# IBM TotalStorage Enterprise Storage Server: A designer's view

by M. Hartung

In this paper, we describe the background, objectives, and major decisions associated with the design of IBM TotalStorage™ Enterprise Storage Server® (ESS), IBM's high-end disk storage system. We first present a brief history of disk storage development over the past three decades and then describe ESS architecture and basic functions. Next we discuss the goals associated with the design of ESS and the methods used to achieve these goals. We then explore some design decisions that significantly affected ESS architecture and performance, and we conclude with some comments about possible future enhancements.

In the 1960s and 1970s, control units served as gateways that provided attachment of various input and output devices to a relatively small number of host channels. This technology was used primarily in System/360* and System/370*.[1] In addition to providing attachment for a variety of devices, the control unit provided the conversion between the channel protocol and the protocols for the attached devices. The control unit also permitted multiple channels from the same host, or from different hosts, to attach to the devices it controlled. The control units provided limited error recovery, as well as error detection and isolation.

In 1981, a read cache (also known as write-through cache) was introduced into storage control units in the 3880 Models 11 and 13. In 1988, a write cache (write-in cache) was introduced into storage control units in the 3990 Model 3.[2] In the late 1980s and early 1990s, control units included RAID (redundant array of independent disks) technology to provide additional reliability for the attached storage.[3] By this time, storage control units had powerful microprocessors, a large read cache, and a large write cache. Next came adding storage-based functionality to the storage control units.

A storage control unit provides sole access to the attached devices and all access to the associated storage/data flows through the storage control unit. Because the storage control unit was uniquely positioned to provide function associated with the stored data, and because it could be equipped with needed processing capability (i.e., processor and memory), it became the focal point for new storage-oriented functionality. A result was replication services, which in effect means creating copies of data for various purposes. The addition of replication services to the function already present (function that exploited the read cache, the write cache, and RAID), completed the transition from the control unit as gateway and aggregator to the storage server, a system whose advanced functions far exceed the storage access function.

Networked storage was the next major development in storage systems. Storage area networks (SANs) enable multiple hosts to work with a common set of storage systems. Both SANs and network attached

storage (NAS) permit multiple servers to share storage systems and facilitate the sharing of the stored data. Networked storage contrasts with "direct-attached" storage, where a storage device is available to just to a single server. With direct-attached storage, no opportunity exists for sharing the storage resource or the data stored on it.

The storage consolidation enabled by storage networking provided an important shift in host-and-storage topology for the UNIX** and Microsoft Windows NT** environments. Historically, the UNIX and Microsoft Windows NT storage environments consisted of direct-attached disks, either internal or external. Disks attached to a host were owned by that host, and unused disk space was not shared with any other server. The relationship was so close, in fact, that the storage could rarely be moved to a dissimilar server. Because storage resources across hosts, be they homogeneous or heterogeneous, could not be pooled together, the purchasing decision for a host was irreversibly tied to the purchasing of storage components. Storage consolidation, however, separates the two purchasing decisions and allows customers to upgrade or replace hosts (even to new platforms) without purchasing new storage. Conversely, storage can be upgraded without installing new hosts.

Another important consequence of storage consolidation is the introduction of storage-based functions, such as replication services. Using the function provided by the storage system, an enterprise can build a single set of procedures and processes for data-related activities, such as disaster recovery or data archiving. These processes and procedures are the same for all data in the enterprise and are applied uniformly across heterogeneous hosts. Such processes cannot be completely independent of the host platform, but the core function consistency is of significant value in that all data have the same high level of usability and protection.

Storage has seen dramatic price reductions of 40 to 60 percent per year. This cost reduction makes possible a rapid increase in configured storage, and more data being immediately accessible to the enterprise. As the configured storage grows, the cost of managing this storage becomes a significant inhibitor to adding more storage. Management costs can grow exponentially with storage capacity. These costs are primarily the cost of human resource, first as payroll, but also as the cost of acquiring and maintaining the required skills.

In order to alleviate the problem of the rising cost of managing storage systems and enable continued growth of installed storage, systems management software for storage systems is being enhanced. Policy-based storage management (PBSM) is directed at reducing the cost of managing storage. PBSM automation maps enterprise policy to various constraints and self-optimizing mechanisms to be used when implementing software components. Ideally, the enterprise policies and goals are formulated as input to PBSM in the language used to manage the enterprise. In contrast, today administrators must define configurations by manually translating business requirements into system requirements. The PBSM software enlists the appropriate technologies and resource controls (e.g., service level agreements, quotas) to support enforcement of enterprise policies through the operation of the information processing system. PBSM usually operates with most of the solution components. It can also provide overall monitoring and a feedback control loop to support consistent delivery of the requested policies.

Another major factor in the evolution of storage systems is the increasing role of autonomic computing (i.e., self-healing, self-optimizing, self-configuring, and self-protecting).[4] For over 30 years, self-healing has been a recognized requirement in enterprise-class storage systems and has come to be known as "continuous availability." The premise of continuous availability is that no single failure will result in loss of data, access to data, or functionality. Scheduled events such as maintenance and microcode load, as well as unscheduled events such as failures, must be accomplished without impacting system availability or functionality. While the self-healing requirement has been relatively constant over the past 30 years, the self-healing requirement for scheduled events has become more stringent. New business requirements such as 24-hour operation and worldwide accessibility have led to the loss of the weekly or monthly batch windows that were once available for scheduled activity.

