of technical terms whenever this can be done in an unambiguous manner.

Host, *server*, *client*, and *computer* collectively refer to data processing systems physically attached to the *front end* of the ASC. From the server frame of reference, the ASC appears to be one or more SCSI *targets* and *logical units* – a storage controller with associated storage devices.

Device or *physical device* refers to physical storage media physically attached to the *back end* of the ASC. Unless otherwise specified, the *type* of device is immaterial. From the device perspective, the ASC appears to be one ore more SCSI *initiators*.

SCSI refers to the SCSI protocol, regardless of the physical transmission layer. Unless otherwise specified, SCSI refers to the SCSI-3 protocol.

Parallel SCSI specifically refers to the SCSI protocol layered on the parallel SCSI physical layer, whether narrow or wide, fast, ultra SCSI, or low voltage differential (*LVD*).

Fibre channel refers to the SCSI protocol layered on the Fibre Channel physical layer, typically Fibre Channel Arbitrated Loop (*FC-AL*).

A *host read* is a request by a host or server to transfer data **from** a device **to** host memory. Whenever the term *read* is used in this document the direction of data is always assumed to be **to** the host.

A *host write* is a request by a host or server to transfer data **from** host memory **to** a device. Whenever the term *write* is used in this document the direction of data is always assumed to be **from** the host.

Any *default values* specified in this document are subject to change, and can be overridden by configuration or operator commands.

ASC Objects

A *physical device* is a real device, external to, is attached to, and is controlled by an ASC.

The image of a *physical device* that a host connected to an ASC accesses is called a *host device* or *virtual device*. Multiple hosts may share the same host device, and may have different authority (for example, one host may have read/write access, while others have read/only access).

The ASC emulation of a physical device, maintained in the ASC cache, is called a *cache device*. Almost always, a host device maps directly onto a cache device.. A cache device normally maps to a physical device, but may also map to a portion of a physical device, or to a number of physical devices. ASC data mirroring mirrors cache devices.

A *local device* is a physical device attached to a *local* ASC.

A *remote device* is a cache device attached to a *remote* ASC. Usually, a remote device is mapped to a remote physical device. Remote devices are accessed via ASC-to-ASC communication paths. These paths can be IP network, parallel SCSI, or Fibre Channel connections.

Device Simulation

The ASC supports host I/O operations by emulating one or more physical devices. This section describes that emulation.

All ASC device simulation is based on the SCSI protocols, specifically SCSI-2 and SCSI-3. SCSI standards are described in ISO/IEC/ANSI publications.

Multiple Host Paths

ASC supports multiple, simultaneously active host paths to the same device. These paths may be attached to the same host (for performance and redundancy), or may be attached to different hosts. The ASC can be configured to prevent any host from accessing any ASC device, even if that host shares the SCSI bus or fibre channel loop with other hosts.

There are no architectural limits on either the number of hosts or the number of devices supported by an ASC; however, SCSI protocol limits the number of devices on any single SCSI bus.

SCSI Simulation

The ASC emulates all of the mandatory SCSI-3 commands and most of the optional ones as well. Notably, ASC supports Linked Commands and Relative Addressing. Auto contingent allegiance (ACA) is also supported.

For parallel SCSI, up to 256 simultaneous tagged queued requests per device are supported. Up to 65.535 simultaneous fibre channel requests per device are supported.

Simple, ordered, ACA, and head-of-queue tags are supported.

All abort and reset messages are supported.

The ASC maintains a separate sense buffer for each host-device nexus, minimizing contingent allegiance contention.

Normally all options of a supported SCSI command are supported by the ASC.

The following table lists the commands supported by ASC. Any deviations from the complete SCSI-3 specification are noted.

Device-type-Independent Commands	**Comments**
Inquiry	1. Standard Inquiry supported. 2. SCSI version returned is *SCSI-3* unless the device is configured to return *SCSI-2*. 3. *ARK* returned as the vendor-id, unless the device is configured to return the physical device's true vendor-id. 4. VPD pages supported: Serial Number Operating Definitions Ark version (vendor-specific page code 0xC0) 5. Command data supported. 6. Returns "no device" for undefined LUN's. 7. Returns "device not available" for offline devices, not-ready devices, or logical units that have not been configured.
Mode Select (6 and 10)	Non-volatile (saved) pages are not supported.
Mode Sense (6 and 10)	Non-volatile (saved) pages are not supported.
Read Buffer	Up to 4,096 bytes of buffer data are supported. Read buffer data are maintained on a host-device nexus basis. The contents of data read without having first being written are indeterminate. This command is used to test the path between the ASC and the host – there is no mechanism for using it to test ASC memory. Data are kept indefinitely (but are not kept across system restarts).
Receive Diagnostic Results	Passed through to the physical device if so configured, otherwise returns check condition -- unsupported command.
Release (6 and 10)	
Report LUNs	
Request Sense	Returns check condition -- LUN not ready for undefined LUN's.
Reserve (6 and 10)	
Send Diagnostic	Passed through to the physical device if so configured, otherwise returns check condition -- unsupported command.
Test Unit Ready	
Write Buffer	Up to 4,096 bytes of buffer data are supported. Read buffer data are maintained on a host-device nexus basis. The contents of data read without having first being written are indeterminate. This command is used to test the path between the ASC and the host – there is no mechanism for using it to test ASC memory. Data are kept indefinitely (but not kept across system restarts).

Disk-type Commands	**Comments**
Format Unit	Only passed on to the physical device if so configured, otherwise performs no operation and returns command complete status. Only the mandatory forms are supported. Returns check condition – unsupported command if issued to a copy set.
Lock/Unlock Cache	If the range exceeds the cache size, the end of the block range is

	locked/unlocked.
Prefetch Cache	If the range exceeds the cache size, the end of the block range is cached.
Read (6, 10, and 12)	
Read Capacity	As there are no fixed locations where a significant delay occurs, if the partial medium indicator is set, the block number returned is the first block number in the cache block following the passed block number.
Read Defect Data (10 and 12)	Passed through to the physical device. Returns check condition – unsupported command if issued to a copy set.
Rezero	Not passed through to the physical device, returns command complete.
Seek (6 and 10)	Seeks are never passed through to the physical device, if block number is valid, returns command complete.
Set Limits (10 and 12)	
Start/Stop Unit	Only passed on to the physical device if so configured.
Synchronize Cache	
Verify (10 and 12)	
Write (6, 10, and 12)	Writes are not allowed if: 1. The device has been configured as read-only. 2. The write includes block zero of the device, and the device has been configured to prevent block zero writes. 3. No physical device is accessible. 4. The number of pinned data blocks has been exceeded. 5. The device is an active secondary of a copy pair. 6. The device is reserved by a remote ASC system.
Write and Verify (10 and 12)	Write and Verify is not allowed if: 1. The device has been configured as read-only. 2. The write includes block zero of the device, and the device has been configured to prevent block zero writes. 3. No physical device is accessible. 4. The number of pinned data blocks has been exceeded. 5. The device is an active secondary of a copy pair. 6. The device is reserved by a remote ASC system.

CD/ROM Commands	**Comments**
Read (6, 10, and 12)	
Start/Stop Unit	Only passed on to the physical device if so configured.
Read Capacity	As there are no fixed locations where a significant delay occurs, if the partial medium indicator is set, the block number returned is the first block number in the cache block following the passed block number.
Seek (6 and 10)	Seeks are never passed through to the physical device, if block number is valid, returns command complete.

Customized versions of the ASC may include support for the pass-through or emulation of vendor-specific commands for control and configuration of vendor-specific physical devices. If they exist, these commands must be specifically configured to allow their use; otherwise they are rejected with check condition—unsupported command.

SCSI Device Control

This section describes ASC physical device control -- specifically for devices containing user data. Communication devices and internal device support is described elsewhere.

Tagged Queuing

The ASC supports up to 256 simultaneous parallel SCSI tagged queued commands, and up to 65,535 fibre channel commands per device.

The *SET MAXTAGS* command can be used to set (or limit) the number of simultaneously active commands issued to any specific device or back end controller. If MAXTAGS is set to one, tagged queuing is not used for this device. The default value is 32.

Generally, all requests are issued using simple tagged commands. High priority requests (such as those used in error recovery or device initialization) use the *head of queue* tag.

The number of simultaneous destages due to host asynchronous writes can be adjusted via the *SET MAXACTD* command. The default value is one.

As ASC maintains the order of host synchronous writes, and the order of asynchronous writes is not defined, physical device ordered tag queuing is never used.

Multiple Physical Paths

ASC supports multiple physical connections to a device or RAID controller. These paths are used for both performance and redundancy.

If multiple paths are operational, I/O requests are directed to the least busy path – that is, the path with the fewest outstanding active I/O requests. If equal, the first path defined will be used.

If a path fails, this condition is logged, the path is set offline, and remaining paths are used to satisfy I/O requests.

Paths are defined on a device-by-device basis. Thus it is possible to tune performance and balance path activity by appropriate configuration statements. For example, if devices *A*, *B*, *C*, and *D* have paths *1* and *2* and they are configured like this:

1. Device A primary path 1, secondary path 2
2. Device B primary path 2, secondary path 1
3. Device C path 1
4. Device D path 2

Then, device *A* will tend to favor path *1*, device *B* will tend to favor path *2*, device *C* will only use path *1*, and device *D* will only use path *2*.

Device Configuration

Physical devices can be individually configured to meet specific requirements and conditions, for example:

- A device can be configured to be *offline* (it is broken, not operational, or is not installed).
- Currently nonexistent devices can be configured. These will be periodically polled, so that if they are added later, they can be used without requiring a reconfiguration (nonexistent devices should be set offline to prevent unnecessary ASC polling).
- Configuration of specific parallel SCSI bus conditions to compensate for device problems, such as setting narrow versus wide data transfer, synchronous negotiation values, and so on.

- Configure a device to enable its write cache. Normally ASC disables device write caching in physically attached devices to insure that data written to the device are placed on the physical media before returning I/O operation ending status to ASC. If a RAID controller is designed such that data loss are not possible **even with write caching enabled**, then enabling the write cache could result in greatly improved host write performance.

To ease operation, reduce mistakes, and (mostly) eliminate confusion, each ASC object (such a each physical device, port, and so on), can be assigned a one to sixty-four character alphanumeric name. All commands and displays will accept and display a device name when one exists.

Native physical device identifiers are composed of a concatenation of the ASC bus identifier (for PCI busses, this is the bus, device, and device function), and the device identifier (for SCSI and Fibre Channel, this is the target identifier and the logical unit number (LUN).

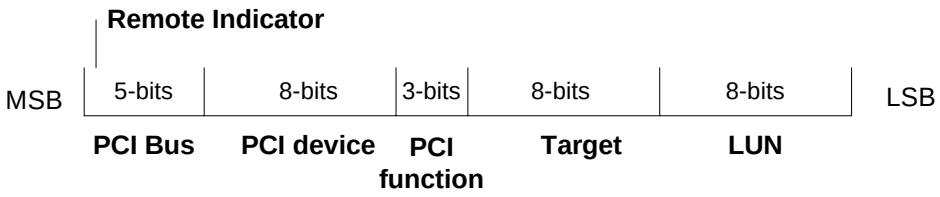

Figure 2: Device Identifier Syntax

Data Caching

The ASC is a *caching* storage controller. This means that user data moving between host(s) and physical device(s) are *staged* (read from physical devices) into cache memory. ASC maintains a *store-through* cache, that is, all data goes through (as opposed to around) the cache. Because ASC maps (virtualizes, abstracts) devices, it is not possible to have a physical path between the front and back ends of an ASC system.

This section provides an overview of ASC caching. Please see the *Data Path* section below for more specific information on the data flow of various host requests through the cache.

Cache Locking

Two types of *locks* are used to control access to cached user data:

1. The *Exclusive Lock* allows a single task read-write access to a cached data page. An exclusive lock is obtained for host write operations, and for data staging operations, as both of these place data **into** a cache page. Because only one task can hold an exclusive lock, write serialization is assured.

2. The *Shared Lock* allows multiple, simultaneous read/only access to a cached data page. This lock is obtained for host read requests (once the data has been staged), and for data destaging operations (after a host has modified a data page). Thus, it is possible for one or more hosts to read cached data while they are being destaged.

System primitives exist to request, wait for, and release locks. When an exclusive lock is requested for a nonexistent buffer, it is implicitly allocated and returned (devoid of user data). Both system (kernel) and task (application) forms exist, allowing ASC to quickly respond to host initial data requests.

Two other kinds of locks exist:

1. *Multiprocessing locks* – these are used to serialize resources in a multi-processor system. These locks are not directly related to cache locks (but are used to obtain and release cache locks). These locks are maintained by the RTOS.

2. *Host cache locks* – these are used to prevent cache pages from being stolen by other host requests. These locks are counters, obtained by host request (the SCSI Lock Cache command), or by operator request.

In this document section, references to locks imply shared and exclusive locks only.

Host Read Data Caching

Because systems and applications tend to read some data much more frequently than others, *data caching* greatly improve performance (as the data does not have to be reread from slow media, such as spinning disks). Any time data can be presented from an existing cache copy, no physical device activity occurs, and we refer to this as a *cache hit* or *hit*. Typically, between 80 and 95 percent of I/O read requests are satisfied from cache.

When a host requests data that are not already resident in cache, they are read from physical devices into ASC memory, then passed on to the host. We refer to this as a *cache miss* or *miss*.

To minimize host delay time, data may be transferred in a series of *packets*. While cached data are transferred to the host, remaining data are simultaneously being read from physical devices. This is called a *partial hit*. The following figure illustrates this:

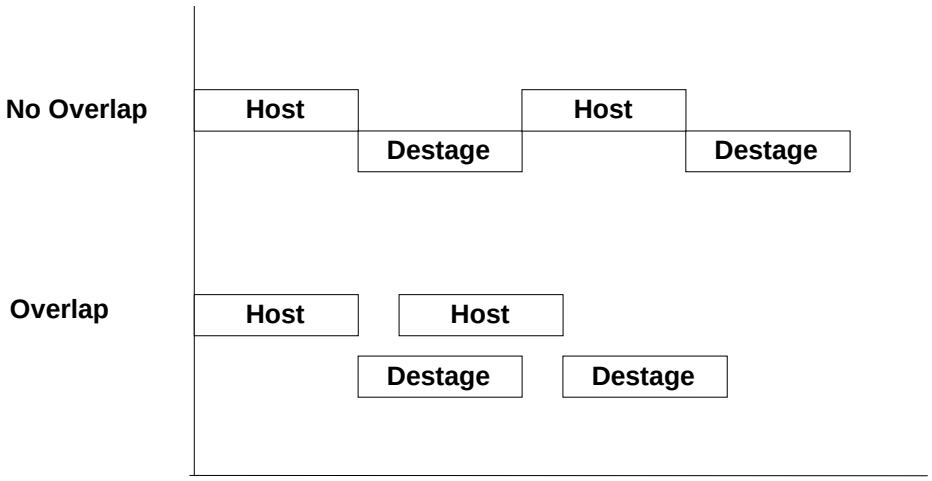

Figure 3: Host Data Read Timeline

The ASC maintains user data in 4,096-byte (*4K*) *cache blocks* or *pages*. Each page contains an equal number of fixed-length blocks, aligned on a 4K boundary. For example, most disks are formatted into 512 byte blocks. The ASC caches eight contiguous 512-byte blocks into a single cache block. These eight blocks are aligned to an eight-block boundary so that the first cache block might contain disk blocks zero through seven, the next eight through fifteen, and so on.

Each cache block descriptor contains a mask of present, modified, pinned, and destages-in-process for each user block contained in the cache block. When a cache block is allocated, only the data needed by the host are staged. The masks are 32 bits wide, allowing for device block sizes ranging from 128 through 4,096 bytes (in even multiples of two).

Each cache block is located via a hashing algorithm, using the cache block number and cache device identifier to locate the hash chain containing a pointer to the appropriate cache block. There are presently 4,096 linked lists of cache blocks (the more lists, the shorter the list search – for a 512MB cache, a typical maximum list size would be 32 elements).

The ASC uses a *least-recently-used* (*LRU*) algorithm to determine which user data to retain in cache. Whenever a block is accessed via a host read request, the block is moved to the beginning of the LRU queue and the beginning of its hash list.

The ASC also contains a *least-frequently-used (LFU)* algorithm, however, this is not presently enabled.

When a new cache block is required and there is no free storage available to assign to a block, the block at the bottom of the LRU list is released (*stolen*).

It is possible to manually *lock* data into cache to tune overall system performance, or to give certain applications an artificial performance boost. The *PREFETCH* command allows an administrator to load and lock pages into the ASC cache.

Data Read Prefetch

Many host applications read data *sequentially* -- that is, from contiguously ascending disk blocks. It is possible to improve performance by anticipating sequential host requests and *prefetching* data from physical devices before they are requested. This works because hosts typically process data between requests, giving ASC the time to bring the data into cache.

The ASC keeps track of each host's read requests on a device-by-device basis. If a number of consecutive sequential read requests are detected (that is, the following request is contiguous with the previous request), then prefetch may occur.

Generally, the more sequential activity a host does, the more the host will do. Therefore, ASC allows three settings for prefetch:

```
SET PREFETCH ON|OFF p1 b1 p2 b2 p3 b3
```

If prefetch is on, and the host does *p1* consecutive sequential reads, ASC will prefetch *b1* blocks.

When the host does *p2* consecutive sequential reads, ASC will prefetch *b1multiplied by b2* blocks.

When the host does p3 consecutive sequential reads, ASC will prefetch *b1multiplied by b2 multiplied by b3* blocks.

When the first non-sequential read is detected, the prefetch counter is reset to zero.

The current default values are:

```
SET PREFETCH 4 16 8 16 32 4
```

The ASC supports host controlled prefetch via settings in the SCSI Caching mode page.

ASC maintains statistics showing the number of blocks prefetched, the number of blocks prefetched but not accessed by a host, and the number of blocks that were already cached when prefetch was attempted. If the second value is large, ASC is doing unnecessary prefetching and the multipliers should be lowered. If the third value is large, the host is probably requesting data faster than ASC can prefetch it.

Host read requests take precedence over prefetch. It the host sequential read pointer is past the prefetch pointer, the prefetch is cancelled.

Host Write Data Caching

As the ASC implements a store-through cache, all host writes transfer data to the ASC cache, then the ASC transfers the data to physical devices.

ASC may break a host request into smaller pieces to perform simultaneous host data transfer and data writing (*destage*) to physical devices. The maximum size of these packets may be configured to tune write performance by use of the *SET WRITEBLK* command. There is no default value, however, the largest single-packet data host (or real device) data transfer is one megabyte. Larger requests are broken up into one-megabyte increments.

ASC destages take precedence over all other back end activity, including host data staging and prefetch.

Data Write Combining and Multitasking

The ASC may merge multiple host asynchronous writes into one physical I/O operation when the data are written to physical devices. The ASC may also combine multiple host writes of the same data into a single write operation. These optimizations improve overall performance by optimizing physical bus activity, lowering the number of individual I/O operations and the number of I/O interrupts serviced.

Host asynchronous writes that are **not** contiguous can be started to physical devices simultaneously using tagged queuing. The number can be adjusted with the *SET MAXACTD* command.

ASC **always** preserves the order of host synchronous writes by issuing them in the order received, one at a time.

Memory Disks

A device may be simulated entirely in cache memory with no physical device counterpart. All memory disk operations are *hits* -- resulting in excellent performance. Memory disks are sometimes called *solid state disks*, or *SSD's*.

A memory disk typically has much fewer blocks than a physical disk. It is often used for very frequently accessed data, such as paging space, temporary files, or database indexes.

Any number of memory disks can be created, using up to 75% of the ASC total cache size. A memory disk is defined with a defined number of user data blocks of a given size (defaulting to 512 bytes). Any number of hosts may share a memory disk, making it an ideal candidate for cross-host data sharing, resource locking or communication.

Based on how a memory disk is configured, permanent (versus temporary) memory disk data can be saved on internal hard drives on system shutdown, power failure, or restart.