Self-optimizing has become a more important requirement for storage systems since the introduction of read caching, write caching, and advanced functions. The system must allocate system resources (e.g., read cache, write cache, and processor) based upon the current demands on the system. This requirement is now prominent in storage system development.

Self-configuring and self-protecting became more important requirements with the introduction of storage area networks (SANs). The additional complexities of configuring networked storage led to increasing requirements for intelligent self-configuring. The "universal" access provided by networked storage led to a dramatically increased requirement for self-protecting, as only those with proper authorization could be allowed to access data stored within the system data.

In summary, over the decades, storage evolved from the simple role of media, where hosts stored data, to powerful storage servers. The declining cost of physical storage led to a greater focus on the cost of managing storage, because this remains the primary inhibitor to the growth of the installed storage. In delivering storage the focus has become the storage system that can contain the management costs. The realization of such function is based on new techniques (e.g., PBSM) implemented in the host as well as in the embedded software of the latest storage servers. IBM TotalStorage Enterprise Storage Server (ESS) is a premier example of a storage server designed to meet these requirements.

The rest of this paper is organized as follows. In the next section we describe ESS architecture, discuss its server-based design, and describe the basic operation. Then we discuss the ESS objectives and the methods used to achieve them. In the following section we explore some design decisions that significantly affected ESS architecture and performance. We conclude with some comments about possible future enhancements.

## ESS hardware and embedded software

ESS is IBM's most powerful disk storage server. It supports a multitude of hosts in a heterogeneous open-systems environment. ESS supports direct connection to SANs and provides a number of advanced functions for data duplication and backup and disaster recovery. We first discuss the server-based design of ESS and then describe its basic operation.

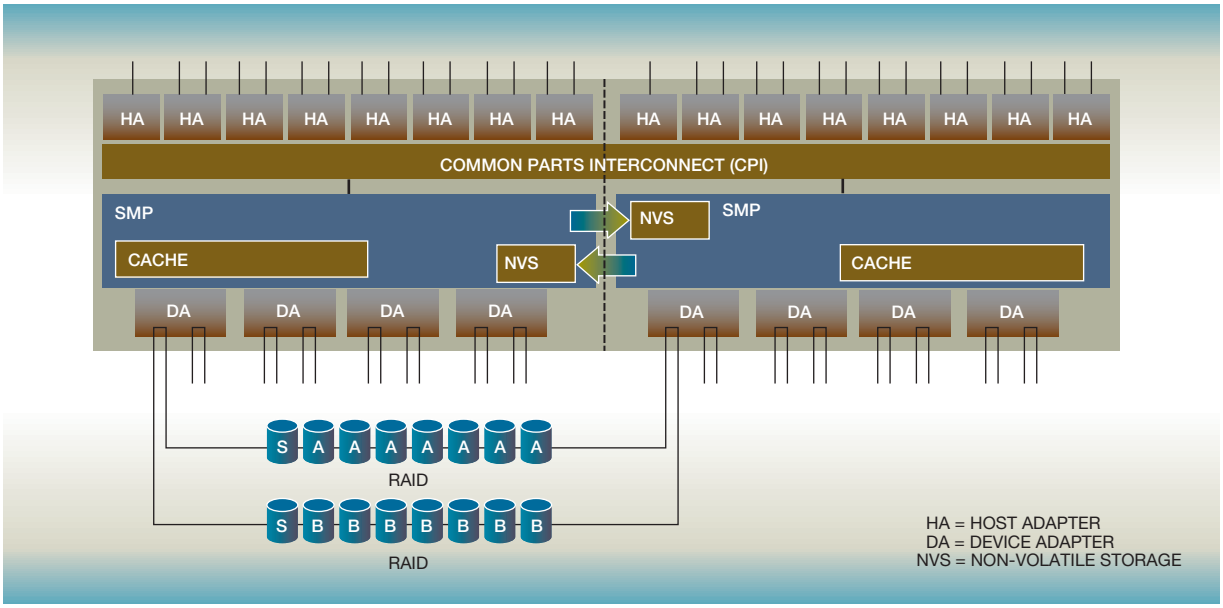
**Server-based design.** ESS is a server-based storage system configured from two IBM pSeries* symmetrical multiprocessors (SMPs).[5] The SMPs cooperate to provide and support the function, performance, and continuous availability so critical to high-end storage. Each SMP has one or more host adapters that provide host connectivity. Each SMP also has one or more device adapters that attach to disk devices.

The pSeries SMPs are the same as those used in the pSeries processor family. Using SMPs as the primary processing engine sets ESS apart from its two main competitors, Symmetrix** from EMC Corporation[6] and Hitachi Freedom Storage Lightning 9900 V Series** from Hitachi Data Systems.[7] The SMPs are powerful processor systems that incorporate state-of-the-art server technologies. The pSeries development group designed a balanced and tightly integrated set of memory, bussing, processor, and cache components that is ready for use as the ESS processing core. Although other vendors use standard parts, such as processors, in their storage systems, they develop in-house the rest of the server infrastructure. ESS has a carefully tested and tuned processing core which includes the read cache (the SMP memory is used as the ESS cache). To this core, the ESS team added host adapters, device adapters, and embedded software. The ESS team focused on delivering system functionality—the embedded software and adapters—important to customers. The consistency of the pSeries architecture from one generation to the next enables the embedded software to be ported to a new platform with little change to the established software base. This greatly reduces the possibility that the port will introduce instability into the system. Moreover, ESS introduces new core technology with each enhancement of the pSeries product line. The resultant improvement in price and performance is further leveraged by the increasingly stable software base. Function is added at an annual rate of about 200000 to 300000 lines of embedded code.

The first two generations, ESS Model E and Model F, used the SMPs as designed for the pSeries platform. Starting with Model 800, however, ESS architects and developers collaborated with the pSeries team of architects and developers so that the SMPs had capabilities that directly benefit ESS. Increased interaction between the two teams has led to an examination of the various infrastructure elements (e.g., the hardware management console, power, packaging) in order to determine whether the commonality of function could be extended. Platform convergence, the endpoint for extended commonality, is leading to storage servers that consist of the pSeries server and its support structure along with the embedded software and the hardware adapters.

Significant benefits have resulted from merging the development of pSeries and ESS. ESS benefits from a fully assembled, leading edge processor and memory system. Both ESS and pSeries benefit from the shared knowledge and the efficiencies gained in the

Figure 1 A representation of an ESS unit with two SMP nodes




joint effort. The pSeries architects bring more extensive experience in systems management and autonomic computing, whereas the ESS team brings more extensive experience in autonomic computing (particularly self-healing and self-tuning), reliability, and serviceability. The improvements derived from this pooling of experience enhance both product families and directly benefit those who buy these products.

**Basic operation.** As shown in Figure 1, each ESS unit consists of two *nodes*, where each node has an SMP, one or more host adapters, one or more device adapters, a (read) cache, and a nonvolatile storage (NVS). There are two nodes in the ESS unit, for redundancy. In the unlikely event of a node failure, the other node is capable of taking over the resources of the failed node and continues the system operation until the failed node is repaired and restored to full operation. The transition from two operational nodes to a single-node operation on failure is known as *failover*; the restoration of two-node operation is called *failback*.

The SMP, the host adapter, and the device adapter include powerful processors with embedded software. The embedded software on the SMP provides most of the ESS unit functionality. The main memory of the SMP also serves as the read cache for the node. The storage resource is presented to hosts either as logical units (UNIX, Windows NT, iSeries*) or as volumes (zSeries*). We use the term *virtual disk* to refer to both logical units and volumes. Each virtual disk is owned by one of the nodes. Each node processes all requests for its virtual disks and caches data associated with those virtual disks in its read cache. Modified data is mirrored in the nonvolatile storage on the other node (see Figure 1). Mirroring of modified data helps avoid loss of data or damage to data in the event of a potential failure.

A host adapter provides host connectivity. It contains one Fibre Channel SCSI (small computer system interface) port, one FICON* (Fibre Connection) port, two ESCON* (Enterprise System Connection) ports, or four parallel SCSI ports. The embedded software, which runs on a PowerPC*, provides the interface protocol. Host adapters also perform the dual write, one to the read cache on the owning node and one to the nonvolatile storage on the non-owning node (for mirroring of modified data).

The division of tasks amongst the various components is designed to help optimize performance. One can get a notion of how this is done by considering some simple operations. Consider, for example, a

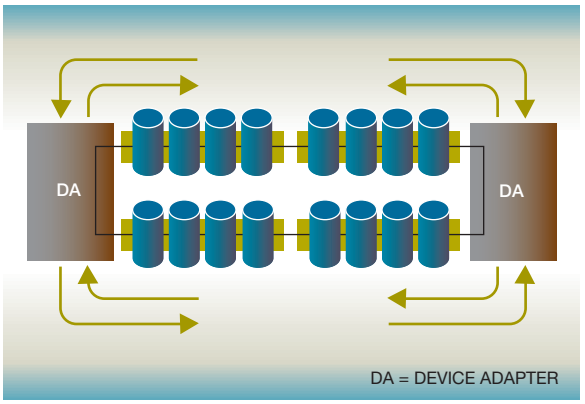
read cache hit. The host adapter receives the request, interprets the received command, and determines that the request is a read. The host adapter passes the request to the SMP that owns the virtual disk represented by the logical unit number (LUN).[8] The SMP locates the requested data in the cache and passes a pointer to that data back to the host adapter. The host adapter completes the read request by sending the data to the server and the I/O request is terminated (some details were omitted in this description, for brevity).

Consider now a write operation. The host adapter receives the request, interprets the received command, and determines that it is a write request. The host adapter sends the request to the SMP that owns the virtual disk, and the SMP searches the cache to determine if the data to be written have space allocated in the cache. If not, cache space is allocated for the data that are to be written. The SMP also allocates space for the data to be written in the nonvolatile storage on the other node. The SMP passes both the cache and nonvolatile storage addresses to the host adapter, and the host adapter writes the data received from the host in both the cache and the nonvolatile storage. The I/O request is then terminated.