Device Mapping

ASC device mapping statements define host-to-device mapping. There are three types of mappings:

- *Host to Device Mapping* – where a host image of a device is defined.
- *Unassigned Device Mapping* – where a physical device that is not accessible by any host is defined.
- *Spare Device Mapping* – where a physical device that is to be used as a *spare* is defined.

Any mapping can be made part of a static system configuration or dynamically defined in real-time. There is no architectural limit to the number of defined devices.

Host to Device Mapping

These mappings are used to associate the host and cache image of a device (sometimes referred to as a *host device* or *virtual device*) with physical storage media.

Generally a mapping takes the following form:

```
MAP host-interface <to> device-interface <options>
```

The host-interface contains:

`host-port`	identifies the ASC physical port the host is attached to.
`host-id`	identifies the host, for example, a SCSI initiator ID.
`target-id`	the target identifier of the emulated host device.
`target-lun`	the logical unit number of the emulated host device.

The *host-port* parameter can be specified as an asterisk (*) implying **all** host connections. The *host-id* parameter is often specified as an asterisk implying **all** hosts on this bus. The word *to* is optional.

For locally attached physical devices, the device-interface contains:

`device-port`	identifies the ASC physical port the device is attached to.
`device-id`	identifies the device, for example, a SCSI initiator ID.
`device-lun`	the logical unit number of the device.

For physical devices attached to remote ASC's, the device interface contains:

`SYSTEM IP-address | network-name`	identifies the remote ASC the device is attached to.
`device-port`	identifies the remote ASC physical port the device is attached to.
`device-id`	identifies the device, for example, a SCSI initiator ID.
`device-lun`	the logical unit number of the device.

Hosts may be identified by additional means, such as a fibre channel fabric address. Similarly, physical volume serial number, fibre channel name, address, or worldwide identifier may identify devices.

The host-port and the device-identifier may be specified by user-created symbolic device and port names.

It is architecturally possible to map a host device on a portion of a larger physical device, or onto a set of smaller devices (including the ability to *stripe* data across a number of devices). ASC mappings have a maximum resolution of a cache data block, that is 4,096 bytes or (typically) eight 512-byte data blocks.

Any number of hosts may share a device – this is accomplished by mapping different host connections to the same host and cache device.

Creating Memory Disks

The device-interface portion of a memory devices map statement contains:

`Memorydisk`	defines this device-interface as a memory disk.
`memory-device-id`	an arbitrary value that uniquely identifies the device.
`memory-device-lun`	an arbitrary value that uniquely identifies the device.

`block-count`	the number of device blocks emulated.
`block-size`	the device block-size (128-4096 bytes in powers of 2).
`Permanent`	optionally specifies that this memory disk is saved to internal storage.

Mapping Options

Options are added to configuration map statements to control host access and permissions, for example:

NF	This host **cannot** issue the device format command
NS	This device **cannot** be used as a copy set secondary device
N0	This host **cannot** write to device block zero
PD	Pass diagnostic commands through to the device
PI	Pass inquiry data from the device to the host
PS	Pass start/stop commands through to the device
Readonly	This host can read the device, but cannot write to it
S2	Emulate a SCSI-2 device
S3	Emulate a SCSI-3 device (default)
Sy	**All** host writes are treated as synchronous writes

Configuration-less Mapping

If an ASC is booted without a user configuration, or if a configuration contains no mapping, the ASC will implicitly map devices.

Each physical device found when the ASC boots is made available to all host connections at the same identifier and logical unit (or as close as possible given the hardware configuration). In this mode, the ASC appears totally transparent to the host (as a *lump in the cable*).

If an ASC finds **no** back-end physical devices, a single memory disk is created and is accessible from each host port. This is used for ASC manufacturing testing to ensure that each port is operational (by running I/O tests after looping each back-end to each front-end).

Unassigned Device Mapping

An *unassigned* device is one that exists, but has no local host connections. Such a device might be used as a copy set secondary, or as a remote device accessed by another ASC (access rules for remotely reserved devices are described in *Remote Data Access* below).

The syntax of an unassigned device is:

```
MAP None device-interface <options>
```

The *device-interface* and *options* are the same as for a host-mapped device.

Host paths may be dynamically added to an unassigned device, thereby assigning it. Conversely, host paths may be removed from an assigned device, leaving it unassigned.

Devices that are found when the ASC boots (via bus scans) but are not explicitly mapped, are considered unassigned (unless there is **no** explicit configuration at all, in which case they are mapped to host ports in a lump in the cable mode).

Internal Devices

Physical devices may be reserved for internal ASC use only. These devices may be used to save ASC logs, configurations, system and application images, system status, point-in-time backup data, and so on.

One or two internal IDE drives are implicitly reserved on some ASC platforms for this purpose.

Internal devices **can not** be reassigned for user use

On some platforms, additional (or different) devices may be reserved for ASC use by configuring:

```
MAP Internal device-interface <options>
```

Spare Device Mapping

A *spare* device is used as needed to replace a failing device. A set of these devices is called a *spare pool*. Spares are selected based on their size versus the size of the failing device.

The syntax of an unassigned device is:

```
MAP Spare device-interface <options>
```

The *device-interface* and *options* are the same as for a host-mapped device.

Generally, spares are temporarily allocated when needed and released when no longer needed. Like unassigned devices, spares can be reassigned as host storage when required.

Data Path

This section describes the order of operations for various host data requests. Please note that:

1. These descriptions assume that no errors, tagged queue delays, or throttling are taking place. For example, checks for pending reserves, tagged queuing state, device ready state, pending contingent allegiance or check conditions, and so on are neither described in the descriptions nor shown in the figures (these checks adhere to the SCSI-3 specifications).

2. Host abort requests, errors, or bus resets (obviously) can truncate or terminate any of these processes (reset/abort processing also adheres to the SCSI-3 specifications).

3. Based on the configuration, the operational state of devices, and the I/O load, data may be staged and/or destaged to a number of devices. For example, an established copy set may spread reads among operational devices. Likewise, host-modified data may be written to one, a number, or all of a copy set based on its state (see *Data Replication/Mirroring* below for more details).

4. Host activity may result in the creation of additional work, for example, anticipatory data prefetch.

Synchronous versus Asynchronous Host Writes

From a host perspective, *synchronous writes* have more integrity than *asynchronous writes*, as synchronous writes do not post completion status to the host until the data are written to physical device(s).

Synchronous writes to asynchronous remote devices are slightly more risky as they post completion to the host when the data are successfully written to the remote ASC's **cache**, before the remote ASC writes the data to physical device(s).

Asynchronous writes are the **most** risky as they post completion to the host when the data are successfully transferred to the local ASC **cache, before the ASC writes the data to physical devices, and before transferring the data to any remote ASC's.**. However, from a host perspective, asynchronous writes are usually much faster than synchronous writes.

Typically, host applications and systems issue synchronous writes for serializing events, such as filesystem directory and file allocation record updates, or database log updates.

Hosts control the type of write by:

1. Issuing Write commands **without** the *Force Unit Access* (*FUA*) bit set implying asynchronous writes.
2. Issuing Write commands **with** the Force Unit Access (FUA) bit set to force synchronous writes.
3. Issuing the Synchronize Cache command to begin, and optionally wait for the destage of all or some of pending modified data.
4. Enabling or disabling write caching by modifying the caching mode page for each device.

The ASC can be configured to

1. Treat **all** writes as synchronous by use of the *SET SYNWRITE* command.
2. Treat **all** writes to specific devices by specific hosts as synchronous by system configuration statements.
3. Force synchronous mode in the event of a power failure, uninterruptable power supply (UPS) battery low condition, or cache saturation condition.
4. Limit the number of concurrently active asynchronous writes by use of the *SET MAXACTW* command.

The ASC defaults to synchronous writes, with a limit of 32 currently active host asynchronous writes (if enabled by the user).

Delayed Writes

The ASC can be configured to insert an artificial delay between the time that data are modified and the time that the data are destaged to physical devices. This is sometimes referred to as a *delayed* or *lazy write* and is enabled using the *SET LAZYWDEL* command. The ASC defaults to no delayed writes.

Lazy writes improve the chances of eliminating duplicate destages owing to multiple host writes of the same data blocks, and increase the chances of merging sequential host writes into a larger single destage.

On the other hand, lazy writes may increase the number of modified cache pages in storage and thus decrease the number of read hits (due to the write data forcing out LRU cached read pages).

Synchronous writes are **never** delayed. A host Synchronize Cache command does not complete until all data in the synchronize cache range are destaged.

Pinned Data

When data cannot be destaged due to device errors or an inability to reach a remote ASC, the data are locked into storage and marked as *pinned data*.

Pinned data are saved to local ASC nonvolatile storage (such as local internal disk drives), until they can be destaged (once a device is repaired, or a connection is reestablished).

The ASC can be configured to stop accepting host write requests once a threshold of pinned data blocks is reached by use of the *SET MAXPIN* command. The ASC default for each device is a maximum of 128 pinned blocks.

Data Read from Cache (Hit)

This data path is used when **all** requested data are already cached.

1. The host requests data..
2. The device driver obtains shared locks for the appropriate cache pages.
3. The data are transferred from the ASC cache to the host..
4. The device driver releases the shared locks.

If possible, the host bus is held for the entire host I/O operation.

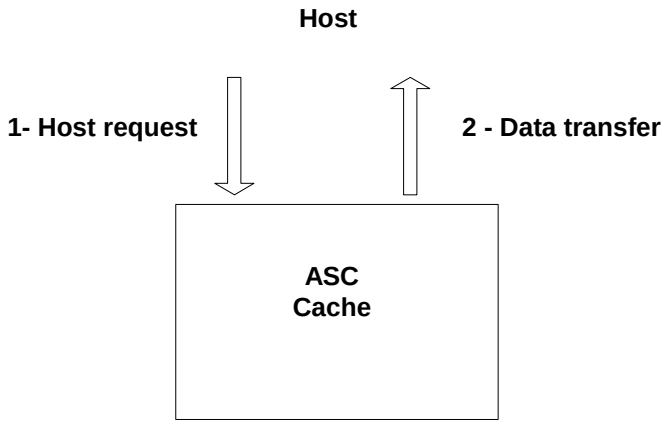

Figure 4: Data Read (Hit)

Data Read from a Local Device (Miss)

In this case, either some or all of the data are **not** cached. The data are on one or more local physical devices.

1. The host requests data.
2. If the beginning of the data is cached, lock then begin transferring this data, then disconnect from the bus. Steps 3 and 4 may run concurrently with this step.
3. A task is created to stage the missing data into cache blocks.
4. The ASC reconnects to the bus to present the remaining data. Based on the size of the request, steps 3 and 4 may be repeated.
5. When the host completes the transfer, the shared locks are released.

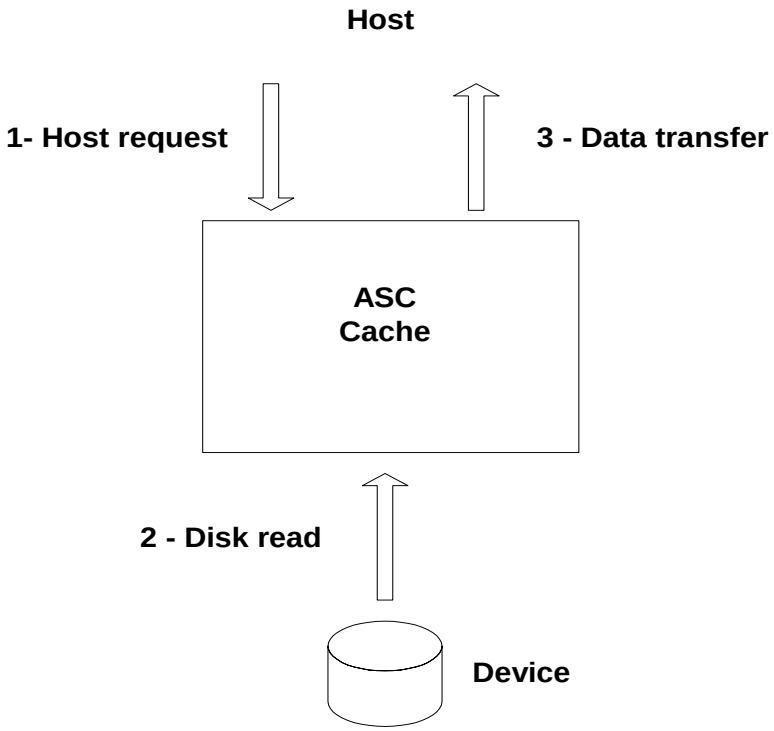

Figure 5: Local Data Read (Miss)

Data Read from a Remote Device (Miss)

Either some or all of the data are **not** cached. The data are on one or more remote devices.

1. The host requests data.
2. If the beginning of the data is cached, lock then begin transferring this data, then disconnect from the bus. Steps 3 and 4 may run concurrently with this step.
3. A task is created to stage the missing data into cache blocks. This task will call the remote device access functions to request and retrieve the data from a remote ASC.
4. The ASC reconnects to the bus to present the remaining data. Based on the size of the request, steps 3 and 4 may be repeated.
5. When the host completes the transfer, the shared locks are released.

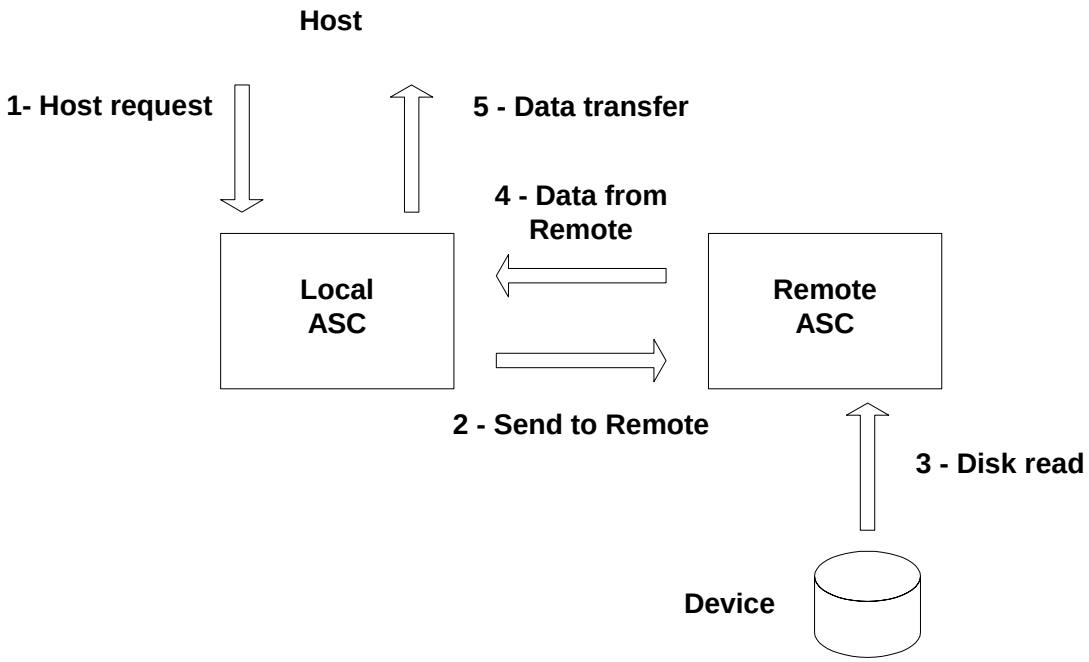

Figure 6: Remote Data Read (Miss)

Synchronous Data Write to a Local Device

In synchronous local device write mode, the host is notified of the end of a write operation when the data are written to the appropriate physical device(s). This guarantees that this write data are on the physical device **before** the host gets acknowledgement of the operation.

1. The host connects to the ASC to write data.
2. The ASC allocates cache blocks by obtaining exclusive locks for them.
3. Data are transferred from the host to these cache blocks.
4. The ASC disconnects from the host bus.
5. Assuming that this host request maps to physical device(s), exclusive locks are exchanged for shared locks, and the data are destaged to the physical device(s). When the destage completes, the shared locks are released. Based on the size of the data, and the availability of cache storage, steps 2 through 5 may be repeated.
6. At the conclusion of the destage, the ASC reconnects to the host bus and presents ending status.

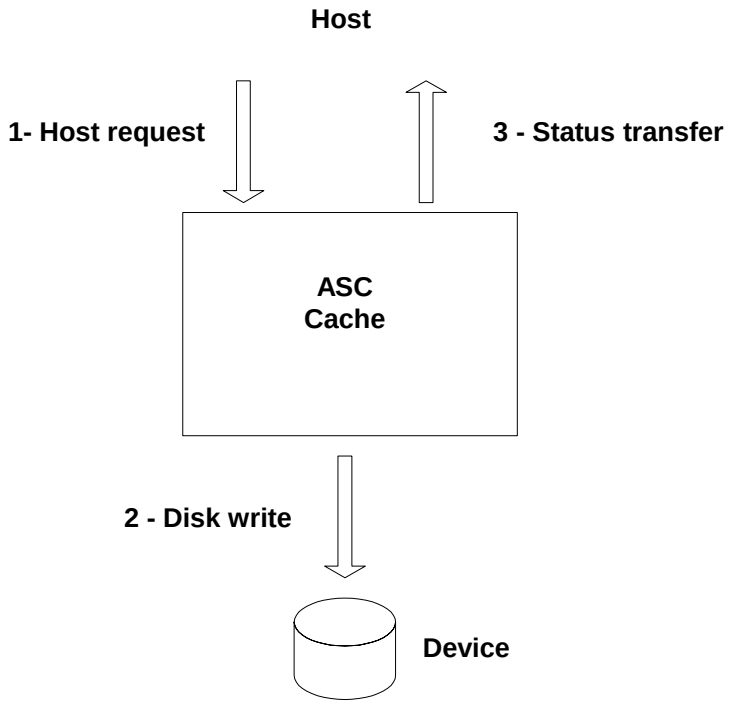

Figure 7: Synchronous Local Write

Asynchronous Data Write to a Local Device

In asynchronous write mode, the host is given completion status when the data are successfully transferred to cache. Data that cannot be destaged are considered to be locked, or *pinned* in storage and are transferred to ASC internal nonvolatile memory for subsequent host access or destage once the failing device(s) are repaired.

1. The host connects to the ASC to write data.
2. The ASC allocates cache blocks by obtaining exclusive locks for them.
3. Data are transferred to these cache blocks and their exclusive locks are released. Based on the size of the data, and the availability of cache storage, steps 2 and 3 may be repeated.
4. The ASC presents ending status to the host. Step 5 may run concurrently with this step.
5. As soon as possible, shared locks are obtained, and the data are destaged to physical devices. When the data are written to the physical device, the shared locks are released.

Based on the size of the data, and ASC configuration parameters, step 5 may run concurrently with steps 2 and 3. The ASC may be configured to delay this destage operation (allowing for more data combining and write elimination). The ASC may combine contiguous operations into a single destage operation, and may eliminate duplicate writes to the same data (obviously using the latest update).

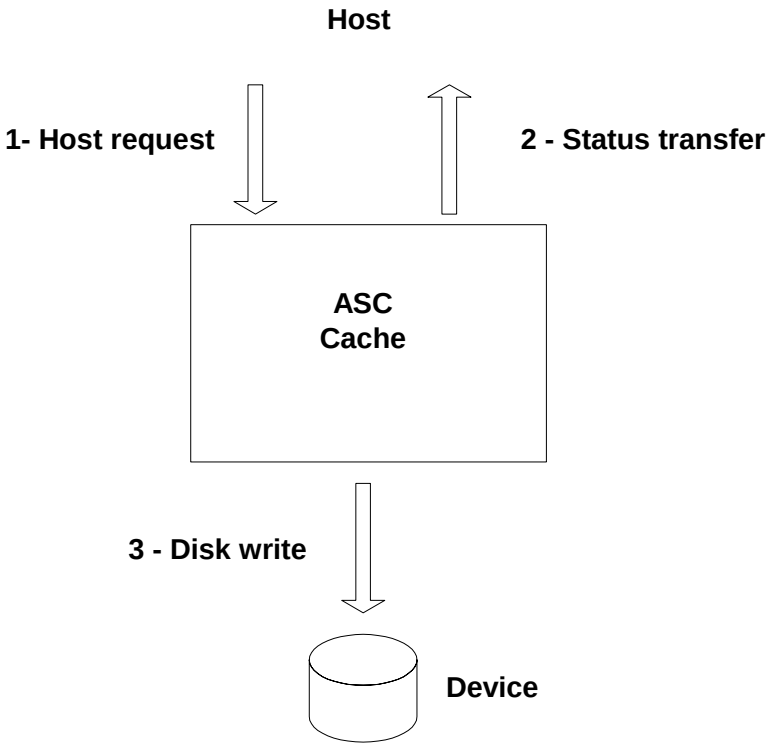

Figure 8: Asynchronous Local Write

Synchronous Data Write to a Synchronous Remote Device

In synchronous remote device write mode, the host is notified of the end of a write operation when the data are written to the appropriate **remote physical device(s)**. This guarantees that this write data are on the physical device before the host gets acknowledgement of the operation.