The device adapters provide for attachment of physical drives to the node. Figure 2 shows the operation of the serial storage architecture (SSA), a high-performance serial connection technology for disk drives. Data are sent from the adapter to the first disk drive on the loop and then passed around the loop until the data arrive at the target disk. Figure 2 shows the multiple simultaneous data transfers that can take place on an SSA loop (the loop may have local groupings of disks called domains). Two device adapters, one from each node, are paired on a serial storage architecture (SSA) loop, providing two loops on each device adapter pair. Both device adapters in a pair are active. In case of a node failover, the device adapter on the surviving node is capable of taking over all of the physical devices that were owned by its partner device adapter on the failed node. The device adapters also provide the available RAID capability, which includes both RAID 5 and RAID 10.[9] A RAID configuration uses multiple drive spindles to provide data redundancy or error correction, or both.

The embedded software on the SMP provides for management of both the read cache and the nonvolatile storage. Management of the read cache has at its core a least-recently-used (LRU) algorithm for

Figure 2 SSA operation




space allocation. With LRU, the least recently referenced entry in the cache is the one chosen for replacement. The embedded software contains many enhancements to LRU, such as predictive staging of data when a sequential reference pattern is recognized, or sequential limiting, which selects data that was referenced sequentially for replacement as soon as the sequential reference moves past the data.

Error recovery procedures are found in most ESS components. Because of the stringent requirements for fault tolerance associated with continuous availability, about half of the embedded software involves error recovery.

## Goals and methods to achieve them

The goals associated with an ESS include virtualization, transparent incorporation of new technology, continuous availability, high performance, portability of code, support for continued growth of function, support for function integration into ready-to-use solutions, and lowering the total cost of ownership.

**Virtualization.** ESS uses all its resources, such as read cache, write cache, and physical devices, to present the storage resource to hosts as virtual disks. Although the realization of a virtual disk requires the allocation of various such resources, the particulars of this allocation do not affect how servers access the virtual disk or the result of such access on its data. For each virtual disk, ESS can read and write cache its data, provide RAID 5 or RAID 10 modes, and perform an automatic RAID build in the event of a de-

vice failure. On failover, the virtual disk of a failed node is designed to transfer to the working node without affecting the virtual disk properties or its data.

**Transparent incorporation of new technology.** A key ESS strategy is to provide a consistent architecture that covers the ESS generations already delivered and those yet to come. This strategy is intended to increase the ability of customers to develop policies and practices that span multiple product generations. Virtualization is an important factor in this strategy as it permits the introduction of new technologies without affecting the virtual disk interface. This permits the ESS team to select the best combination of technologies for delivering the storage system functionality. IBM is thus better able to deliver a balanced set of technologies that are optimized for storage serving without requiring customers to understand or accommodate the underlying technical aspects.

**The basic strategy employed to achieve continuous availability is the use of redundancy and highly reliable components.**

An example of incorporation of new technology that remains transparent to applications is offered by SSA, a device attachment method introduced by IBM for its performance, reliability, serviceability, and availability features.[10] There are three prominent methods for attachment of high-end devices to storage systems: SSA, parallel SCSI, and fibre-channel-arbitrated loop (FCAL). The drive vendors delivered the device attachments in the order of the amount shipped of each type that was sold: parallel SCSI first, FCAL second, and SSA third. ESS uses SSA for device attachment. Thus the drive technology required by ESS was delayed from the initial delivery of that technology for parallel SCSI. In the spirit of delivering the best technologies available, a chip was built that can bridge between SSA and parallel SCSI, and it was deployed with each ESS parallel SCSI drive. Because each device has its own private SCSI bus, the limitations of parallel SCSI do not affect the previously mentioned attributes of the SSA fabric. Thus ESS delivers the values of the SSA device fabric connection while delivering the most current device technologies. With this approach, ESS offered both 10K RPM (rotations per minute) 146 GB (gigabyte) drives and 15K RPM drives before its two main competitive products, Lightning[7] and Symmetrix.[6]

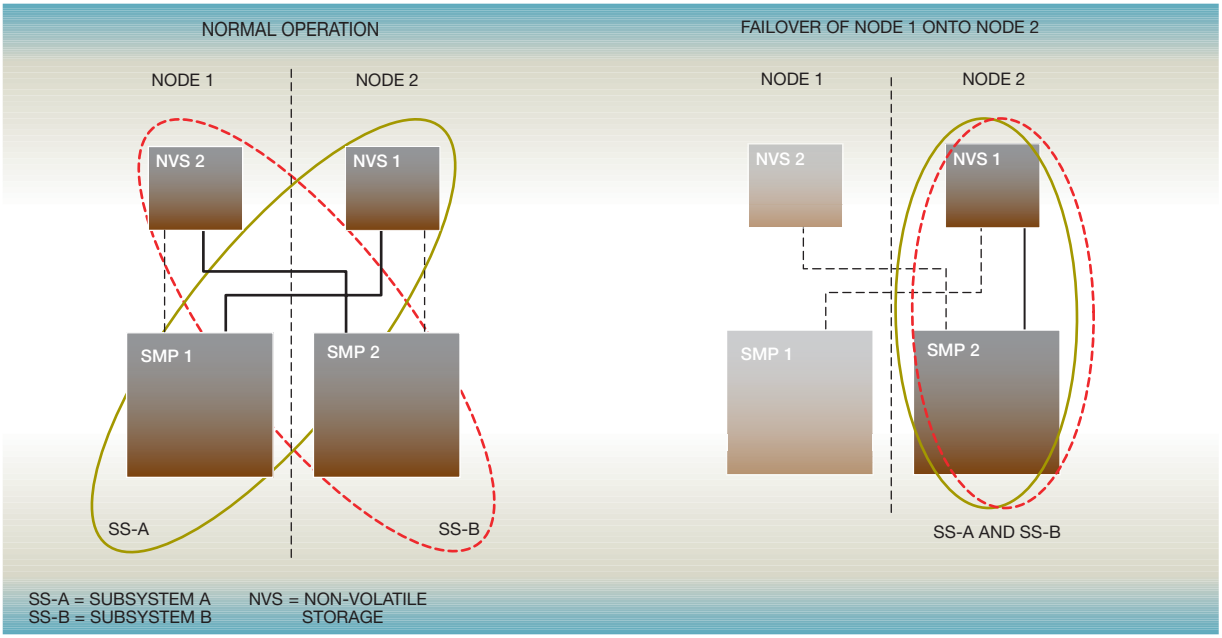
**Continuous availability.** Continuous availability requires that single points of failure be avoided so that there is no loss of data, loss of access to data, or loss of function. The continuous availability requirement, however, does not exempt all multiple failures. Multiple simultaneous events, for example, can result in loss of continuous availability. One must consider the repair time for determining whether multiple events are indeed "simultaneous." If the system continues to operate without the failed part, then subsequent failures occurring before repair of the original failure are not viewed as simultaneous. The probability of occurrence must be included in the calculations to determine the continuous availability achieved by the design. If a second failure is likely to occur as a result of the original failure, it cannot be considered as independent. Thus a continuous availability solution must account for the simultaneous occurrence of the dependent failures.

The basic strategy employed to support continuous availability is the use of redundancy and highly reliable components. The various ESS components are designed for early failure detection, and for failure isolation of the smallest possible component of affected hardware. The number of hardware components affected determines the scope of failure. Limiting the scope of failure to those components that must be deactivated limits the effect of running without the hardware. Limiting the scope of failure is another design objective for ESS. In a similar vein, the hardware that is unavailable during repair determines the scope of repair. Limiting the scope of repair is another important ESS objective.

The basic design covers early detection of failures, limiting the scope of failure to the smallest extent possible, and reconfiguring the system to continue operating until the repair is performed. After concurrent repair is performed, the hardware is reconfigured to include the newly restored components as part of the system, and operation then continues.

Failover and failback at the node level exemplify the design of error recovery functions within ESS. The hardware and software components provide for early detection and isolation of failures. Thus loss of a node is frequently detected and identified as a failure. To

Figure 3 An ESS unit with two nodes




complement these direct notifications of failure, each node performs a "heartbeat" on the other node to confirm that it is operational. The heartbeat mechanism uses two independent hardware paths to ensure that heartbeat path failures do not erroneously show up as "node failure." Should the heartbeat mechanism determine that a node is not operational, the surviving node records the failure. Whether failure is reported by failure detection code or as a result of the heartbeat, the surviving node updates control data in well-known areas on shared disks (status tracks) so that the node declared "failed" does not inadvertently try to continue or resume operation. Such mechanisms support system integrity. Two nodes operating independently could have unpredictable consequences on the system operation and on the data stored with the system.

The left half of Figure 3 shows a two-node system in normal operation. The ensemble consists of subsystem A (runs with SMP 1 and NVS 2) and subsystem B (runs with SMP 2 and NVS 1). The right half of Figure 3 shows the failover of node 1 onto node 2. The surviving node assumes ownership of all host adapters (initially ownership is shared), the nonvolatile storage on the node, and all physical devices in the system. The surviving node then restores redundancy by destaging (writing modified user data from cache to the disk arrays) all modified data—data in its read cache and data in the local nonvolatile storage belonging to the failed node. The node then allows the host adapters to recognize that all virtual disks are owned by the surviving node, and the node resumes operation using the local nonvolatile storage as its nonvolatile storage. Operation continues in this way until the failed node is repaired. Following repair, all elements—the virtual disks, nonvolatile stores, read caches, device adapters, and physical devices—are restored to their original ownership, and operations resume. Both failover and failback are designed to take place while the system continues to operate. Activity is suspended for a matter of seconds and then immediately resumes, without significant disruption to the system operation.

Another example of concurrent repair is the automatic RAID rebuild that occurs upon the failure of a physical device that is part of a RAID. A typical device configuration includes one or two spare devices per SSA loop. When a device that is part of a RAID fails, the device adapter that owns the RAID assigns a spare device, reads the data from the surviving devices, and rebuilds the contents of the spare device.

This activity occurs concurrently with normal read and write activity to the RAID undergoing the rebuild.

Thus, ESS failure recovery is designed to be autonomic—not requiring human involvement—and nondisruptive. The system then records the information that enables service personnel to perform the repair.