1. The host connects to the ASC to write data.
2. The ASC allocates cache blocks by obtaining exclusive locks for them.
3. Data are transferred to these cache blocks.
4. The ASC disconnects from the host bus.
5. Shared locks are exchanged for the exclusive locks, and the data are destaged to remote physical devices via calls to the remote device access routines. When the data are written to the remote physical device and acknowledgement of the write is returned to the local ASC, the shared locks are released. Based on the size of the data, and the availability of cache storage, steps 2 through 5 may be repeated.
6. At the conclusion of the destage, the ASC reconnects to the host bus and presents ending status.

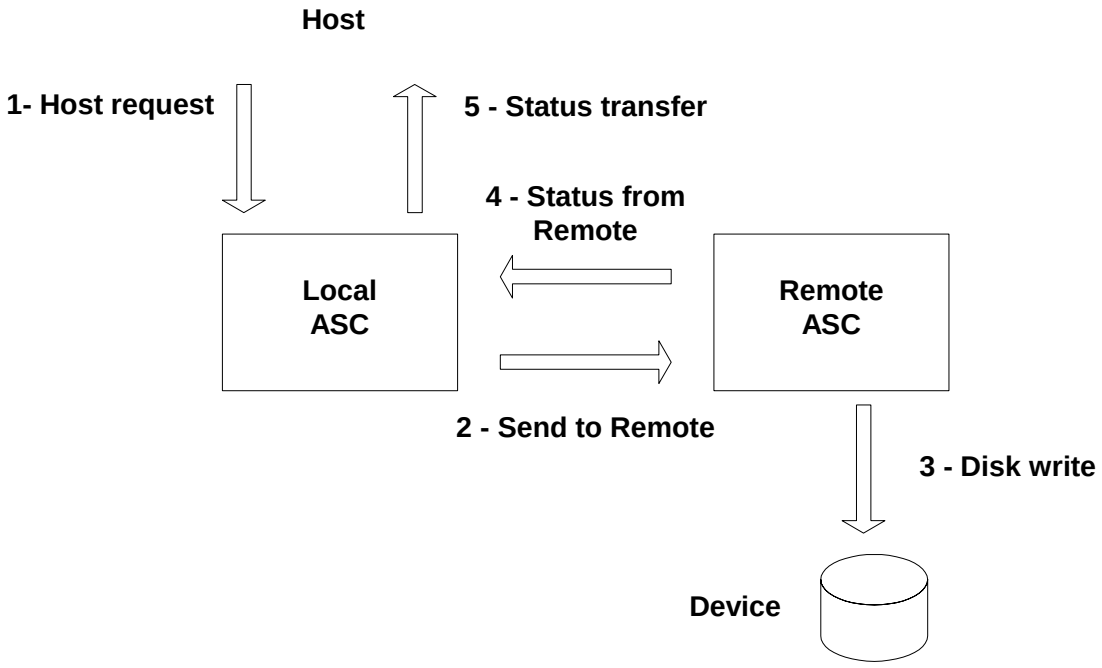

Figure 9: Synchronous Remote Write to Synchronous Remote Device

Synchronous Data Write to an Asynchronous Remote Device

In synchronous write to asynchronous remote device mode, the host is notified of the end of a write operation when the data are written to the appropriate **remote system cache**. This guarantees that the write data are at the remote ASC before the host gets acknowledgement of the operation. If the remote ASC cannot write the data to a physical device, the data are pinned on the remote ASC.

1. The host connects to the ASC to write data.
2. The ASC allocates cache blocks by obtaining exclusive locks for them.
3. Data are transferred to these cache blocks.
4. The ASC disconnects from the host bus.
5. Shared locks are exchanged for the exclusive locks, and the data are destaged to remote physical devices via calls to the remote device access routines. When the data are written to the remote cache, the shared locks are released. Meanwhile, the remote system queues the received data for destage to physical devices. Based on the size of the data, and the availability of cache storage, steps 2 through 5 may be repeated.
6. At the conclusion of the destage, the ASC reconnects to the host bus and presents ending status.

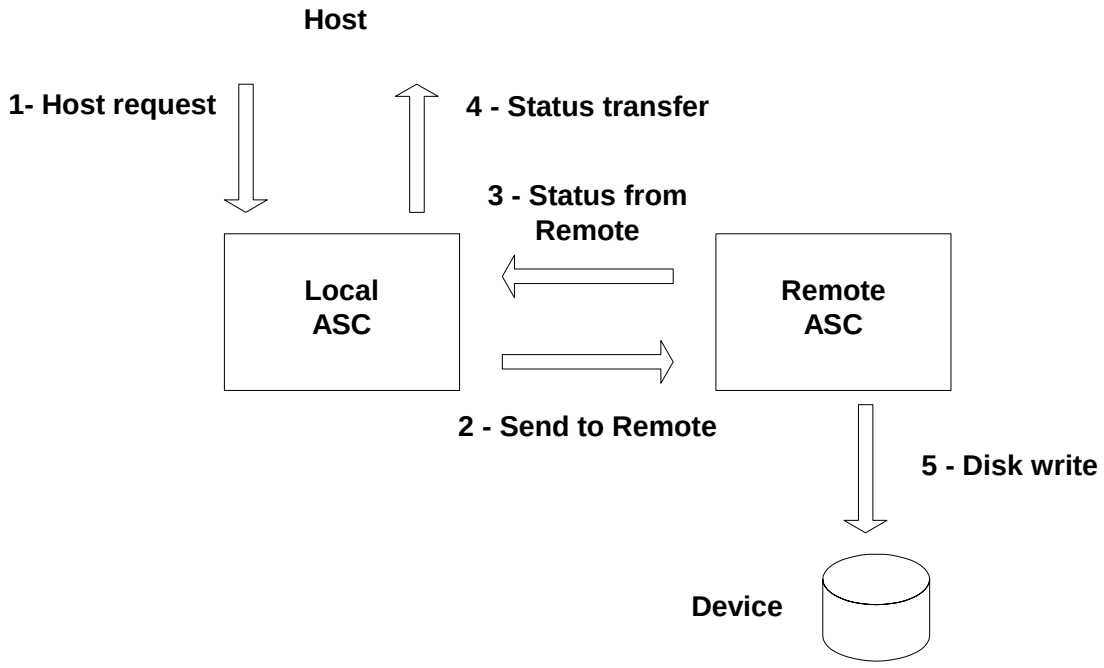

Figure 10: Synchronous Remote Write to Asynchronous Remote Device

Asynchronous Data Write to a Remote Device

In asynchronous write to a remote device mode, the host is given completion status when the data are successfully transferred to the **local cache**. Data than cannot be destaged are considered to be locked, or *pinned* in storage and are transferred to ASC internal nonvolatile memory for subsequent host access or destage once the failing devices are repaired.

1. The host connects to the ASC to write data.
2. The ASC allocates cache blocks by obtaining exclusive locks for them.
3. Data are transferred to these cache blocks and their exclusive locks are released. Based on the size of the data, and the availability of cache storage, steps 2 through 3 may be repeated.
4. The ASC presents ending status to the host. Step 5 may run concurrently with this step.
5. Shared locks are obtained, and the data are destaged to remote physical devices via calls to the remote device access routines. When the data are written to the remote cache, the shared locks are released. Meanwhile, the remote system queues this modified data to be written to its physical devices. If the data cannot be sent to the remote ASC, the data are pinned in the **local** ASC. If the data are received by the remote ASC but cannot be written to physical devices, the data are pinned in the **remote** ASC.

Based on the size of the data, and configured parameters, step 5 may run concurrently with steps 2 and 3. The ASC may be configured to delay this destage operation (allowing for more data combining and write elimination). Either ASC may combine contiguous operations into a single destage operation, and may eliminate duplicate writes to the same data (obviously using the latest update).

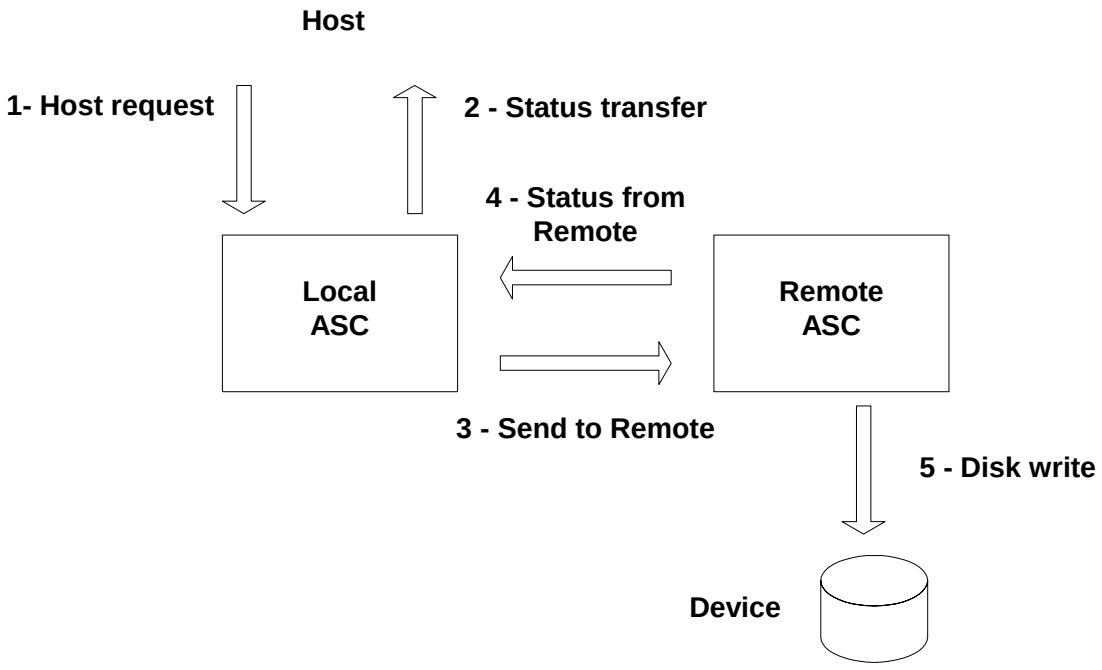

Figure 11: Asynchronous Remote Write

Data Replication/Mirroring

The ASC can create and maintain redundant copies of user data, both locally (within a single ASC), and remotely (among multiple interconnected ASC's). This facility is often referred to as *data mirroring* (needless to say, this does **not** imply that the data is "reversed" -- as in a mirror image).

ASC data mirroring is performed internally, using no host resources, that is, mirroring is transparent to the host. This allows for data replication regardless of the host environment, application or operating system. Server hardware and software resources are not consumed by mirroring.

Any number of mirrored copies can be maintained simultaneously. All of the devices containing mirrored data are called a *copy set*.

We refer to the original device in a copy set as the *primary*. We refer to each copy of the primary as a *secondary*. A *copy set* always has one primary and one or more secondaries.

We refer to any pair of volumes consisting of a primary and one secondary as a *copy pair*. Each copy pair within a copy set may be in a different state (for example, one copy pair may be established, while another is suspended).

Any number of copy sets may exist simultaneously.

It is also possible to *cascade* copies – that is, a remote ASC may copy data it receives to yet another ASC, thus creating a *bucket brigade* of copies.

The objects that the ASC mirrors are *cache devices*, that is, the logical representation of a host attached device, as opposed to physical devices. Consequently, mirroring occurs on a level above logical to real device mapping, striping, or remote device access.

Copy Pair Establish

The action where a pair of unassociated devices become a copy pair is called *copy pair establish* or just *establish*. In most cases, a copy pair is initialized by the ASC copying data from the primary device to the secondary device. Once this copying is completed, the pair is *established*. An operator creates a copy pair by using the *ECOPY* command.

An operator can specify that a copy pair is immediately established without any initial data copying. This is a more efficient way to establish copy pairs if the source volume is either empty or will have its data completely overwritten (say, by a volume restore), thereby eliminating needless copying of useless data.

A user can also specify that, on completion of copy establish, the copy pair is to be suspended or terminated. These actions can be used as preparation for backing up a volume, or to create a *point-in-time* copy of data for subsequent backup or testing.

Host writes that occur during the copy establish process are mirrored to the two volumes. Host initiated I/O takes priority over the establish operation.

Establish copying starts at the beginning of the logical volume (typically from block zero), and continues to the end of the volume. As the copy proceeds, any host writes that **completely** change a copy segment of the device are not copied (as they have already been mirrored). Generally, the size of the copy is the size of the primary device.

Copy establish is *double buffered*, that is, as the one copy segment is written to the secondary, the following segment is read from the primary. The operator can specify that the copy process delay between

segments, lowering the I/O rate (but increasing the establish time). The operator also has some control over the size of the copy segments.

It is architecturally possible to only mirror ranges of a primary device (to avoid overwriting volume labels or to only mirror one partition of a volume).

For the duration of the establish process, the secondary device is **not** a valid copy of the primary (as it is incomplete, and one cannot determine which data are old and which data are current).

Mirrored Reads

Once a copy pair is established, the ASC can read from any device in the copy set, as long as it is established and not suspended.

The ASC *SET COPYSET* command allows selection of the method the ASC chooses to direct a given read operation to a given device, for example:

Type	**Description**
Balanced	Issue reads in a round robin style among the copy set. This evenly divides I/O among physical devices in the copy set.
Closest	Issue the read to the device whose previous I/O operation block number was closest to the requested block number. This option minimizes disk head movement.
Dividedreads	Divide reads across the copy set by block number range – for example, a copy set of two devices would direct reads of the lower half of block numbers to the first device, and the higher half of block numbers to the second device. This also minimizes disk head movement.
Local	Read from local device only.
Normal	Read from the least active device in the copy set. This option optimizes bus usage.
Primaryonly	Only read from the primary device.
Remote	Read from either local or remote devices. This should only be enabled in environments with very fast network connections.

Read I/O Error Recovery

If data cannot be read from a device (due to an I/O error, or the device being suspended), the read is directed to other devices in the copy set (whether local or remote). Read recovery may attempt to fix the bad block by writing the data read from another device. It is also possible that the bad block is reassigned before being updated.

Mirrored Writes

Each host write causes *n* physical writes, where *n* is the number of established, non-suspended copy pairs in a copy set. If a write operation fails, the failing device is *suspended*, and is no longer used for I/O operations. A suspended device remains in the copy set until it is removed (by copy termination). A suspended device must be successfully re-established before it is no longer suspended.

Synchronous Mirrored Writes

If a host write is synchronous, the host is not notified of I/O completion until data are successfully written to **all** devices in the copy set. Depending on configuration, remote destaging may be considered complete when the data are successfully transferred to the remote ASC cache, or to the remote ASC device.

If the data cannot be written to a single copy set device, that device is suspended. If data cannot be written to **any** device in the copy set, the data are pinned (see *Pinned Data* above).

If copies are *cascaded* (see below) the remote system may not signal successful destage completion until **all** of the cascaded writes are complete. Thus, one should carefully consider the performance implications of combining synchronous and cascaded mirror settings.

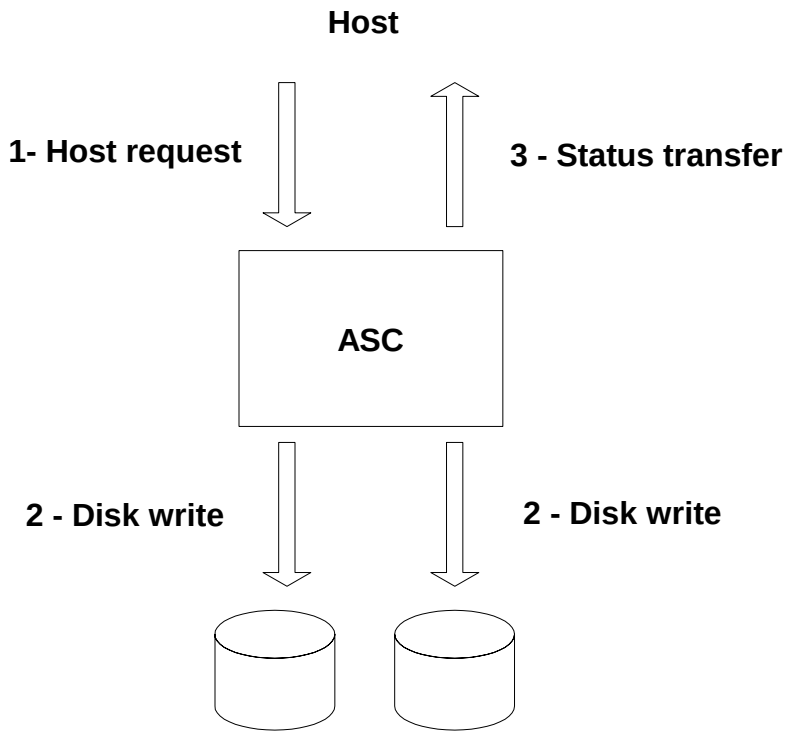

Figure 12: Synchronous Mirrored Writes

Asynchronous Mirrored Writes

Asynchronous host writes are posted as complete when the data are in the local cache. If the data cannot be written to a single copy set device, that device is suspended.

If data cannot be written to **any** device in the copy set, the data are pinned (see *Pinned Data* above).

Copy Pair Suspend

A copy pair may transition from the established (or establishing) states to the suspended state if:

1. An operator specifically suspends a device using the *SCOPY* command. Once suspended, the operator must explicitly terminate (*TCOPY*), reestablish (*ECOPY*), or relocate (*RCOPY*) the copy pair. The operator may also specify whether a spare device is to be used as an alternate secondary while the copy pair is suspended.

2. If a device write error occurs In this case, the operator is notified (via SNMP and console/log output). The copy must be manually reestablished (*ECOPY*).

3. If a device becomes inaccessible. The ASC polls offline devices, and if the device becomes accessible, is the same device, and again appears operational, the copy is automatically reestablished.

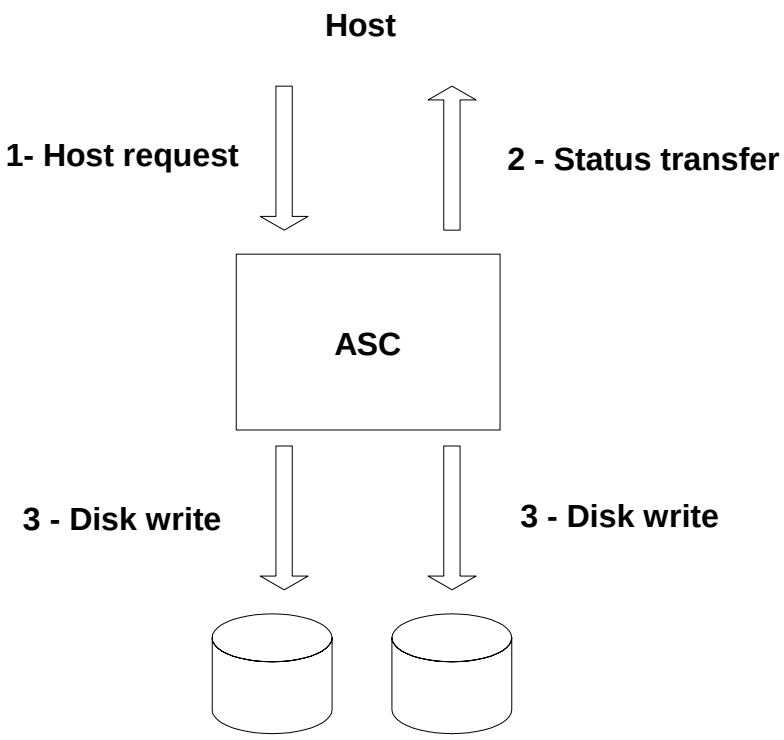

Figure 13: Asynchronous Mirrored Writes

4. If a remote connection is severed. In this case, the ASC may automatically reestablish the copy set when the remote connection is again established. This alleviates the need for operator intervention during sporadic temporary line failures or timeouts.