**Performance.** ESS is designed to optimize both throughput and response times. All work performed by the system should "spread" so that all the possible hardware components are processing useful work as often as possible. The read cache and write cache support multiple simultaneous data transfers. The RAID configurations, RAID 5 and RAID 10, include striping of the data from a virtual disk across multiple physical devices, increasing the chance that multiple requests for that virtual disk will access different physical devices. The SSA loops provide for multiple simultaneous data transfers on the loop, known as *spatial reuse*, unlike bus arbitration, which permits only one operation at a time on the bus. The SSA-to-SCSI bridge on each SCSI device provides a private SCSI bus for each device. Because there is only one target on the bus, the bus arbitration associated with SCSI, one operation at a time, does not affect throughput. The two ESS SMPs are either 4-way or 6-way, and all of these elements operate in parallel. The host adapters and the device adapters, likewise, all operate in parallel. In addition, all of these elements and more are designed to operate simultaneously. The virtual disks are mapped onto the hardware in a way that enables concurrency in the execution of a logical operation.

ESS architecture attempts to make all work items "visible" and thus enable the work to be executed as soon as synchronization requirements permit and resources—such as physical devices—are available. Tag command queuing (the support of multiple concurrent outstanding requests to the same virtual disks) for the SCSI protocol permits multiple requests using the same LUN to be issued to the system at one time. Thus, the system can view the work and execute the requests as resources become available. Ordered writes to the same LUN can be specified in this case, with the order taking precedence over resource availability. Otherwise, resource availability determines when work can be completed.

Parallel access volumes (PAVs) provide similar functionality for zSeries I/O requests. PAVs help eliminate an artificial point of contention that has existed in the zSeries architecture since its inception as System/360. In the original System/360* I/O architecture, disk drives performed one operation at a time, seek and read or seek and write. The control units could not queue multiple requests and because the control unit could execute the operations only in the order received, queuing at the control unit would not have improved system operation. Operations were designed to be queued in the operating system where the resources (e.g., processors and memory) existed to manage the queues. The device-busy status was used to indicate to one host (an operating system) that another host was using the same volume. The device-available status would later indicate that the volume had become available. Within a single host, the I/O system used a software structure—the unit control block (UCB)—as part of the meta-data for controlling I/O operations. UCBs were designed so that only one I/O could be in execution on a given volume at a time. If an application issued an I/O request to a volume that had an outstanding I/O from that same system, the I/O operation was queued with UCB-busy status until the volume was free to execute the queued operation. These two mechanisms, device busy and UCB busy, were among those that enforced the architectural requirement that a volume have only one I/O operation active at a time. The restriction was not an issue for volumes until the 1980s. In 1981, read caching made it physically possible for reads from cache to overlap with other read hits, read misses, or writes for a single volume. In 1988, write cache made it physically possible for writes to overlap with other writes or with reads for a single volume. In the early 1990s, introduction of RAID striping of a logical volume across multiple physical devices made it physically possible for physical device reads for the logical device to overlap with other physical device reads for the same volume. However, in all cases, the I/O architecture prevented the overlap because the system could not see the multiple requests for a volume. Therefore, no chance existed for overlapped execution.

Two new ESS features were introduced in 1999 that were intended to eliminate the artificial bottleneck created by the zSeries I/O architecture. One, multiple allegiance (MA), provided for "simultaneous" I/O to a single volume from multiple systems. The other, PAV, provided the same capability from a single system. In both cases, ESS is designed to enforce access restrictions that are necessary to help preserve the integrity of the data. For example, multiple reads can execute in parallel without restriction, writes

and/or reads to different volume areas can execute in parallel without restriction, and reads or a write to an area for which a write is pending must wait for the write to complete before the next request can execute. Beyond the necessary synchronization, operations can execute in parallel as the system permits.

For ESS and the zSeries operating system, introducing PAVs involved significant changes. Each volume has a base address and 0 to 255 alias addresses, and each address (base or alias) can be used to issue an I/O request to the volume. Thus, a volume can receive up to n + 1 (n being the number of aliases) I/O requests before another request gets queued in the host. Whereas significant performance enhancements can result from PAVs, they also introduce an additional burden for the storage or system administrator. Yet another set of limited resources—the 256 device addresses—must be allocated according to the expected usage of the volumes in the system.

**With dynamic PAVs, the set of aliases for a logical subsystem is treated as a pool.**

A more autonomic solution was provided by dynamic alias management, or dynamic PAVs.[11] With dynamic PAVs, the total set of aliases for a logical subsystem is treated as a pool. The Workload Manager component of the zSeries operating system works with ESS to allocate the aliases to the volumes that are currently the busiest. Thus the limited resource is allocated automatically according to the current demand. This allocation provides the best of PAV performance without putting the allocation burden upon the administrator.[11] In addition, dynamic allocation of the aliases results in a more efficient resource use. In an experimental environment designed to represent real workloads, a configuration with a pool of dynamically allocated aliases of between 0.25 to 0.5 alias per 3 GB volume, the performance benefit was equivalent to an environment in which each 3 GB volume had three statically allocated aliases. Dynamic PAVs has provided one of the most significant performance enhancements for the zSeries I/O attachment since the introduction of read caches.