5. If performance parameters are set to suspend the copy set on the occurrence of an event, for example if the amount of host write activity exceeds the ability for remote connections to transfer the data in an acceptable time (*SET SUSPMODC*). In this case, the ASC will automatically reestablish the copy pair when the performance threshold falls below a settable value.

Copy Pair Map

While a copy pair exists, a *copy differences map* is maintained reflecting the blocks that are identical, or are *at parity* with the primary device. This map is kept in ASC storage, is saved to internal hard disk when the system is shutdown or restarted, and is restored when the system is booted.

When a suspended copy is reestablished, this map is used to determine the ranges of device blocks that must be copied from the primary to the secondary (or vice versa, if the primary is the device that failed).

The copy map contains a *counter byte* for each copy set block range. The size of the range depends on the size of the cache device. Typically a range varies from eight to 128 4,096 byte cache blocks. If **any** block in the range is not at parity, the **entire** range is copied.

Before any destage occurs, the copy map is adjusted to reflect the data that are no longer at parity. When the destage completes successfully, the map is readjusted to reflect that the data are again synchronized.

Critical Copy Sets

Sometimes the existence of a functional copy set will be so critical that host applications should not be allowed to run without it. Or, it may be essential that a remote copy exactly match the local copy at all times

For these cases, the ASC can be configured such that if **any** device in a copy set is suspended, then the cache device becomes not ready to all hosts. Each copy set can be individually configured using the *SET SUSPSTOP* command..

An operator must repair and reestablish the copy set (*ECOPY*) before the cache device becomes ready and host I/O is again accepted.

Copy Pair Re-establish

A suspended copy pair is brought back to parity by a copy *reestablish* (using the same operator command as establish – *ECOPY).*. This process is virtually identical to an initial copy establish. A reestablish **always** copies data to the suspended device.. A **replaced** device must be terminated then established.

If the failing device was a secondary, data are copied from the operational primary to the reestablishing secondary. If the failing device was the primary, the data are copied from the first operational secondary (or any working secondary in the case of a read I/O error) to the reestablishing primary.

At the successful conclusion of a copy reestablish, the copy pair is again established. If the reestablish fails, the pair returns to the suspended state.

If the original primary was suspended, but is now functional and completely reestablished, it again becomes the copy set's primary device. If any internal device swap was in effect because of the suspend, the device swap is cleared.

For the duration of the establish process, the secondary device is **not** a valid copy of the primary (as it is incomplete, and one cannot determine which data are old and which data are current).

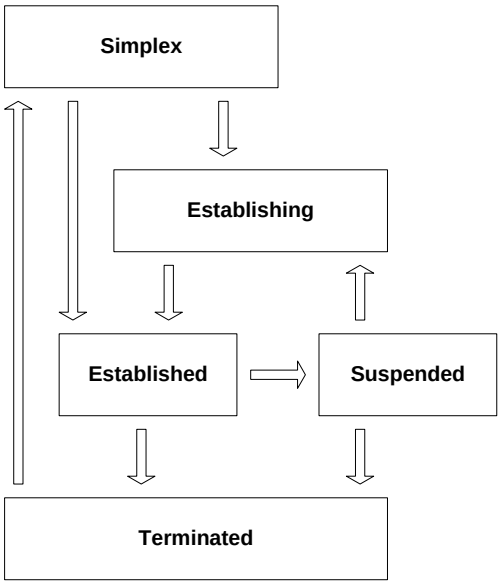

Figure 14: Copy Set State Transitions

Copy Set Spares

Spare devices can be used as alternate secondary devices If the system suspends a copy pair, or if a user suspends a copy pair and requests that a spare be assigned, a spare volume will be assigned, and a copy establish to that spare will be automatically started.

If more than one spare exists, the spare that is equal in size to the primary device will be chosen, otherwise a spare that is bigger than the primary, but closest in size to the primary will be chosen. A remote device can be a spare, but this is not recommended (unless it is a spare for a remote secondary).

When a suspended pair is successfully reestablished, the spare is released and returns to the spare pool (the spare is not erased, but, being a spare, is inaccessible to host I/O operations).

For the duration of the establish process, the spare secondary device is **not** a valid copy of the primary (as it is incomplete, and one cannot determine which data are old and which data are current).

Secondary Device Relocation

To minimize the time required to establish a remote device, it is possible (but not necessary) to establish a device locally (*ECOPY*), suspend it (*SCOPY*), physically move the device to the remote location, then reestablish it there (*RCOPY*). This eliminates the time and network activity otherwise required to bring a remote device to parity.

The copy map is maintained while the device is in transit. When the device is attached to the remote system, data that were modified are copied across the network to the device.

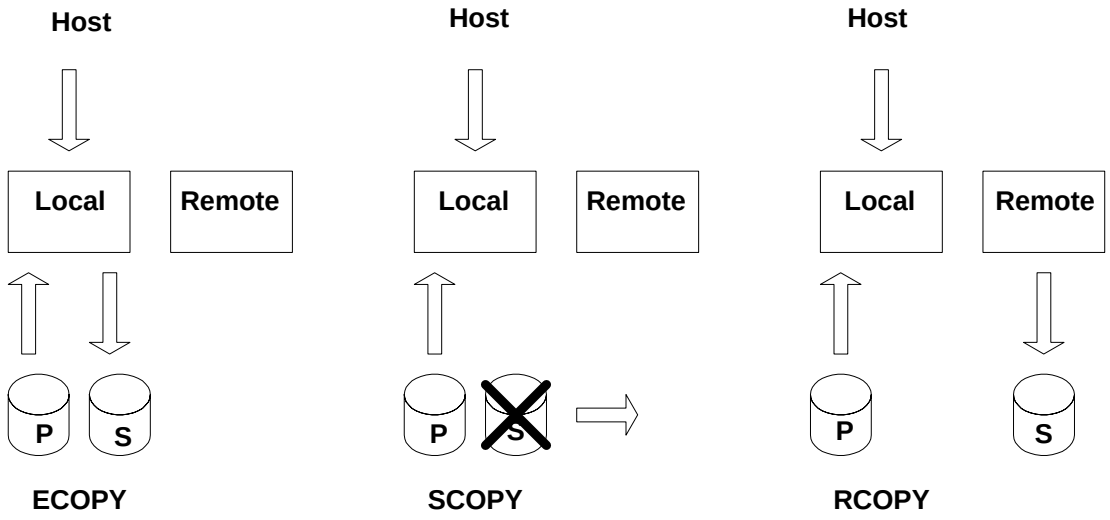

Figure 15: Secondary Device Relocation Process

Cascaded Mirrors

A **remote** secondary device can be used as the a primary of another copy set. This allows the remote ASC to further mirror data to another local device, or yet another remote system. This is called *cascaded mirrors.* Local secondary devices **cannot** be used as primary devices of other copy sets.

Cascaded mirrors allow data to be passed along from system to system, rather than requiring the initial system to retransmit the same data to the same or multiple locations.

For example, if you want two copies of data on a remote system, you could define both devices as remote secondaries on the local machine (in which case the local machine would send two copies of the data across the network). Alternately, you could define one remote mirror on the local machine, and a cascaded local mirror on the remote machine. This way, one copy of the data crosses the network, and the remote machine destages the data to two physical devices.

There is no logical limit to the number of cascades that can be performed, however:

1. Each cascade adds to the end-to-end time it takes to complete a destage. The performance impact is especially apparent on synchronous host writes.

2. The local system is unaware of the state of cascaded copies (as these copies do not belong to the local system).

3. One should not cascade ASC's in loops, for example, data from ASC *A* to ASC *B* should **not** be cascaded back to ASC *A*. Aside from being silly, this could result in resource deadly embrace – each ASC waiting for the other to release resources.

One use of cascaded mirrors is to distribute content over a network to many servers.

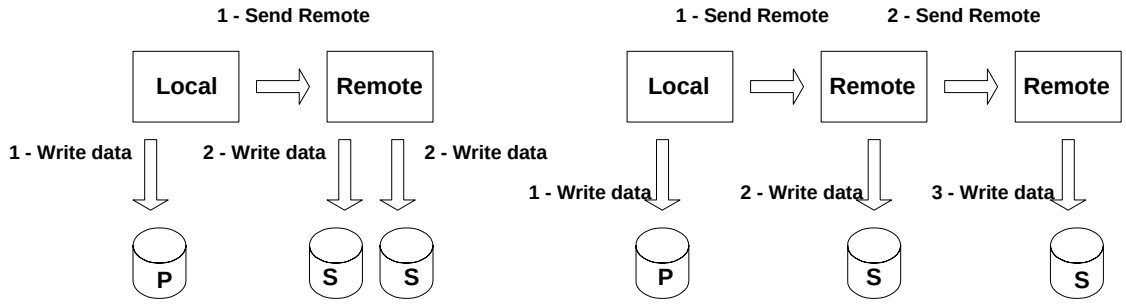

Figure 16: Cascaded Mirrors

Host Access to Copy Sets

The following table summarizes host access to members of copy sets. This table assumes that the primary and the secondary devices have been configured to allow read/write host access.

Copy State	**Which Device**	**Host Access**
Not in a copy set	N/A	Read/write.
Establishing or established	Primary	Read/write.
	Local Secondary	Not accessible (check condition – not ready, operator intervention required).
	Remote Secondary	Read/only to hosts on the remote ASC – data coherency is indeterminate if host applications are writing to this copy set (because of potential data in transit on the network at the time of the host read).
Suspended	Primary	Read/write.
	Local suspended device	Read/only if functional, data consistent to the time of the suspend.

	Remote suspended device	Read/only if functional, data consistent to the time of the suspend.

Remote Data Access

ASC's can reserve and utilize physical devices attached to other ASC's via TCP/IP network connections. This section describes this support.

Each *physical device* configured for use by an ASC may be either a physical device attached to this ASC (*local device*) or a physical device attached to some other ASC (*remote device*). One (or more) ASC's can access devices on other ASC's. ASC's can *reserve* and access each other's devices, allowing for (among other things) *cross system mirroring*.

Remote Device Reservations

Remote devices are devices that are physically attached to ASC's other than the one controlling the device. We refer to the *local ASC* as the ASC *owning* (or *sharing*) the device. We refer to the *remote ASC* as the ASC physically attached to the device. All interconnected ASC's are full partners and can cross-access multiple devices.

ASC's do not have to be directly connected. ASC's take advantage of various Internet Protocols (*IP's*) to route ASC-to-ASC messages from the one ASC to another (and vice versa).

Every ASC-to-ASC *connection* is defined in both the local and remote ASC configurations. This insures that only *authorized* ASC's can reserve and control remote devices. It is possible to restrict ASC access by physical network connection path, thus reducing the possibility of processing imposter, or *spoofed* messages. Adding an ASC to a configuration implicitly grants that ASC the authority to reserve devices on this ASC.

Access to a remote device is defined by:

1. Adding the **remote** ASC's IP address or name to the **local** ASC's configuration's *TCPIP* statement.

2. Adding the **local** ASC's IP address or name to the **remote** ASC's configuration's *TCPIP* statement.

3. Mapping all or parts of a local cached device to a remote device. The system IP address or name, and its physical connection on the remote ASC define the complete physical location of the remote device.

When an ASC-to-ASC connection is established, each eligible remote device is queried and reserved by the local ASC. The mechanism for doing this is described below.

Devices may be reserved as read/write (implying that the reserving system is the only one authorized to write to the device), or read/only (implying that the reserving system can share the device with other reserving systems). Read/only access allows data sharing, for example, allowing multiple remote host access to a single physical CD ROM drive.

TCP/IP Protocol

ASC network packets are TCP/IP compatible. This means that an ASC can communicate over any TCP/IP network, including existing LAN's and WAN's regardless of speed or distance.

Specific implementation details of the RTOS TCP/IP protocol may be found in the *Design of a Reduced Function Operating System*. Some ASC application related specifics are described in this section (these

notes do not necessarily refer to RTOS implementations of Internet standard protocols, such as Telnet, FTP, and so on).

TCP/IP is a *stream* protocol, that is, the protocol transmits a two-way stream of individual bytes (as opposed to specific messages). ASC-specific protocols use TCP/IP as a transmission medium, but transmit one complete message per TCP/IP *packet* (this, of course, is transparent to the network and the protocol).

As the maximum Ethernet packet size is 1,514 bytes, and a number of bytes are required for MAC, IP, TCP and ASC headers (typically no more than 86 bytes), up to 1,024 bytes of user data are transmitted per packet (typically two 512-byte data blocks). The ASC does not support IP packet fragmentation (and marks all transmitted packets "don't fragment"). As each TCP/IP message is self-contained, outgoing packet fragmentation is not necessary.

To maximize ASC performance, user data are transmitted directly from local ASC cache buffers – no data are moved to RTOS TCP/IP buffers. The RTOS gives the application tasks the responsibility for handling TCP acknowledgements and retransmissions of lost packets. This flexibility allows the ASC to greatly minimize network latency and turn-around time. This also obviates the need for many generalized TCP *fat thick pipe* algorithms (as the characteristics of the ASC packets are well known by both ends of the connection)

ASC supports and initiates Selective Acknowledgement (*SACK*). The RTOS maintains last ACK received and SACK information, and only retransmits unacknowledged packets.

Each message advertises a window size of 32,768 (8000 hex) bytes.

To help prevent spoofing, all TCP connections are opened using pseudo random initial sequence numbers.

TCP timestamps are used to extend the 32 bit sequence number size and thus prevent delayed message sequence number confusion on very high speed (such as gigabit) connections.

ASC-to-ASC Protocol

ASC's set up a number of Transmission Control Protocol (*TCP*) *streams* or *sockets* between two ASC systems. This section describes the TCP *control connection*.

One TCP/IP stream is created to send control messages between the ASC's. By default, this stream is opened on both sides with port number 8192 (2000 hex) – this value can be changed by the ASC configuration *SYSPORT* statement.

Each ASC opens a TCP connection in *passive open* mode. ASC uses two methods to detect working connections between systems:

1. ASC-to-ASC periodic Address Resolution Protocol messages (A*RP's*) are used to identify functional physical connection paths from one ASC to another.

2. ICMP echo requests/replies (*pings*) are used to locate *routed* (indirect) connection between ASC's.

Once a logical connection is sensed, the ASC converts the passive open to an *active open* and attempts to establish a TCP stream.

Once the control stream is established, a series of ICMP echo messages and TCP/IP messages are sent to:

1. Determine the average round trip time (*RTT*) and the number of *hops* (intermediate notes) between the two ASC's. This is accomplished by exchanging a number of pings containing ASC-known data.

Based on this information and the ASC configuration, the remote device TCP/IP protocol between systems may be altered to take best advantage of the characteristics of the connection.

2. Request and return device information (SCSI inquiry data) for each locally configured remote device attached to this remote ASC.

3. Reserve each remote device. The local ASC sends the TCP port number it will use for this stream. The remote ASC creates a TCP stream for the device (a passive TCP open), and returns the TCP port number it allocated. When the local ASC receives the reserve reply, it opens the stream (using an active TCP open).

Once a device stream is opened, the remote device is ready for use – if a simplex device, the device becomes available to local hosts. If a member of a copy set, it is reestablished.

Other messages may be sent on this TCP/IP stream. For example, one or more ASC's can be designated as *timeservers*. These ASC's will send periodic date and time messages to closely synchronize the clocks on an ASC collection (there is no need for this – it just helps when inspecting log entries generated by a group of ASC's). Also, occasional *keep alive* messages may be sent on this stream.

If an ASC-to-ASC stream is lost, remote device streams are aborted and closed. The local system marks these devices *not ready* to prevent subsequent host access, and suspends any copy pairs associated with them. The remote system drops appropriate device reserves. Both ASC's then start new control connections with passive opens. When the systems again appear to be connected, the control connection procedure begins again.

The ASC-to-ASC stream is the only stream in the ASC protocol to (potentially) issue simultaneous TCP active opens (as each system may detect the presence of the other simultaneously). Occasionally, this will result in a number of back off's and retries. A random delay is inserted between retries to minimize the number of times this occurs.

Virtually no communications take place on the ASC-to-ASC connection once remote device reserves are complete, however, the connection remains open for additional traffic such as device re-initialization (if a device connection fails), and occasional keep-alive messages (to ensure that the connection is still good)..

Remote Device Protocol

Each remote device has one or more TCP streams associated with it. These streams are created with port numbers in the range of 12,268 (3000 hex) through 65,535 (FFFF hex). The starting port and port range may be altered by the ASC configuration *SYSPORT* statement.

Each stream is explicitly opened during the remote device reservation phase (see above). Specific port numbers for a given connection are also exchanged.

Each remote device TCP stream is used to:

1. Transmit requests for device data.

2. Transmit solicited device data in response to a request for data. This is typically done when reading (staging) remote data to a local ASC.

3. Transmit unsolicited device data. This is typically done when writing (destaging) local ASC data to a remote ASC.

4. Transmit completion status of an operation, for example, completion of a request, including any state change or error status.

5. Throttle the other system (to temporarily slow down incoming requests).

6. Transmit unexpected device status, such as a remote device coming on line, going off line, and so on.

Remote Device Data Read

Each host (or establish copy) data read request is passed on to the appropriate remote device task. This task and the remote ASC do the following:

1. A request is sent to the remote system on a device TCP stream requesting the data, passing the starting block and block count.

2. Data are returned in up to 1,024 byte chunks (typically two 512-byte disk data blocks).

3. When all of the data are received, the local system acknowledges the entire reply and posts the waiting data requestor.

4. If a timeout occurs before all of the data are received, the local system retries the request. If the *data received* acknowledgement was not received, the sender retransmits the unacknowledged data. If, after a number of retries, the data cannot be received, the TCP connection is restarted, and, if a connection no longer exists, the requesting application receives a device I/O error, and if part of a copy set, the copy set is suspended.

Based on configuration, a request may be split into smaller consecutive requests (*SET TCPSIZE* – no default value). Also, based on the speed of the connection, artificial delays may be placed between each message sent in step 2 (*SET TCPDELAY* – the default is no delay), and/or the sending system may request (and wait for) TCP acknowledgements between each message (*SET TCPLS* – the default is off).

Every TCP stream is serial, therefore, a stream can handle only one I/O request at a time. If *n* streams are open, *n* simultaneous requests can be processed.

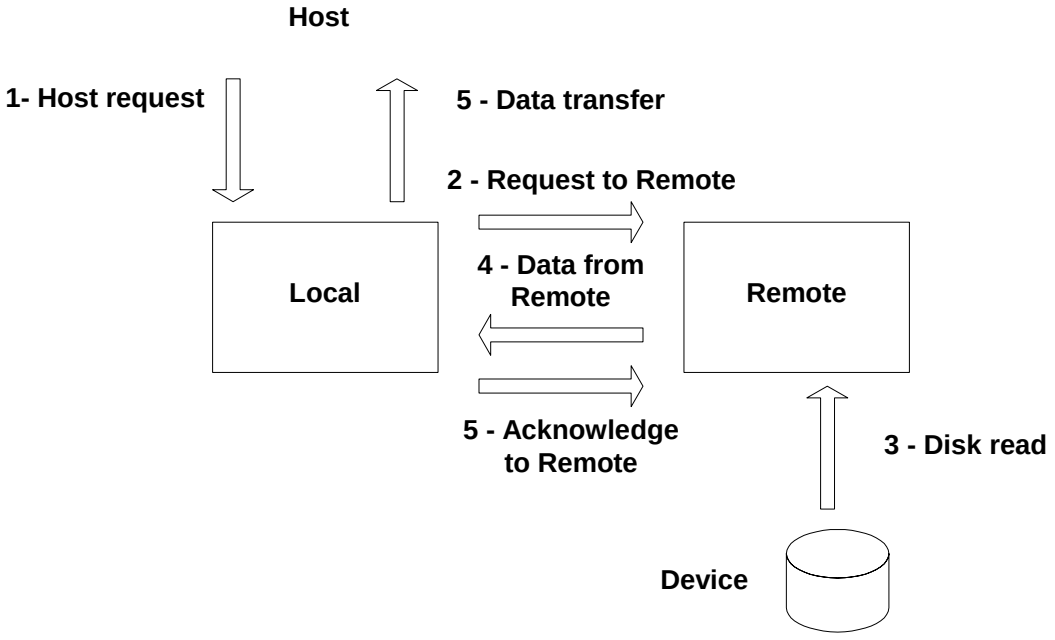

Figure 17: Remote Data Read Protocol

Remote Device Data Write

Each host (or establish copy) data write request is passed on to the appropriate remote device task. This task and the remote ASC do the following:

1. Data are sent in up to 1,024 byte chunks (typically two 512 byte disk blocks). The remote system moves these data into local cache blocks and marks the blocks as containing modified data.

2. The header of the last data block transmitted includes information describing the entire write operation (starting block, block count, write type (synchronous vs. asynchronous), verify, etc.).

3. Based on the write (destage) type, the remote system either:

   If an asynchronous write, the remote sends a destage complete message (implicitly performing a TCP acknowledge), then queues it for subsequent destage.

   If a synchronous write, the remote sends a TCP acknowledge message, destages the data, then sends a destage completion message.

   For other operations, the remote sends a TCP acknowledge message, performs the I/O operation requested (such as data verify), then sends an operation complete message.

4. If a timeout occurs before all of the data are acknowledged, the local system retries the transmission. If the destage complete is not received, the sender retransmits the unacknowledged data. If, after a number of retries, the data cannot be transmitted, the TCP connection is restarted, and, if a connection no longer exists, the requesting application receives a device I/O error, and if part of a copy set, the copy set is suspended.

Based on configuration, a request may be split into smaller consecutive requests (*SET TCPSIZE* – no default value). Also, based on the speed of the connection, artificial delays may be placed between each message sent in step 2 (*SET TCPDELAY* – the default is no delay), and/or the sending system may request (and wait for) TCP acknowledgements between each message (*SET TCPLS* – the default is off).

Every TCP stream is serial, therefore, a stream can handle only one I/O request at a time. If *n* streams are open, *n* simultaneous requests can be processed.

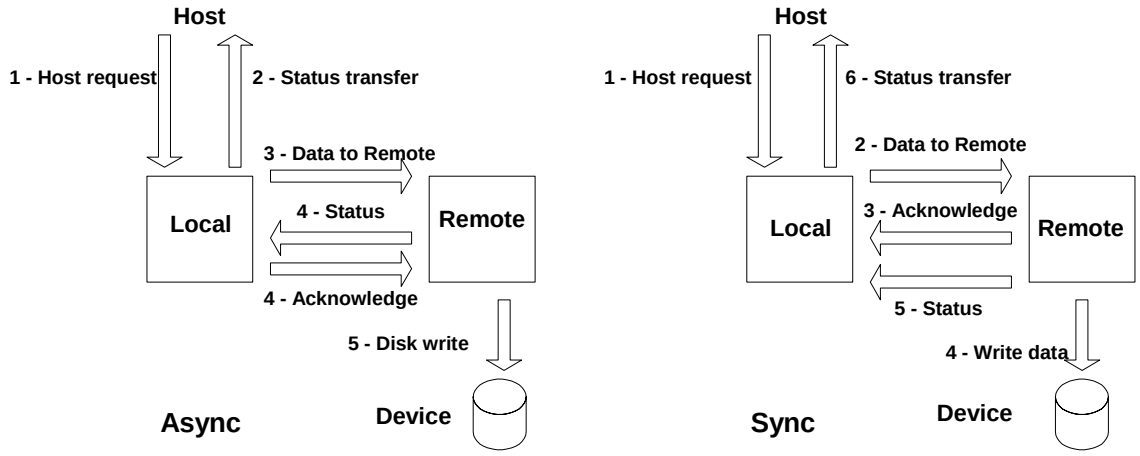

Figure 18: Remote Data Write Protocol

Routing

The ASC supports *static routing*, that is, the ASC configuration (*ROUTE*) defines the IP node(s) (if any) used to transmit messages to other ASC's and hosts. The routing configuration provides the ability to direct messages through a number of destinations, including one or more *default destinations*.

The ASC can store and forward messages destined for another node that is known to the ASC and is on a physical path other than the one receiving the message.

The ASC supports any number of physical paths, and has the ability to maintain parallel connections for both performance and redundancy.

The ASC can be defined as *multi-homed*, that is, it can support multiple IP addresses.

Pings, ARPs, and Timeouts

The ASC issues periodic Internet Control Message Protocol (*ICMP*) echo requests (*pings*) to:

1. Detect when an ASC is logically connected.

2. Determine the *round trip time* and the number of intermediate nodes (*hops*) between ASC's.

The frequency of both pings and ARPs can be altered by configuration or operator command (*SET ARPFREQ* – the default is five seconds).

The ASC issues periodic Address Resolution Protocol (*ARP*) messages to:

1. Determine which physical paths are connected to other ASC's, hosts, and/or routers/bridges, etc.

   Broadcast ARPs are sent to discover unknown connections.

   Specific ARPs are sent to test known connections.

2. Test the functioning of an individual physical path. ARP replies are always received on the physical path of the transmitted ARP request. If a number of ARP requests do not generate an ARP reply, it is assumed that this connection is inoperative (or is partially operative, for example, it can transmit, but not receive).

Virtually all ASC TCP/IP messages (or message sets) are sent by a routine that waits for either a reply or an expiration of a period of time – typically about one second. Any timeout condition results in a number of (potential) message retransmissions – typically about five. If multiple physical paths exist between two systems, retransmissions are attempted on a path **other than** the one originally sending the message.

UDP Messages

The ASC can be configured to allow certain ASC-to-ASC User Data Protocol (*UDP*) messages. These include:

- Issue an operator command on another ASC. Port 4,096 (1000 hex) is used by default.

- Send a message to an operator on another ASC (via the console display). Port 4,097 (1001 hex) is used by default.

- Receive a reply to an operator command from another ASC. Port 4,098 (1002 hex) is used by default.

Any or all of these messages may be disabled (for security reasons) via the configuration *SYSPORT* statement.

Remote Device Performance

ASC network packets are TCP/IP compatible. This means that an ASC can communicate over any TCP/IP network, including existing LAN's and WAN's. However, the performance requirements for these packets are unique, and may require some careful planning and tuning to achieve optimal ASC performance without adversely impacting the existing network.

Typical TCP/IP streams exist for a short period of time, and send a small number of very small packets. For example, an HTTP connection typically transmits a very short request, receives a relatively small number of data packets, and then (usually) closes the connection. FTP data connections typically only exist long enough to transfer a single file.

ASC TCP/IP streams are open indefinitely, and transmit large amounts of data in (relatively) large packets. Good remote performance requires high throughput and short response times.

Three major factors significantly effect remote ASC performance:

1. Physical connection speed.

2. Physical connection latency.

3. Physical connection reliability.

Physical Connection Speed

The speed (in bits per second) of a connection defines the maximum amount of data that can be transmitted over time. To calculate the minimum acceptable connection speed between ASC systems, the following need to be taken into account:

1. The *quantity* of remote activity. For example, if a remote device is only accessed as a copy pair secondary, then (as long as the local device is working) only host write data will be transmitted over the remote connection.

2. The *frequency* of remote activity. A system with a high percentage of write activity will need more bandwidth than a system with very little write activity.

3. The *clustering* of remote activity, that is, whether the remote I/O load is constant over time, or varies dramatically..

   Assume that ten percent of all host I/O are writes, and all requests transfer the same amount of data. Then, a line that is at least 10% as fast as local busses is sufficient to handle the I;O load (actually it would have to be a bit higher to take message header overhead into account). However, if the write/read ratio changes significantly over time (such as during database reorganizations), then lines must be able to support acceptable response times at peak network loads.

Physical Connection Latency

Latency may be loosely defined as the *end-to-end* or *round-trip* time between when a message is sent, to the time a reply is received. Connections needlessly waste potential bandwidth if they incur long turnaround delays. Latency tends to be introduced by intermediate nodes that buffer, then retransmit packets.

The ASC protocol attempts to minimize the effects of latency by requiring the fewest possible acknowledgements. Under most circumstances, this is a performance benefit. However, it could result in large retransmissions if a significant number of packets are lost – the fewer acknowledgements, the more data sent between acknowledgements.

The best way to minimize latency is to have the smallest number of nodes (*hops*) between ASC systems. It is also a good idea to insure that intermediate node buffers and processors are not overloaded..

Physical Connection Reliability

The ASC attempts to minimize delays incurred by retransmission of *lost* packets. Individual packets can be lost, or they can lost in groups. Outgoing packets, incoming packets, or both may be lost.

Here are some of the places where packets are typically lost:

1. Local Ethernet traffic causing *collisions* (typically caused by several devices simultaneously transmitting, or attempting to transmit, resulting in overrunning hubs).

2. Fast Ethernet traffic overrunning local device buffers or processors (routers/bridges).

3. Traffic overrunning *funnels* (places where high bandwidth connections meet low bandwidth connections, for example, in a device that bridges Ethernet to a T1 or T3 line).

4. Packets discarded by the network due to low **guaranteed** service level settings – typically occurring in Frame Relay environments.

5. Failing connections – due to faulty equipment, poorly conditioned lines, or overtaxed resources.

One should carefully inspect the network map for the path(s) data are to take between ASC nodes (often the network configuration may "change out under you" without notice). Check the path in **both** directions – it is possible that the network routes returned data differently than transmitted data. The *TRACERT* command can be used to display the path between two ASC's.

Performance Aids

Here are some of the tools that can be used to tune ASC remote performance:

1. A multiple packet transmission (such as a host destage request) can be configured to be entirely transmitted on a single physical path, minimizing *out-of-order* packet reception (*SET TCPPATH* – defaults to on).

2. The maximum size of a logical group of packets can be defined (*SET TCPSIZE* – no default). Smaller values may lower the chances of overrunning routers, bridges, and hubs.

3. A delay can be introduced between each transmission (*SET TCPDELAY* – defaults to zero). Non-zero values lessen the chances of overrunning routers, bridges, and hubs, however, they can significantly lower bytes transferred per second.

4. The ARP and ping times can be adjusted (*SET ARPFREQ* – defaults to five seconds). A larger value helps prevent false detection of failing connections, however it also causes a longer delay in detecting actual failed connections.

5. The message reply timeout values can be configured (*SET TCPTIMES* – defaults to one second). The faster the value, the quicker lost packets will be detected, however, smaller values may result in unnecessary timeouts and message retransmissions.

6. *PING* can be used to determine the number of hops and round trip time between systems.

7. Route tracing (*TRACERT*) can be used to map the path between two systems.

8. Statistics can be used to determine average, minimum, and maximum times by connection, stream, and system (*TCPB, RMTBLK*).

9. Statistics can be used to view collision and error rates by physical connection (*LINESTAT*)..

Hubs

Ethernet hubs typically have multiple connections with an aggregate bandwidth of the **slowest** Ethernet connection attached. Hubs echo **all** data received on one connection to **all other** connections on the hub.

ASC Ethernet ports **can** be connected to Ethernet hubs. Consider the following when doing this:

1. If you attach multiple ASC paths to the **same** hub, the hub will prevent the ASC from using paths simultaneously, effectively cutting aggregate speed down to a that of a single connection.

2. As the hub can only process one packet at a time, any attempts at sending other packets at the same time will result in collisions. Usually, hardware retries eventually get the data transmitted. However, packets that cannot be transmitted will be treated as lost messages or timeouts, resulting in delays and needless retransmission overhead.

3. As hub connections are not point-to-point (by definition), hardware pause packets will neither be transmitted nor received. Thus, they cannot be used to throttle Ethernet activity (see below).

4. The ASC on a hub has to compete for bandwidth with all other devices connected to that hub.

Generally speaking, it is almost always better to directly attach an ASC to another ASC, or to a switch or router. The *SET HUB* command can be used to help an ASC perform better when attached to a hub. If the ASC receives its own transmitted broadcast message on another physical port, it automatically sets hub mode on.

Pause Packets

The ASC supports Ethernet *pause packets*. A network device may send pause packets when it begins to run out of internal storage space to buffer incoming data. The sending device, on receiving a pause packet, delays the transmission of subsequent data until the pause message time interval expires, or until a *zero-delay* packet is received.

Pause packets help to decrease unnecessarily discarded packets. If an ASC is attached point-to-point to a device that optionally supports the sending of pause packets, it is a good idea to enable this option.

Pause packets only exist on high speed (100 megabit or greater) Ethernet.

The ASC *LINESTAT* command can be used to display pause packet reception and transmission counters.

Data Protection

This section describes some of the methods ASC uses to protect user data.

Hardware Protection

All of the usual hardware assisted protections such as parity, error detect and correct memory, and so on are requirements for a suitable ASC hardware platform. Specifically the following are minimum requirements:

1. Error checked and corrected (ECC) storage.

2. Parity enabled system and PCI busses.

3. Parity enabled processor paths.

4. Parity enabled SCSI busses.

The ASC will enable all error detection, recovery, and reporting on supported platforms. Generally, recovery is performed by a number of retries. Uncorrectable errors cause an error log and a shutdown of the system.

Internal Disk Storage

At minimum of two redundant mirrored internal disk devices are supported for saving vital information in the event of a system crash, power outage, or shutdown.

The following data are saved:

1. Pinned data (data that cannot be written to user media).

2. Copy set state, including suspended device change maps.

In addition, the internal drives can be used to keep:

1. Multiple copies of the operating system and application code.

2. Multiple user configurations.

3. Console and error log files.

4. System failure storage dumps.

5. A basic file system (see *HTTP* below)

UPS Control and Monitoring

The ASC architecture assumes that **all** memory is *non-volatile*, that is, that its contents will never be lost. To insure this, the ASC **requires** that hardware be provided to guarantee the state of all memory. This is usually done by employing redundant power supplies and by providing battery backup in the event of an external power failure via an *uninterruptable power supply* (*UPS*).

The ASC can control and monitor up to three simultaneous uninterruptable power systems connected through either a proprietary interface or to serial ports. These are used to guarantee power to the system for a short period of time in the event of a main power or power supply failure (the **minimum** battery time should be at least three minutes).

When a power failure (or a UPS battery low condition) occurs, the ASC restricts host writes to synchronous mode, minimizing the amount of host-modified data in cache. Optionally, if the duration of a power failure

exceeds a user-configurable time interval (*SET POWEROUT* – default is 30 seconds), the state of the system is saved to local disk, and the system is shut down.

If the power fails, **and** the UPS battery is (or becomes) low, the state of the ASC is preserved, and the system is immediately shut down (the UPS may also be shut off to preserve battery life).

Environmental Monitoring

The ASC supports the Intelligent Platform Management Interface (*IPMI*). This protocol is used to query the environmental state of the system, such as processor and board temperatures, fan function and speed, voltages, currents, and so on.

The ASC can be configured to report on any out-of-band conditions, and to save state and shut the system down on critical deviations.

A command is provided (*IPMI*) to display the current, low and high values for each system sensor.

Tools

The ASC includes a number of built-in tools to ease installation, perform diagnostics, and measure performance. Some of these are described below. Most tools require authorization to run (see *Operator Commands* below).

Device Utilities

Standalone and online utilities are provided to initialize devices, test devices, measure device performance, and determine device characteristics. This section summarizes these utilities.

Device Testing

ASC provides commands (*DISKIO*, *CDIO*) to inspect, test, and repair physical devices attached to an ASC. Syntax and usage details can be found in product command reference guides. Here is a sampling of these:

Subcommand	**Description**
COMPARE	Compare the contents of two devices.
FORMAT	Format (low level initialize) a device.
INQUIRY	Display device inquiry data (device characteristics).
LOAD	Load/unload compact disk (*CD*).
LOGRESET	Reset device log data.
LOGSENSE	Display device log data.
LOCK	Lock device pages into device cache.
MODEMASK	Display device mode sense mask data.
MODESENSE	Display device mode sense data.
PATTERN	Write and/or read/compare one of a number of patterns to device blocks.
PREFETCH	Prefetch device pages into device cache.
RBUFFER	Do read buffer test on device. This tests the physical connection between the ASC and the device.
READ	Do a number of device read tests, such as random, same sector, sequential reads of user definable ranges and sizes.
READDEF	Read device defect list.
REASSIGN	Reassign device blocks.
REPORTLUN	Display device logical unit information.
SELFTEST	Invoke device self test.

SENSE	Display device sense data.
START	Start device.
STOP	Stop device.
SYNCCACHE	Synchronize device cache.
TESTUNIT	Issue one or more device *test unit ready* commands.
UNLOCK	Unlock pages from device cache.
VERIFY	Do verify of device data.
WBUFFER	Do read/write buffer test on device. This tests the physical connection without altering user data on the device.

Data Verification

Three forms of destructive and nondestructive data verify and compare utilities exist in the system:

1. A device diagnostic tool (*DISKIO PATTERN*) can be used to write, and/or read and verify one of a number of data patterns to locally attached physical devices (remote physical devices can be tested by use of the same command on the remote ASC). Patterns include writing blocks of all zeros, all ones, and decrementing byte values. The block number and the ASC device identifier can be written into each block and verified to insure that the appropriate block and device are being accessed.

   This test overwrites (**destroys**) existing device data, so this command can also be used to perform a high security erase of data before a device is taken out of service (at least ten different pattern runs should be done).

2. A device diagnostic (*DISKIO COMPARE*) compares two physical devices (or compares a device to itself). This diagnostic operates on local physical devices only, and can be used as a non-destructive physical connection test.

3. A cache and mirror set diagnostic (*CCOPY*) compares two cache devices. This can be used to insure that the volumes of a copy set are at parity. As this utility compares two logical device images, the physical contents may be either local or remote. This utility can be used to test network speed and connections (by comparing a remote device to itself).

4. A data mirror map display command (*MCOPY*) displays the number and block ranges of data blocks that are not at parity between a primary and a secondary device. This command can be used to determine the status of data on any secondary device.

System Utilities

To facilitate porting the ASC software to various hardware platforms, and to ease new device driver development, the RTOS and ASC provide utilities to inspect and modify storage, I/O memory space, and I/O registers. These maintenance commands are hidden from end users and are only for internal development and debugging use.

PCI and APIC Configuration

The RTOS can display and alter PCI configuration and display multiprocessing registers. While the RTOS requires that the system BIOS assigns I/O memory, I/O registers, and IRQ values (as these are motherboard design dependent), these values can be displayed in a number of .ways, for example:

1. Display of PCI configuration data by bus/device/function (*PCI*)..

2. Display of PCI memory and register assignments (*DISPLAY, PORT*)..

3. Display of PCI IRQ assignments (*PCIINT*).

4. Display of BIOS multiprocessing information (*MPH*).

5. Display of I./O Advanced Programmable Interrupt Controller (APIC) registers (*MPI*).

6. Display of local APIC registers (*MPL*).

7. Display of IPMI data (*IPMI*).

Device registers

The RTOS provides maintenance commands to display and modify device memory and/or I/O registers (*DISPLAY, PORT*). Some RTOS device driver implementations provide additional commands to format these data, easing development, problem isolation, and debugging.