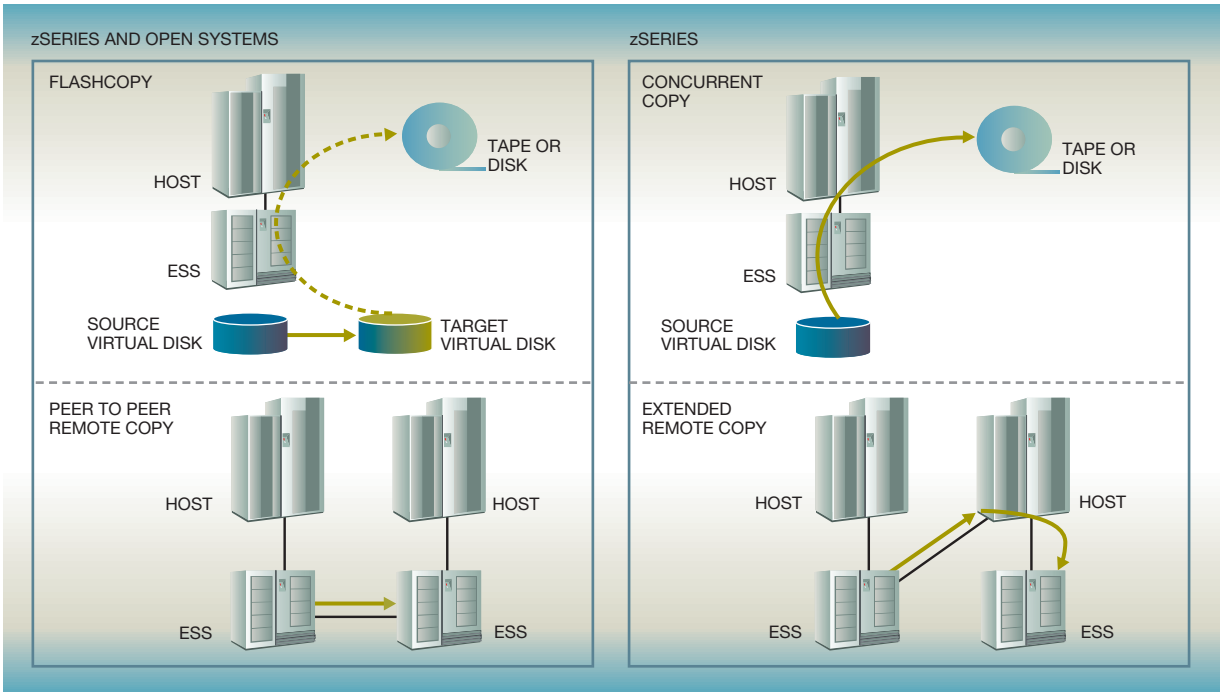
Dynamic PAVs, MA, and other parallelism built into ESS are intended to provide many significant benefits. Experience has shown that hot spots (intense activity against specific logical volumes, data sets, or files) are alleviated. The autonomic handling of resource allocation eliminates a significant management cost that plagued many environments. ESS parallelism also enables maximum exploitation of the installed hardware, helping to provide a significant price performance advantage to the customer.[12] Artificial contention and the resultant impact on throughput and response time are eliminated. For example, cache hits execute simultaneously with cache misses rather than sequentially. Likewise, random transfers of small blocks of data need not queue behind unrelated activities that are moving large amounts of data in each request and therefore causing long delays. Also, dynamic PAVs permit additional pathing to be provided as needed for very large volumes. This facilitates introduction of 9 GB and 27 GB volumes for zSeries, permitting more storage to be configured with the 256 device addresses available for a system.

Responsiveness and throughput are also driving the ESS design. Although AIX* provides a rich set of functions—system bringup, hardware diagnostics, some hardware reconfiguration—it does not provide the response time and throughput necessary for a high-end disk storage system. Response time and throughput considerations led to the introduction of a kernel mode extension (extensions to the operating system) to address these performance requirements.

**Portability of code.** The portability of the embedded software from one ESS generation to the next is essential. Each function, implemented once, is designed to be easily ported to the new product. Thus the embedded software continues to grow in stability and function. The chosen pSeries platform architecture and AIX* provide the infrastructure stability that helps support this capability. In addition, an added layer of code, called platform code, provides the primary interface to the hardware and to AIX. This layer, which is less than 10% of the entire embedded software, absorbs most changes (cost of porting) from one generation to the next.

**Support for continued growth of function.** ESS has added new functions at a significant rate. In 2000, Fibre Channel SAN capability was introduced and replication services were extended. In 2001, FICON was introduced, replication services were further enhanced, and new devices were supported. 2002

Figure 4 ESS advanced copy functions




brought a new hardware platform, new physical devices with improved performance or capacity, or both, new SCSI devices, and more replication services. ESS offers, with each hardware generation, a growing functional base, increased stability, and increased speed (history shows a doubling of speed with each generation).

Replication services are used for both data and storage management and for disaster recovery. They provide for creating copies of data that can be used for data sharing, disaster recovery, and restart in the event of human or application error. The replication services use two major techniques: point-in-time copy and continuous copy. Point-in-time copies, as the name implies, capture an image of the storage contents at a specified time and preserve that image for any desired purpose. Point-in-time copies are used for checkpoint, archive, disaster recovery, and data sharing. Continuous copies, on the other hand, are used primarily for disaster recovery, where a copy of the data is kept relatively current and geographically separated from the original data. The copy can be used to restore operations should the original copy of the data be destroyed or become inaccessible or unusable. The simplest version of continuous copy is synchronous continuous copy. The target is intended at all times to be the same as the source copy. Synchronous copy covers most of the requirements for continuous copy solutions, but distances between the source and target are limited by application performance requirements and the effects of the speed of light. Figure 4 illustrates the ESS advanced copy functions. FlashCopy* is a point-in-time copy; the resulting virtual disk may be the source for an optional copy to tape or disk. Concurrent Copy, also a point-in-time copy, is a z/Series function that works with either volumes or data sets. Peer-to-Peer Remote copy (PPRC) is a synchronous continuous copy. Extended Remote Copy (XRC), a zSeries function, is an asynchronous continuous copy suitable for large distances.[14]