Configuration

The RTOS and ASC are configured using an ASCII text file. Each line of the configuration contains a single configuration statement. The ASC does a multi-pass scan of the file, so configuration statements can be entered in any order. Statement commands and parameters are separated by one or more blanks or tab characters.

The ASCII file format was chosen to avoid developing and maintaining host-dependent applications (having said that, work is in process to allow a user to interactively create or modify a configuration file using *web browser* graphical user interface (*GUI*) technology).

Each configuration file is saved on the ASC via an FTP file transfer. The ASC maintains a number of saved configuration file images, allowing for easy reconfiguration, configuration testing and configuration back off. Configuration changes require a system warm or cold boot to take effect.

Three types of information are contained in a configuration file:

1. Optional fundamental system definitions. These define settings used to initialize the system, for example, multiprocessing enable and type, device driver parameters, memory allocation overrides, IP definitions, initial network authorizations, user authorizations, and so forth.

2. Optional initial device mapping. These statements define the mapping between host devices, cache devices, and physical devices (be they local or remote).

3. Optional initial settings, such as copy set definitions, performance tuning parameters, and so on. One or more operator commands may also be invoked as system startup.

Here is a sampling of configuration file statements:

Statement	**Description**
CMD	Issue command when system is initialized.
CTL	Alter default controller parameters.
DEV	Alter default device parameters.
DEVNAME	Assign port, physical device, etc. alphanumeric name.
DNS	Define domain name server(s).
DOMAIN	Assign ASC domain name.
IPLINE	Define physical network connection authorizations.
IPNAME	Assign static network name.

LINE	Alter default physical network connection parameters.
MAP	Map host devices to cache and physical devices.
NAME	Assign static network name.
PASS	Assign operator privilege passwords.
QCMD	Queue command to run as a task when the system is initialized.
ROUTE	Define static TCP/IP routing information.
SER	Associate configuration file with specific ASC (by serial number).
SET	Alter default system settings.
SNMP	Define SNMP manager authorization(s).
SYSPORT	Alter default TCP/IP port defaults.
TCP/IP	Define authorized TCP/IP connections.
TIMESRVR	Define ASC timeserver.
TIMEZONE	Assign ASC time zone (offset from GMT/UTC/Zulu).
USER	Define Telnet, FTP, and/or HTTP user(s).

Operator Commands

The ASC and RTOS provide a wealth of commands to allow for control, monitoring, tuning, testing, debugging, and dynamic configuration. Many of these have been referenced in this document.

As of this writing, there are about 150 distinct commands, however the average operator and system administrator typically use a small subset of these.

Here are a few of the most frequently used commands:

Command	**Security Level**	**Description**
CACHE	Display	Display cache utilization summary.
CPU	Display	Display CPU utilization summary.
DHOST	Display	Display host device summary.
DRDEV	Display	Display real device summary.
ECOPY	Operator/Privileged	Establish or reestablish a copy pair.
HIT	Display	Display host utilization summary.
QCOPY	Display	Display copy set summary.
SCOPY	Operator/Privileged	Suspend a copy pair.
SET CMDAUTH	All	Change command security level.
SHUTDOWN	Operator/Privileged	Shutdown the system.
TCOPY	Operator/Privileged	Terminate a copy pair.
WARMBOOT	Administrator	Warm boot the system (load new configuration and/or system image).

Security Levels

The ASC has four levels of command access authorization. To access a level higher than *display* access, masked password entry is required. Each higher level authorizes all lower level commands. The levels are:

Security Level	**Description**
Display	Issue display only commands, such as the status of copy sets, I/O rates, and so on. No commands exist to alter the state of the ASC.
Operator/Privileged	Operational commands such as starting, stopping, and suspending copy pairs, bringing devices online and offline, etc.
Administrator	Administrator commands such as adjusting performance values, defining devices,

	shutting down or restarting the system, etc.
Maintenance	The full command set. Provides commands for debugging the system, running diagnostics, etc.

Console Function Keys

The following local console function keys are supported by RTOS:

Key	**Function**
Alt-F1	Lock the keyboard.
Alt-F12	Unlock the keyboard.
Down-arrow	Place a previously entered command in the input area. Move one command forward in the command retrieve stack. The last thirty previously entered commands are maintained in the command retrieve stack. Duplicate commands are not saved.
ESC	Clear the input area.
F12	Clear the screen.
Tab	Move the cursor to the beginning of the input area.
Up-arrow	Place a previously entered command in the input area. Backup one command in the command retrieve stack. The last thirty previously entered commands are maintained in the command retrieve stack. Duplicate commands are not saved.

Remote Operator Command Access

The ASC may be remotely controlled and monitored by a set of standard network protocols (described below). Any access requires a pre-configured IP source address, and user identifier, and a password. Remote commands are also subject to the four security levels described above.

Operator commands may also be directed to remote ASC's from the console of a local ASC (if authorized). Such commands are logged on the remote ASC's log, and command response messages (if any) are returned to the issuing ASC console.

Management/Network Access

The ASC and RTOS provide mechanisms for configuring, administrating, controlling and monitoring an ASC system. These are based on existing Internet standards, eliminating the need for platform-specific host software.

To maintain security, ASC configuration statements (*USER, SNMP*) authorize access to these protocols. Each user, identified by IP address (or domain name), user identifier, password, and optionally by physical network connection, must have explicit permission to use each protocol, and the levels of access within that protocol.

For example, an FTP access permission might include the ability to download but not upload files. An SNMP permission might include the ability to receive traps, but neither issue queries nor management updates..

FTP, Telnet, HTTP, and SNMP default TCP port numbers can be changed via the *SYSPORT* configuration statement. Each service may also be completely disabled.

FTP, Telnet, and HTTP users can be dynamically added and deleted, and can be forced off the system by operator command.

FTP

The File Transfer Protocol (*FTP*) is used to manipulate the following objects:

Object Type	**Description**
Configuration	Up to eight flash memory configuration files (*CONFIG.0* through *CONFIG.7*), and up to sixteen internal disk configuration files (*DISKCONFIG.0* through *DISKCONFIG.15*) may be uploaded and downloaded. The currently active configuration (*CONFIG*) may also be uploaded.
Dump	Up to four system abend dump files (*DUMP.0* through *DUMP.3*) may be uploaded for post-mortem debugging.
Files	Internal disk filesystem files may be uploaded and downloaded.
Log	Three saved log files (*LOG.0* through *LOG.2*), or the currently open log file (*LOG*) may be uploaded.
System	Up to four flash memory images of the RTOS and ASC code (*SYSTEM.0* through *SYSTEM.3*), and up to eight internal disk images (*DISKSYSTEM.0* through *DISKSYSTEM.7*) may be uploaded or downloaded.

Telnet

Telnet session(s) can be established to issue ASC commands and receive ASC responses. Both ANSI and VT-100 emulation (up to a 50 line display) are supported. Each Telnet session prompts for a userid and password. Command entry levels higher than *display* require the entry of additional passwords using the *SET CMDAUTH* command..

To preserve security, telnet sessions can only issue commands to the ASC they are logged on to. Multiple, simultaneous telnet sessions are supported.

HTTP

The RTOS contains a simplified Hypertext Transmission Protocol (*HTTP*) server. This server allows the ASC to be monitored and controlled from standard web browsers such as Netscape or Internet Explorer, thus providing the potential for a pleasant graphical user interface without developing host-specific application code.

HTTP objects are kept in a simple file system on ASC internal hard drives. These objects are uploaded and downloaded into the filesystem by FTP. Virtually any HTTP object may be stored, up to a present limit of 4,096 files, ranging in size from one byte to 4 megabytes.

SNMP

The ASC and RTOS provide Simple Network Management Protocol (*SNMP*) Version 1 support to monitor and control the system, and to report exceptional conditions.

Information on the overall system and individual system objects are available. The current list of objects and traps may be found in the management information block (*MIB*) provided with the ASC. The following shows some of the information provided:

Object	Description
Mandatory System Group	The SNMP standard MIB information, including ASC up-time, contact information, system description, physical location, and so on. Authorized SNMP managers may update some of these fields.
ASC System	Vendor specific information on the overall system, including: 1. Software version. 2. Memory size. 3. Overall cache usage information. 4. Current configuration identifier. 5. Current RTOS/ASC identifier. 6. Cache hit/miss rates. 7. Processor utilization. 8. Environmental information such as UPS and power state.
Host Devices	Vendor specific information on each host device, including: 1. Device identifier and name (if any). 2. I/O counts. 3. Hit/miss counts. 4. Prefetch statistics. 5. Disconnect/reconnect statistics.
Logical (cache) devices	Vendor specific information on each cache device image, including: 1. Device identifier and name (if any). 2. Copy set state. 3. Primary device identifier (if a secondary). 4. Block count and block size. 5. Number of secondaries (if a primary). 6. Copy establish progress (if establishing).
Real (physical) devices	Vendor specific information on each physical device, including: 1. Device identifier and name (if any). 2. Device type. 3. Device status (online, offline, failed, etc.). 4. IP and MAC address (if a network port). 5. Bus type (if SCSI or Fibre Channel). 6. I/O operations per second. 7. Bytes transferred per second. 8. Network messages sent and received. 9. Interrupt count. 10. Device block count, size, and volume serial number. 11. Maximum simultaneous active I/O operations. 12. Error statistics (missing interrupts, timeouts, aborts, resets, etc.). 13. Last I/O error information..
Remote Systems	Vendor specific information on each connected ASC, including: 1. IP address. 2. MAC address of physical connection. 3. Domain name. 4. Physical connection type. 5. Connection status. 6. Messages sent and received. 7. Message timeouts.

The following are some of the SNMP traps that may be generated:

1. System boot.
2. Power failure.
3. Power restore.
4. UPS battery low.
5. Copy pair establish start.

6. Copy pair establish complete.
7. Copy pair suspend.
8. Network connection failure.
9. Network connection restore.
10. Device failure.
11. Device offline.
12. Device format start.
13. Device format complete.
14. Environmental conditions.

The following system management functions are provided via SNMP put requests:

1. Initial establish copy to create a copy set.
2. Suspend the primary of a copy set.
3. Suspend the secondary of a copy set.
4. Re-establish a suspended copy set.
5. Change the default active system image.
6. Change the default active configuration file.
7. Reboot the ASC.
8. Reset ASC statistics.

DHCP

Dynamic Host Configuration Protocol (*DHCP*) allows an ASC to request one or more IP addresses from a DHCP server. The ASC transmits the ASC domain name to the DHCP server, so that it may be subsequently identified by other ASC and host systems.

DNS

ASC may be configured to act as a Domain Name Service (*DNS*) client, requesting domain name to IP address resolution. This may be used in environments where IP addresses are dynamically (rather than statically) defined.

The ASC can also be configured with a static name to IP address table by use of the *IPNAME* configuration statement..

ASC Performance and Tuning

This section describes ASC performance considerations, and describes some of the tools for optimizing overall and specific ASC performance.

Central Processing Units (CPU's)

The ASC supports multiprocessing, so it is possible to have multiple processors working concurrently. This has three benefits:

1. Multiple processors increase the amount of available processing power. The obvious implication of this is that CPU intensive activity can be spread among operating processors. The largest single benefit is in paralleling TCP/IP checksum generation and checking.

2. Multiple processor provide the ability to simultaneously process multiple host requests.

   The SCSI or Fibre Channel interface hardware and ASC driver handle the initial portion of each host

request (see *Drivers* below). If multiple CPU's are running, it is possible to process multiple host requests simultaneously, thus dramatically shortening each host's request-response time

3. Multiprocessing, combined with ASC multitasking, enhance the system's ability to destage data while processing subsequent host requests.

4. Multiprocessing support allows working processors to handle the work of failing processors.

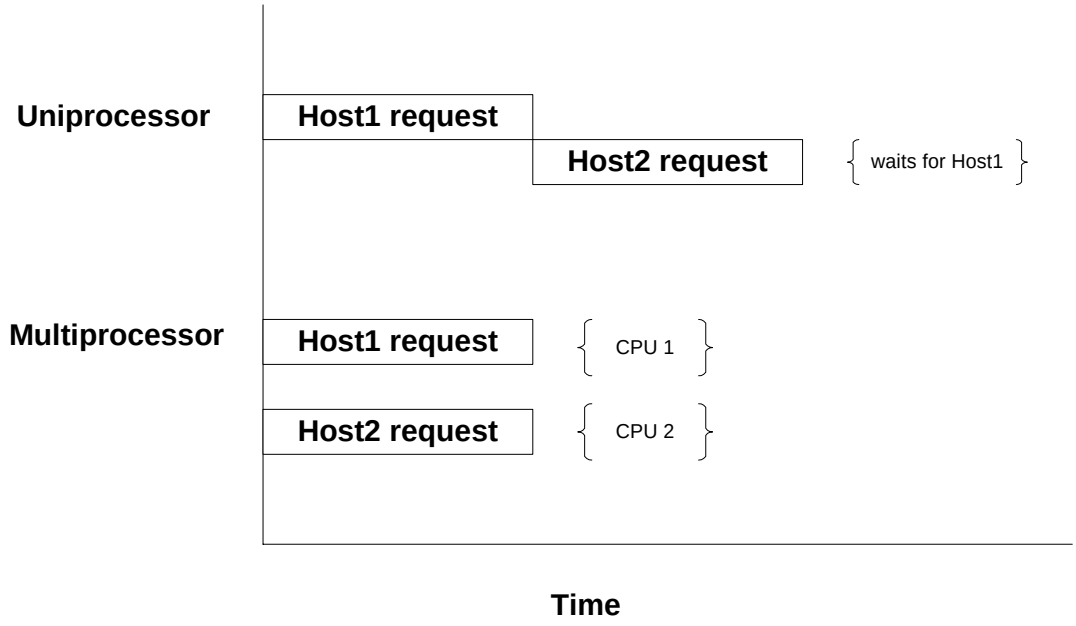

Figure 19: Uniprocessor versus Multiprocessor Host request timeline

Memory

Main and cache memory should be as fast as possible, and should have fast and wide, independent paths to all I/O (usually *PCI*) busses. ASC device data transfer is much more prevalent than processor-to-memory data transfer.

More memory is always better than less, since more data may be cached, resulting in a higher hit rate.

Data Busses

The number, speed, and configuration of internal data busses directly impacts overall system performance.

Fast devices should be kept on fast busses, slower devices on slower busses because slow devices take bus time away from fast devices.

Host interfaces should be placed on faster busses than device interfaces. Assuming that the system has a reasonable hit rate, there will be more host I/O activity than device activity.

Design or choose hardware platforms that have multiple busses with independent paths to memory instead of designs that have *cascaded* or *bridged* busses – while this design adds device connectivity, the overall bandwidth is only that of a single bus.

If a host and/or device has multiple I/O ports, these should be connected to different physical ports on different busses to reduce *single point of failure* issues, enhance performance, and allow port or path failure recovery.

I/O Interrupt Queues (IRQ's)

Controllers should be plugged into PCI bus slots which route their hardware interrupts to unique hardware interrupt queues (IRQ's).

It is especially important to separate host/device and network controller IRQ's, as network interfaces generate many more interrupts than hosts or data devices. Because of the small packet size in networking a loaded system may receive three orders of magnitude more network related interrupts than device and host interrupts combined.

Every time an I/O interrupt occurs, each device driver sharing an IRQ must be called since a single interrupt may be pending on any or all of the cascaded devices. This causes an unnecessary waste of CPU and I/O resource as each driver associated with the IRQ level is called and the driver must poll its hardware to see if an interrupt is actually pending.

Statistics

The ASC maintains a large number of statistics. These can be inspected to monitor operations and gather data for tuning performance.

Statistics can be displayed on operator command, or can be retrieved by SNMP queries (see *SNMP* above). Statistics accumulate over time. Statistics can be reset by type, so that a specific period of activity can be monitored.

Here are some of the available statistics, sorted by type:

Type	**Units**	**Description**
Processor		
CPU utilization	Percentage	Idle, system, task, and I/O driver CPU time.
Interrupt counts	Counter	Number of hardware interrupts by type, host interrupt count, and device interrupt count.
Multiprocessing	Counter	CPU wakeup count, CPU sleep count.
Task	Counter	Task dispatches, task preemption, dispatcher calls.
Host Activity		
Destage	Counter	Destage write merges by host device and overall system..
Disconnects	Time	Average, minimum, and maximum I/O disconnect/reconnect time.
Hit/partial/miss	Counter	Hit, partial hit, miss counts by overall system and by host device.
I/O type	Counter	Number of host reads, asynchronous writes, synchronous writes, synchronize cache overall and by host device.
Physical I/O	Counter	Physical I/O read, write, and remote I/O requests issued as a consequence of host activity by host device.
Prefetch	Counter	Prefetches, prefetch merges, unnecessary prefetches, late prefetches overall and by host device.
Requests	Counter	Number of host requests by interface.
Size	Blocks	Average, minimum, and maximum I/O request size by device.
Cache		
Allocations	Counter	Total blocks, currently allocated blocks, and maximum allocated blocks by size and type.
Status	Counter	Modified and/or pinned cache blocks overall and by cache

		device, locked blocks by cache device, allocated blocks by cache device.
Physical Device		
I/O counts	Counter	I/O operations, timeouts, errors, aborts, resets, etc.
I/O performance	Time	Bytes transferred per second, I/O operations per second, interrupts serviced per second
Status	Miscellaneous	Device type, status, serial number, block count, block size, last block accessed, etc.
Remote		
Connection	Miscellaneous	MAC address, physical connection ID, messages sent, messages received, minimum, maximum, and average round trip times.
TCP/IP	Miscellaneous	TCP/IP messages sent and received, average and maximum round trip times, acknowledgements sent and received, socket state, out of order packets, SACK's sent, etc.
Network		
Network users	Miscellaneous	Currently connected Telnet, FTP, and HTTP users.
Physical connection	Miscellaneous	Messages sent and received, connection state, routing information, local and remote MAC addresses, bytes transferred per second, interrupts serviced per second.
Physical layer	Counter	Ethernet collisions, bad CRC, short messages, pause packets

Logs

The ASC maintains a time-stamped log file containing all exceptional conditions and error conditions on internal disk drives. Optionally, operator command input and output can also be logged.

Three four-megabyte (4MB) log files are kept. When one file fills, it is closed, and the next one is opened. When all three are full, the first is overwritten, and so forth.

Log entries are ASCII text messages, with a carriage return/line feed suffix. Exception conditions have a timestamp prefix (*HH:MM:SS*). At midnight, a new day message is logged. A single message may spill from one log to another.

Either the currently open log (*LOG*) or any of the log files (*LOG.0* through *LOG.2*) can be retrieved via FTP.

It is possible to attach an ASCII printer to the ASC's parallel port and log messages and operator activity to paper. This may be useful for debugging, but is otherwise not recommended as it slows down the ASC.

For security, all Telnet and remote operator command requests are logged.

Certain non-critical operator messages and log entries can be disabled, for example, remote message timeouts, ICMP messages, ping requests, etc.

Most exceptions may optionally be sent to configured SNMP manager(s) via SNMP trap messages (see *SNMP* above).

Here are a sample of logged events:

Logged Event	**Description**
Host	
Aborts, resets…	Host issued bus reset, abort, abort tag, reset device, etc.
Unknown requests	Host attempted an unsupported command.
Cache	

Copy Pair Establish	Copy establish start and completion.
Copy Pair Suspend	Copy pair suspend.
Copy Pair Terminate	Copy pair terminated.
Pinned data	Logs the first occurrence of pinned data on a cache device. Also logs when all pinned data are written to a device and cleared.
Physical Device	
Format	Device format start.
I/O errors	Device check conditions including sense data, missing interrupts, bus resets, etc.
Network	
Connections	Established, closed, and aborted TCP/IP connections.
ICMP	Received ICMP exception messages.
Timeouts	Timed out and/or lost messages.
Users	Telnet, FTP, and HTTP user activity.
Security	
ICMP echoes	Unauthorized pings.
Operator commands	Local, remote, and Telnet operator commands.
Users	Unauthorized Telnet, FTP, HTTP, and SNMP access attempts.

Tuning Parameters

There are many parameters for optimizing ASC performance. Many of these can be altered in real time -- thus it is possible to monitor performance and alter settings dynamically. Settings may also be adjusted based on the anticipated requirements of a given workload.

Here are some of the ASC settings:

Setting	Description
Processor	
Dispatch time slice	Alters the amount of time a given task can run before being preempted by another task of equal priority.
Host	
Concurrent destages	Defines the number of simultaneous physical device writes that can be concurrently active.
Concurrent host writes	Defines the number of asynchronous host writes that can be concurrently active.
I/O size	Set maximum size of host data transfer without a bus disconnect.
Synchronous Mode	Determines whether all host writes are treated as synchronous writes. This may be set globally, or by individual host device.
Throttle	Defines conditions when the ASC will "throttle" host activity. These include: 1. Amount of cache holding modified user data. 2. Free storage available. 3. TCP/IP backlog. 4. Active tasks running.
Cache	
Concurrent activity	Set the maximum concurrent I/O allowed per device and/or controller.
Copy sets	Suspend on destage backlog, manually suspend based on workload, etc.
MIH times	Set missing interrupt times by device type.
Physical Device	
Pinned data	Stop host writes if pinned data count exceeds a threshold.
Network	
ARP's and Pings	Set amount of time between ARP and Ping message transmission and receipt timeout.
Inter-message delay	Set delay between messages.
Inter-packet gap	Set gap size between packets.

Message size	Set maximum logical message size.
Timeouts	Set timeout time for various messages.

Tuning Tips

Tuning a storage subsystem is a complicated, continuous, evolving, installation-dependent effort. As it would take a paper bigger than this one to even scratch the surface of this topic, here are a few general tips in no particular order:

1. Performance is inversely proportional to integrity. This means that the more you integrity you require, the slower the performance is likely to be.

   As a simplistic example, a copy set requires double the write I/O activity of a simplex device.

   Synchronous writes guarantee that the host is not notified until data are physically written to devices – however this increases the end-to-end time of each host write.

2. Performance can be improved by **not** mirroring non-essential data, such as temporary data or system paging space.

3. Limiting host synchronous writes to critical data such as database recovery logs can enhance remote and local performance.

4. Asynchronous write performance can be improved by enabling physical device write caching if you are **positive** that the physical device **cannot** lose cache data (*DEVICE* configuration statement)

5. Asynchronous write performance can be improved on some back end devices and RAID controllers by increasing the number of simultaneous destages (*SET MAXACTD*).

6. Certain devices and RAID controllers are able to support more simultaneous device activity than others. Performance may be improved by adjusting back-end controller simultaneous I/O and device tagged queuing levels (*SET MAXTAGS).*

7. Configured but nonexistent physical devices should be set offline (*OFFLINE* command). Otherwise, the ASC periodically polls them (about once every five seconds). On some busses, this can introduce a large delay (as the bus times out), greatly impacting performance of all devices attached to that bus.

8. Larger cache sizes are better as they increase the chance of host hits.

9. Faster, reliable communication lines, with low latency, and little unrelated activity improve remote performance.

10. Multiple connections enhance performance (as activity spreads across the connections), and enhance integrity (as fail-over paths).

11. ASC hardware should be configured to separate host, device, and network activity busses and interrupt levels.

12. As the ASC minimizes inter-CPU synchronization, multiple slower CPU's are often better than fewer faster CPU's.

13. CPU's with larger on-board caches tend to decrease the system bus and memory bandwidth requirements.

14. Full device copy establish should be performed during off peak times whenever possible. The number of simultaneous copy establish tasks should be minimized.

15. Copy pairs may be suspended and reestablished if redundancy isn't needed during an intensive host write application (such as a restore). Many times, a reestablish requires less I/O then real-time host mirroring as host applications (especially databases) typically write the same blocks many times.

16. It may be more efficient to establish multiple copy pairs one or a few at a time.

17. Cascaded mirrors establish more efficiently if they are established sequentially outward. It is possible to concurrently establish a cascaded mirror, but it tends to increase physical device head movement, and may result in unnecessary copying (if the cascaded copy runs faster than the cascading copy).

18. Prefetch values may improve or adversely effect performance based on workload. Sometimes (such as during a UNIX filesystem build) it is better to temporarily disable prefetch.

Typical Uses

This section illustrates a number of uses for the ASC. As with any new technology, additional applications are continually being created and put to productive real-world use. Here are a number of these that are in use today. Many of these applications can be combined in an ASC environment.

Storage Area Networking

Storage Area Networking (*SAN*) attempts to separate storage and processing resources, giving users the flexibility to pool, allocate, share, manage, and easily distribute storage amongst divergent interconnected resources.

While SAN doesn't **require** any specific protocol, Fibre Channel has become the *de facto* protocol of choice for implementing SAN's. Since many storage subsystems today use some form of parallel SCSI protocol, it is assumed that any SAN implementation requires that SCSI hardware be replaced or upgraded, or at least *bridged* to be part of a SAN.

A significant challenge to implementing any new technology is planning how to integrate (or at least non-disruptively replace) existing hardware and software, minimizing cost, errors, and downtime.

The ASC technology is well suited for a migration to and use of SAN.

SAN Bridging

Bridging allows you to seamlessly and transparently interface different protocols together. In SAN networks, bridging refers to SCSI to Fibre Channel conversion, and Fibre Channel to SCSI conversion.

Ideally, a bridge is invisible, or at least transparent. Practically speaking, this is seldom the case, as different protocols run at different speeds, may have different standards, and have different error reporting and recovery issues.

Since the ASC architecture emulates devices and is independent of the physical layer protocols (which are handled by ASC device drivers), the ASC is a natural bridge.

ASC caching enhances this bridging function by potentially eliminating many back end transactions as all read hits are processed from the cache.

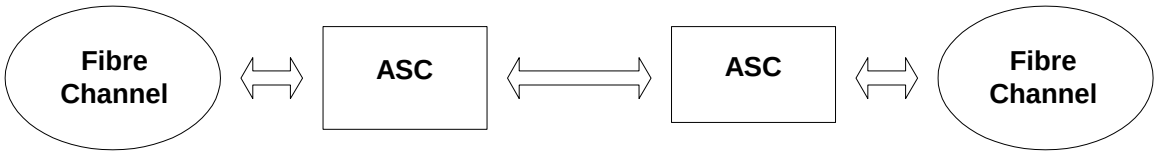

Figure 20: Fibre Channel Bridge

SCSI to Fibre Channel

An ASC can map any host interface to any physical device. Consequently, an ASC can be used as a *SCSI to Fibre bridge*. SCSI to Fibre Channel bridging is typically used to extend the life of existing servers by allowing them to access Fibre Channel devices and/or SAN networks. Bridging also allows you to use lower cost SCSI interface cards in low end servers.

The ASC provides local caching, allowing for potential performance improvements through read hits.

Since the ASC emulates devices, it can only bridge supported device types, presently disks and CD ROM's. (Tape support is under study).

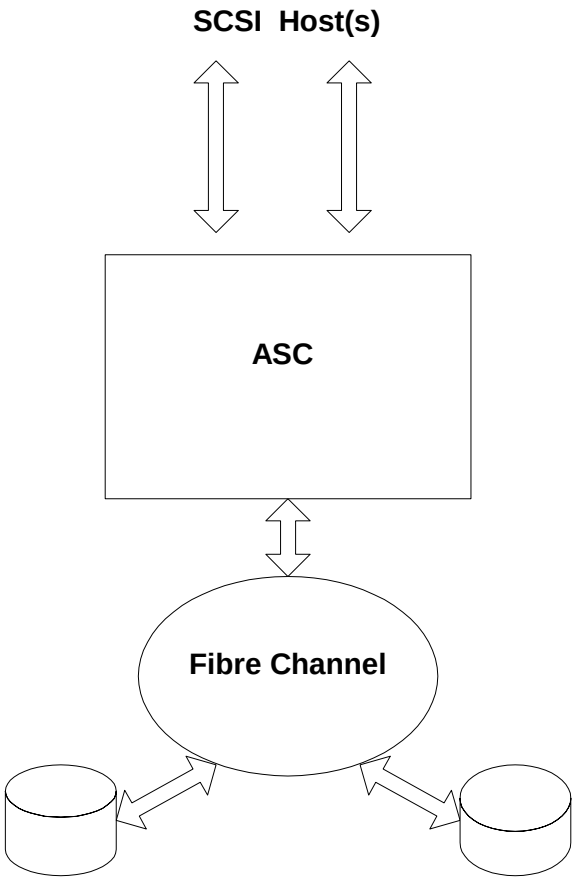

Figure 21: SCSI to Fibre Channel Bridge

Fibre Channel to SCSI

An ASC can bridge from Fibre Channel to SCSI. This can be used to allow new servers or SAN networks to access older legacy equipment.

ASC caching potentially provides improved performance of slower devices by taking advantage of read hit transfers from the ASC cache.

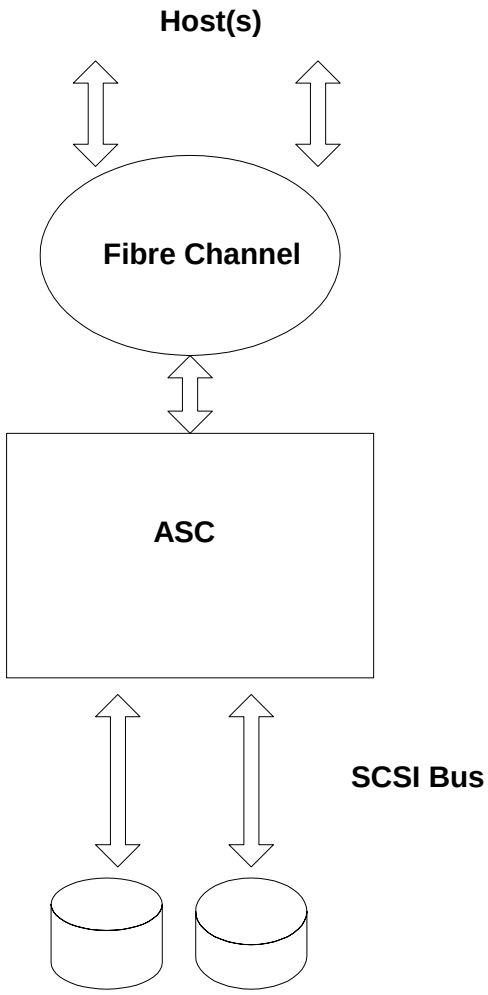

Figure 22: Fibre Channel to SCSI Bridge

SCSI/Fibre Channel Split/Merge

ASC mapping allows you to map any host device to any physical device. This provides the following benefits:

- Several hosts can access the same device without requiring costly multiple-path device hardware.
- A host can have multiple hardware paths to an ASC, resulting in improved performance and redundancy.
- A host with a fixed number of SCSI or Fibre Channel connections can access more devices (by mapping many individual devices as logical units of a single device on the same host bus).
- ASC caching provides an improved hit rate across multiple hosts and multiple devices.

Note that mixing hosts and devices with different performance capabilities may result in faster hosts throttling slower hosts' data transactions.

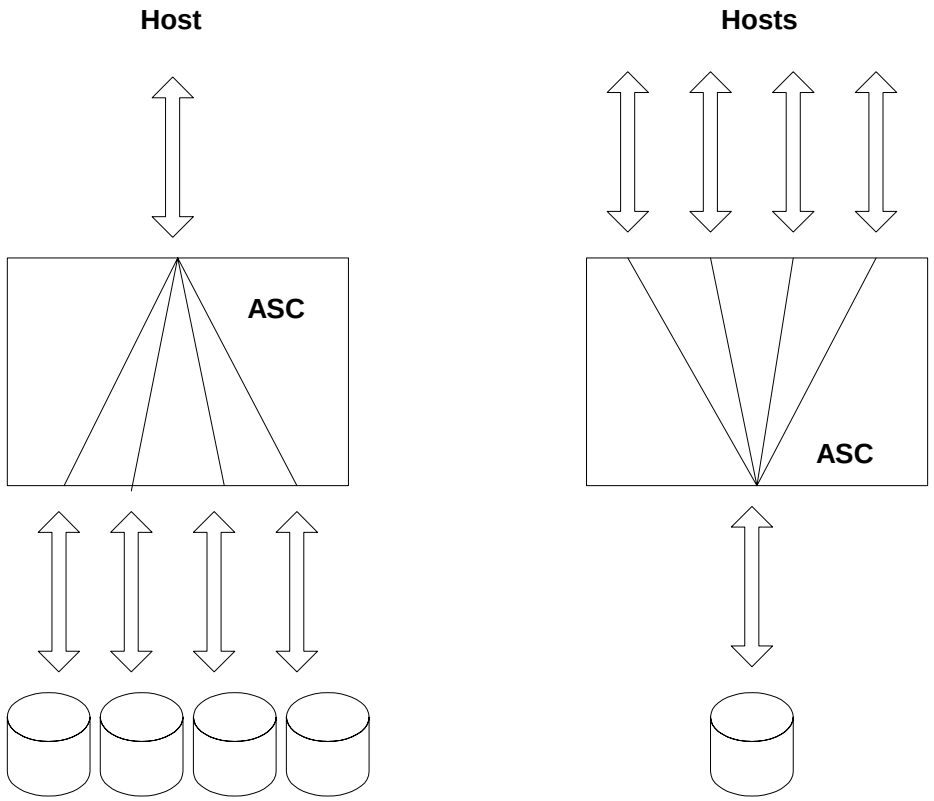

Figure 23: SCSI to Fibre Channel Split/Merge

Data/Technology Migration

There is often a need to migrate data from one device to another, particularly when changing technologies. For example:

- To move from SCSI to Fibre Channel.
- To move from a smaller, slower, less reliable device to a larger, faster, more reliable device.
- To replace an intermittent or failing device).

An ASC can be used to perform this data migration virtually no down time. Here is a sample procedure for accomplishing this:

1. Attach both the old drive(s) and the new drive(s) to the back end of an ASC.
2. Attach the front end of the ASC to the host(s) normally accessing the drives. These hosts may have SCSI connections and/or Fibre Channel connections.
3. Test the new drives using ASC diagnostic commands (*DISKIO*) to ensure that they are properly installed, perform acceptably, and are stable. This may be done while the host(s) are accessing the old drive(s).
4. Use ASC copy set facilities (*ECOPY*) to copy and mirror data from the old drives to the new drives. Leave the copies running until you are satisfied that the new drives are operating properly.
5. Terminate the copy and remove the old drives.

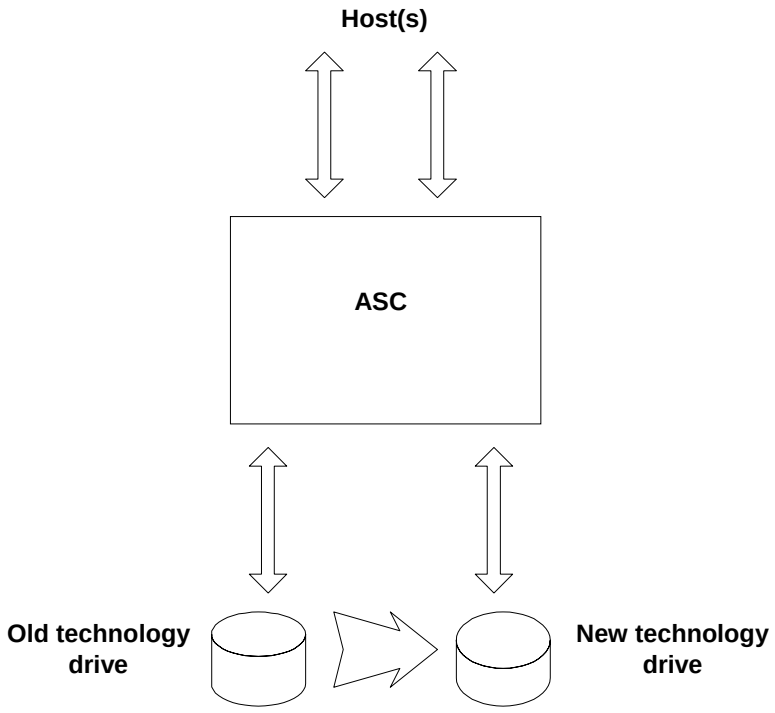

Figure 24: Technology Migration

Server Migration

It is also possible to migrate from a SCSI-based server to a Fibre Channel server by simply swapping out (or adding) the ASC front end interface cards (or by moving from an installed SCSI to an installed Fibre Channel interface).

This allows you to change servers without changing data storage devices. This may also be used to test new server interfaces on existing servers (server upgrades).

For the duration of the server burn-in period, both the old server and new server can be connected to one ASC, so that you can fall back to the old server if necessary.

Device Testing

The ASC provides a large number of device testing commands (see *Tools/Device Testing* above). An independent testing and validation ASC can be used to burn in newly acquired devices before placing them into production.

ASC's can be used to aid in device manufacturing testing or as part of a device acceptance package.

Device Performance Analysis

ASC device testing commands and ASC statistics can be used to analyze the **true** performance of devices under test.

While performance statistics are often cited, they are seldom specific enough to allow for accurate device comparisons. For example, the value of *I/O operations per second* seldom includes the *type* of I/O operation, the *number of simultaneous operations*, the *number of devices* or *paths used*, the *data size* (if any) of the transfer (if any) etc.

ASC testing commands can be used to perform virtually any sort of I/O by quantity, type, size, path, and so on, and return specific I/O and byte transfer rates for each device or for the overall system.

Unlike tests that run on general purpose servers, since the ASC is based on a dedicated RTOS, there is virtually no ASC/RTOS overhead associated with this testing.

SAN Routing

A major advantage of Storage Area Networking is the ability to connect many hosts to any in a pool of devices. A serious challenge in implementing SAN is preventing unauthorized hosts (or outsiders) from accidentally or willfully accessing or changing data.

Zoning

The ASC allows you to map each host, host connection, device identifier, and logical unit to a specific cache device (and, implicitly, to one or more physical devices). This obviates the need to have physical SAN sub-networks for the sole purpose of isolating data from unauthorized access and creates a SAN *firewall*

ASC mapping authorizes the following forms of access by individual host and host port:

1. Read/write access.

2. Read/only access.

3. Write access to device block zero.

4. Diagnostic *pass-through* to a device.

5. Ability to format a device.

6. Ability to issue device-dependent commands to a device.

Global Caching

The ASC cache provides faster read access to often-accessed data. This cache functions across all configured cache devices, providing a *global cache*. Unlike local device caches, which are dedicated to a specific device and cannot be used to cache other device data, global caching allows an entire set of devices to share a large cache, insuring that the most relevant data are cached.

The ASC lets hosts or a system administrator *lock* certain ranges of data into cache, prioritizing access across multiple hosts or applications.

Disk and Compact Disk data are cached.

The ASC provides anticipatory prefetch algorithms to stage data in anticipation of subsequent host requests.

The ASC supports the SCSI *Prefetch* and *Lock/Unlock Cache* host commands.

Memory Disks

Portions of ASC cache can be used to emulate volatile or non-volatile disk drives with no physical device counterpart.

Volatile memory disks lose their contents on a system shutdown or restart. *Non-volatile* memory disks are saved to internal disk storage and are preserved across system shutdowns and restarts.

These drives can be used to house temporary files, temporary database or search engine indexes, system paging space, or as a means for communication and synchronization between hosts and/or applications.

Memory disks can be used in conjunction with all other ASC devices.

Disaster Recovery/Business Continuation

ASC technology is a significant part in the implementation of disaster recovery and business continuation plans.

Disaster recovery is the ability to recover data following an unanticipated disaster (for example, your building burns down, or California finally falls into the ocean).

Business continuation is the ability to quickly and inexpensively return to normal operations following any disaster or unanticipated outage of equipment, power, facility, etc.

Generally, disaster recovery plans focus on the **availability** (versus irretrievable **loss**) of data, while business continuation plans focus on minimizing the **effect** of an outage on your operation. Lack of disaster recovery could be fatal, lack of business continuation is costly (and possibly fatal). *Disasters*

seldom do, but can occur. *Outages* are much more frequent. Disaster recovery is often addressed by the creation and maintenance of (seldom tested) offsite backups. Reduction of outages (and, indirectly, disasters) is minimized by computing redundancy and facility distance.

Remote facilities are ideally situated on different power grids, different earthquake zones, have rugged construction, use different communications vendors, and so on. Remote facilities also tend to be untended, and usually have higher security than local facilities.

This section describes some of the ways an ASC can aid in data protection and business continuation..

Device Protection

When an ASC mirrors data, each device is protected by another copy (or copies) of identical data. If a device block cannot be read, it is read (and the original possibly updated) from another copy. If a device fails, one of the secondary devices is used instead.

There are many excellent descriptions of RAID and RAID levels, so I will not endeavor to describe them here. It is accepted that RAID disk *mirroring* (*n* copies of data) has better performance and is more reliable than any parity-based striping RAID implementation (at the cost of the number of disks required for the copies).

Since disk devices have continued to decrease in cost, the performance overhead required to implement RAID no longer justifies the one-time cost benefit (less disks).

If parity/striping RAID *is* required (or is already available) in your installation, RAID units can be attached to the back end of an ASC, resulting in the protection of both mirroring and RAID.

Server Protection

The ASC provides multiple host connections, so it is possible to leverage ASC technology to provide server fail-over. When a server fails (or is brought offline for maintenance), another server, connected to the same ASC's, may take over the failing server's workload. Then, when the server is repaired, the work may be moved back.

The ASC only provides multiple host paths to facilitate server protection. The ASC does not take any active part in the server fail-over procedure.

SNMP management may be used to reconfigure the ASC's for a backup mode, then return to normal mode when the outage is corrected.

Facility Protection

Since ASC's can mirror data at any distance through the use of TCP/IP networking, it is possible to protect entire installations. Here is an example of how this is done:

1. Assume that the ASC at location *A* mirrors to the ASC at location *B*. Location *A* is processing applications. Location *B* may or may not be processing independent applications.

2. When location *A* fails, the ASC at location *B* is reconfigured to process data at location *B*. At this point, location *B* is running location *A*'s applications.

   Depending upon whether the ASC at location *A* is still accessible, data written at *B* **may** be *reverse mirrored* back to *A*. Depending on resources and priorities, location B may also continue processing its own applications.

3. When *A* recovers, data changed at *B* is copied **back** to *A* (assuming that it wasn't already being copied during the failure).

4. Once the data are again at parity, *B* is reconfigured to its original configuration, and location *A* resumes production.

It is possible to have both *A* act as a backup to *B* and *B* act as a backup to *A*.

We find that ASC users are roughly split between wanting to automate this procedure, and **not** wanting to automate the procedure.

Those favoring automation argue that no human intervention is required, shortening recovery time.

Those opposing automation argue that the damage potentially caused by a *false positive* failure-detection outweighs the benefit of automation.

Given the myriad number of potential failures, it seems better to have at least some human authorization involved in any automated process.

Remote Data Access

The ASC enables transparent host access of remote devices. This means that any ASC attached device may be accessed at any distance. The host device's performance is, of course, based on the quality and speed of the network connection and the cache hit rate of the application.

A remote device may simply be a mirror of local data, or it may be directly accessed. **Many** remote devices may be accessed, at different locations (assuming there is an ASC at each location connected to the local ASC).

Here is a sampling of some of the applications that be realized with remote devices:

1. Data Haven – centrally locating data accessed by remote hosts.

2. Clean Room – data volumes kept in a clean environment while servers run in a hostile, unprotected, or dirty environment.

3. High Security – where physical access to data media needs to be restricted.

4. Data Sharing – multiple remote hosts share centralized data, such as CD ROM data or a centralized database.

5. Remote Backup – backups are performed at the remote location, from the remote volumes, centralizing the backup process.

6. Data Warehousing – mining data at a central site.

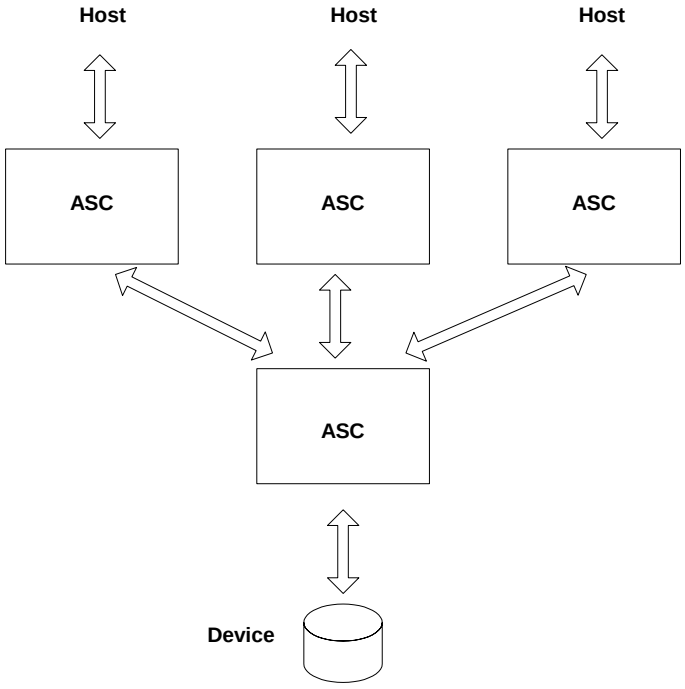

Figure 25: Remote Data Access

Remote Mirroring

Coupling the ASC remote device support with ASC *n*-way mirroring provides the ability to do remote mirroring of local data.

This facilitates the implementation of hot backup sites, where, if a local facility fails or has a disaster, the remote site can continue operations with a minimum of business disruption.

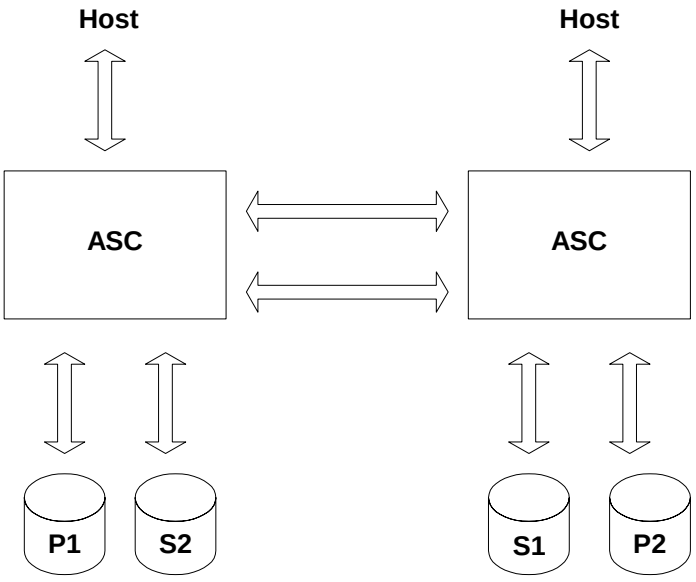

Figure 26: Remote Mirroring

Data Center Consolidation

With dynamic use of remote mirroring, it is possible to non-disruptively migrate data from one facility to another. This lets you relocate, consolidate, or split data centers.

Data Sharing and Clustering

The ASC allows virtually any number of hosts to access a logical volume. This enables one to potentially create cross-system applications that share data.

Cross-system file systems and some databases have the ability to share device data using multiple physical device interfaces.

Data sharing also lets you to create server clusters without the hardware failure risks that could result from sharing busses amongst them.

Content Distribution

ASC remote mirroring allows for server and application independent content distribution:

1. Data to be distributed are written to *n*-way mirrored volumes.

2. The remote ASC's are fed these changes automatically via ASC *n*-way data mirroring.

3. The remote host(s) read the updated data.

Internal Operations

This section gives a very brief overview of the internal logic of the ASC. These data are provided to aid in the understanding of the ASC architecture. This information is by no means an exhaustive internals or logic manual.

ASC Objects

This section summarizes some of the objects most frequently used by the ASC application. The embodiment of each object is a *C* data structure whose structure name is an acronym of the object name. The structure name is provided in parenthesis following the first use of the object name. To further prove that the author is as old as the hills, objects are sometimes called *control blocks* or *blocks*.

I/O Request Block (IOB)

One I/O request block (*IOB*) exists for each physical I/O operation. This object contains all of the information necessary to perform an I/O operation, such as the operation type, length, starting block, and storage addresses of I/O buffers. Physical back end I/O operations to disks, compact disks, network lines, and host data transfers, are executed under control of an IOB.

A single host I/O operation may have multiple IOB's associated with it, as portions of the operation may be operating simultaneously.

The IOB is an RTOS object, however, many fields have been added to facilitate the ASC application (all of these are associated with host or *target mode* emulation).

Host Task List Entry (HTLE)

Each host task list entry (*HTLE*) represents a single active host request. An HTLE exists from the start of a host operation until ending status is presented to the host.

HTLE's are used to enforce SCSI tagged queuing rules, and are used to delay host activity if the ASC does not have sufficient resources to begin the host request.

Hash Block (HASHB)

Each in-storage cached data page is efficiently located by *hashing*, using the host device identifier and the device block number (rounded to a cache page boundary).

The cache page is located by scanning the appropriate hash block queue. The hash queue is reordered to keep frequently accessed blocks toward the front of the queue. The HASHB contains the device identifier, block number, and a pointer to the cache page.

Data Block Descriptor (DBDB)

One data block descriptor (*DBDB*) exists for each potentially usable 4K byte cache page in storage. The DBDB contains the use of the cache page (cache data, allocated storage, free storage), a pointer to the cache page, the cache device identifier and block number, the data present, data modified, data pinned, and write in process masks, multiprocessor locks, and statistical information. Unused and cache pages containing user data are hardware protected from inadvertent ASC software access.

Multiple queues of DBDB's are maintained for hash lists, the LRU queue, for unused blocks, and for each individual cache device.

Real Device (RDEV)

A real device block (*RDEV*) describes each physical controller and device. An RDEV describes either a controller-device unit (such as an Ethernet controller), a multi-device controller (such as a SCSI or Fibre Channel controller), or a device (such as a disk or CD ROM). Each device that is physically connected to a given SCSI or Fibre Channel bus has an RDEV that points to its controller RDEV. Remote devices are represented by RDEV's on the local ASC.

If a physical device has several connections (such as a multi-port RAID controller), the *alternate* path device RDEV's are linked to the *primary-path* RDEV.

Each RDEV contains a queue of *active* IOB's (those with I/O in process), and a queue of *waiting* IOB's (that are ready, but are not able to run due to resource limitations). I/O operations are generally issued in the order received (FIFO), although in some cases, the IOB queue will stall while high priority I/O operations execute (for example, for error recovery or device re-initialization).

The RDEV contains counters, statistical information, device characteristics, and state information

Cache Device (LDEV)

A logical device block (*LDEV*) describes each cache image device. A cache device may exactly map to a physical device, in which case, we refer to it as a *virtual equals real* (or *V=R*) cache device, in which case the LDEV contains an RDEV pointer.

Cache devices that map to multiple real devices use a tree structure to map a given block range to the appropriate real device. Cache device RDEV's that *stripe* across a fixed number of devices point to a pseudo-RDEV that describes the striping algorithm and device set.

ASC copy sets **always** mirror cache devices. Each LDEV in a copy set points to an additional object, called an *L2RBASE* that identifies the copy state, points to the primary LDEV, and points to the copy set maps.

LDEV's contain counters, statistical information, and a queue of their in-storage cache blocks

Host Device (HDEV)

A host device block (*HDEV*) describes each host device. In almost all cases, an HDEV directly maps to a cache device, or an LDEV. It is architecturally possible to map a host device to portions of several cache devices (for example to restrict a host to a portion of a cache device such as a partition).

As the host device is the entity potentially shared by multiple hosts, all active host requests accessing this host device (IOB's and HTLE's) are queued from the HDEV.

Host Mapping

This section describes the mapping between a host to host device nexus and its objects.

Target Lookup Table

An RDEV describes each ASC front-end physical host connection (port). Within this RDEV is a pointer to an array of pointers to emulated target devices. An emulated target is found by indexing into this table.

Logical Unit Lookup Table

Each emulated target device (see above) contains a pointer to an array of pointers to emulated logical units. Each logical unit is described by a logical unit block (*LUNINFO* – see below). A given logical unit (associated with a given target device) is found by indexing into this table.

Logical Unit Block (LUNINFO)

A LUNINFO block describes each logical unit. The LUNINFO, in turn, points to a (possibly) shared HDEV. One LUNINFO exists for each host-host port-target-logical unit nexus. This block contains the host's device access authorizations (for example, read/only versus read/write, device format, diagnostic pass through, and so on).

Initiator Block (INITINFO)

An initiator information block (*INITINFO*) describes each host initiator referencing a logical unit. This object is created the first time a host accesses the ASC from a given host port and host initiator identifier. This object is located by indexing into an array of INITINFO pointers in the logical unit block.

The INITINFO object contains pending SCSI unit attention and sense data, linked command status, SCSI set limit data, sequential access monitoring data, various statistics and counters, and emulated SCSI mode page data.

Abort Request Blocks (ABORTREQ)

Abort request blocks (*ABORTREQ*) describe host abort requests. This object is created by the host interface driver and is passed on to the ASC for processing. The ASC determines which I/O operations (if any) need to be aborted based on the request. The ASC marks associated IOB's as abort pending, then calls the driver to selectively remove any related requests that may be active within the driver or hardware.

Tasks and Drivers

This section introduces ASC-specific tasks and drivers. More information on RTOS programming and multitasking can be found in the *Design of a Reduced Function Operating System.*

Kernel Code

Virtually all ASC application code, including the TCP/IP protocol stack, execute as *user mode* tasks. Certain ASC-specific functions are implemented as kernel (*system mode*) code for either serialization or performance.

Here are some of the kernel functions added to the RTOS for the ASC application (for information on RTOS-generic functions, please see the *Design of a Reduced Function Operating System*):

Function	**Description**
Cache block inquiry	Serialization of cache data lookup.
Cache data bit manipulation	These routines serialize the setting and clearing of data present, modified, write in process, and pinned masks.
Cache lock/unlock	These routines serialize the exclusive and shared cache block locks.
Copy set maps	These routines serialize the setting and clearing of copy set changed data maps.
Copy set status	The routines serialize of copy set status changes.
Host I/O reconnect	This routine is a special case of an RTOS I/O start -- it causes an eventual host reconnect, potential data transfer, potential disconnect, and/or potential ending status (versus a device I/O).
Host Abort	Given abort/reset information, serially locates and marks outstanding host I/O activity for cancellation.
Host Request	This routine, called by host drivers, determines whether a host request is valid and whether it can be immediately handled (via a hit) or must be postponed (via a miss).
Storage allocation	These routines serialize the allocation and deallocation of cache blocks, data blocks, and all common system objects.

Permanent Tasks

Permanent tasks are tasks that are typically created at system initialization and persist as long as the system is running or until their function completes. These tasks are used to monitor the system, perform periodic work, or serialize resources.

Some tasks are created to handle overall system functions (like updating the date and time on the console). Other instances of tasks are created one for each resource (for example, to manage the staging of data for a particular cache device).

Each task has an alphanumeric name describing the function of the task. If a task is associated with a particular resource, that resource name is appended to the task name, for example a device I/O task might have the device name appended to the task name.

Most permanent tasks spend most of their time waiting for work (posted via a work queue), or waiting for a time interval to elapse (for handling regularly occurring events).

Here are some representative permanent tasks:

Task Name	Description
ArpCheck	Periodic send/receive of ARP messages.
CopyModMon	Monitor copy sets for excessive modified blocks.
DateTime	Date/time update.
DevScan	Periodic scan of physical devices.
DstgTask<devname>	Data destaging task.
FTPInit	Wait for an FTP connection.
HTTPInit	Wait for an HTTP connection.
IOCTask	Periodic statistic gathering.
IPMI	Handle periodic IPMI updates.
KbyCmd	Handle locally attached keyboard input.
LCDTask	Handle periodic LCD and LED updates.
LostCTask	Monitor for lost TCP/IP connections.
PrefTask	Data prefetch task.
Persist	Handle transmission of TCP/IP persist messages.
PingChk	Periodic send/receive of ICMP echo messages.
RdstTask<device>	Remote data destage task.
RmsgTask<port>	Handle reception of TCP messages from a single port
Rs2sTask<IP address>	Send ASC-to-ASC requests.
Rd2dTask<port>	Send ASC device requests.
RstgTask<device>	Remote data stage task.
RTCPTask<port>	Handle reception of TCP messages for a given socket.
RwrtTask<device>	Send data to remote system task.
ScrnTask	Handle locally attached display output.
SernUPSTask	Monitor a UPS.
SNMP	Handle reception of SNMP requests.
SNMPTrapMonitor	Monitor possible SNMP trap conditions.
SNMPTrapTask	Send SNMP traps.
StagTask<device>	Data staging task.
TCPAbort	Serialize TCP/IP abort requests.
TCPCtrl<IP address>	Handle ASC-to-ASC requests.
TCPDtrl<port>	Handle ASC device requests.
TelnetInit	Wait for a Telnet connection.
Throttle	Handle system resource throttling.
TsrvTask	Time server task.
WcmpTask<device>	Send data destaged to local system task.

Transient Tasks

Transient tasks are created to perform a specific function, then exit. An example might be a task to establish a copy pair. Generally, the ASC is designed to minimize the number of transient tasks whenever possible (to shorten the dispatcher path, and to avoid task thrashing).

Here are some representative transient tasks:

Task Name	Description
BscnTask	SCSI/Fibre Channel bus scan (device location/initialization).
CopyTask	Copy establish task (spawns copy establish read and write tasks to double

	buffer copy data).
CseRtask	Copy establish read data task.
CseWtask	Copy establish write data task.
IOB-address-S	Process a single data stage request.
IOB-address-D	Process a single data destage request.
TCPCStart<IP address>	Start an ASC-to-ASC TCP/IP connection.
TCPDStart<port>	Start an ASC data connection.

Device Drivers

The *Design of a Reduced Function Operating System* describes RTOS device drivers in some detail.

The RTOS and ASC were designed to isolate the RTOS system and ASC application from physical device hardware dependencies, and to make the driver interface as simple as possible.

Back-end driver I/O is straightforward – an IOB is passed describing the operation to be performed.

The data transfer portion of *front-end* or host I/O very closely resembles back-end I/O and is invoked using IOB's and the same driver calling sequence. A number of additional flags in the IOB direct the driver to disconnect, present ending status, present sense data, hold or release the bus, and so on.

A function unique to host operations, is host *initial selection*. A host selects the ASC, causing the driver to receive an interrupt. The driver allocates an IOB, collects initial SCSI information (the SCSI command (*CDB*), SCSI message data, device identifier and LUN, tagged queuing data, and so forth), then passes it to an ASC kernel mode routine for validation and initial processing. This routine will initiate ASC task work if necessary and will fill in the passed IOB with data pointers if the driver can immediately transfer data to or from the host.

Host drivers are interrupted when a bus reset, host initiated reset or abort, or a bus related error occurs (such as a parity error). These conditions cause the driver to allocate an ABORTREQ and call a common kernel mode routine to handle the condition.

Device emulation and generic error recovery are common to, and independent of, all host drivers.

The RTOS provides the ability to detect *missing* and *hot* interrupts. The device driver is notified of these conditions so that it may take hardware-specific recovery actions.

Coding Conventions

All ASC and RTOS code is *read-only*. There is no self-modifying code, and no writable data segments.

All code is reentrant – each task, interrupt handler, kernel request, and so on, that invokes common routines executes these routines from the same in-storage copy (only the parameters and stack space passed to the routines are different). This minimizes the code size, and optimizes the use of internal processor instruction caches.

All permanent data are referenced via defined data structures. Temporary variables and data structures are allocated in task or system stack space.

All queues of data structures use a standardized doubly linked list. RTOS linked list routines are multiprocessing-safe. All permanent queues can be located from a common base block.

Future Enhancements

To be completed.