In order to accommodate the longer distances that customers require, ESS also includes asynchronous continuous copy. Asynchronous continuous copy allows the distance between the source and the target to be increased without affecting the performance at the source. The target is a coherent copy of the source data; that is, write sequence is preserved at

the target. In the event of a complete loss of the source, the target is designed to be a consistent version of the source at some point in the past. The maximum age of data is specified by the recovery point objective for the data. For the asynchronous continuous copy solution, recovery point options range from seconds to days but are frequently in seconds or a small number of minutes. Asynchronous continuous copy solutions are constructed from integrating point-in-time copies with data mover solutions, as well as by grouping time-contiguous sets of writes and permitting each specific set of writes to be reflected atomically (either all writes occur or none occur) at the target. Replication services are provided at many levels in the solutions that we integrate. They are provided, for example, by intelligent systems, by middleware, by file system and NAS, and by virtualization engines.[13]

**Support for function integration into ready-to-use solutions.** Although the value of delivering ESS function is important within information technology solutions, it does not represent the main goal. The ESS team jointly with application and services vendors (both IBM and non-IBM) strive to integrate ESS into ready-to-use solutions for the enterprise.

**Cost.** Business efficiency as represented by cost is key to the adoption of information technology initiatives. Storage purchase cost is particularly critical as it can represent up to 75 percent of total IT purchase costs. What's more, because the cost of managing storage dominates the total cost of ownership, elements such as autonomic features and policy-based storage management are receiving a lot of attention. For implementations of policy-based storage management, in particular, see References 15 and 16.

## Major design decisions

Countless decisions are made when a storage server as complex as ESS is designed and implemented. We present in this section some of the design decisions which significantly affected ESS architecture and performance.

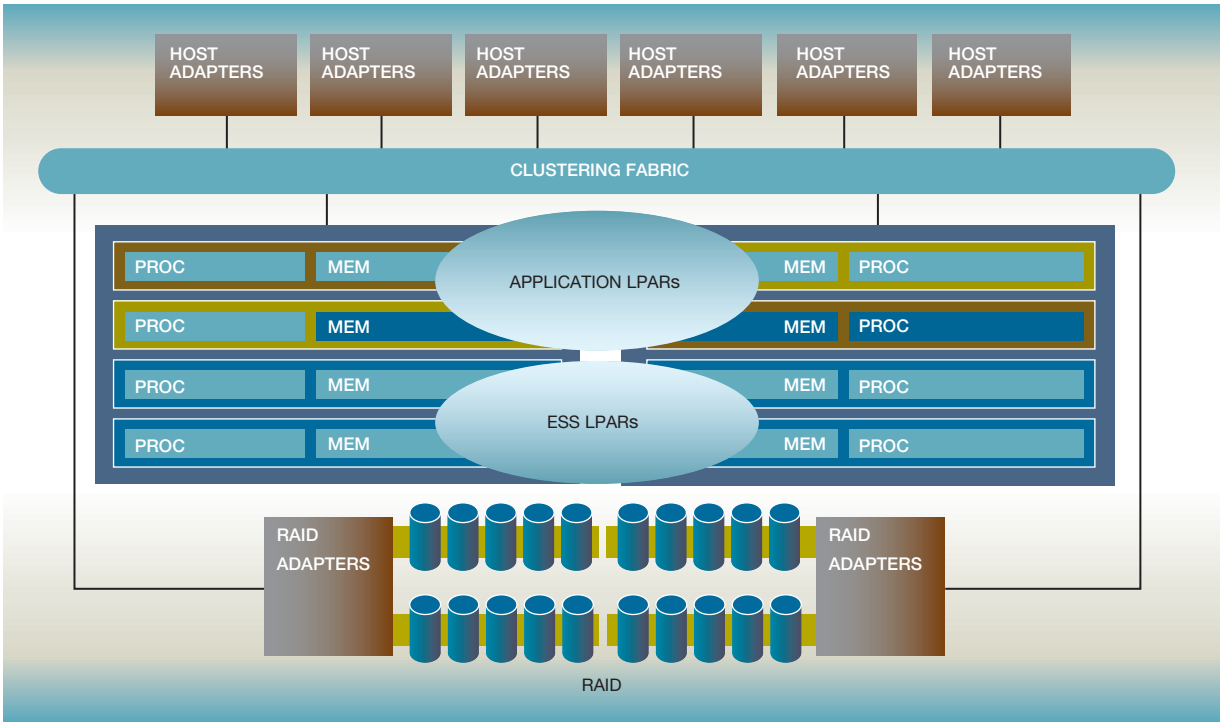
**SMP main memory as the read cache.** The decision to use SMP main memory as the read cache went counter to the IBM experience with high-end storage servers. Since the first read cache in a control unit was delivered in 1981, the storage system designers had architected, designed, and implemented the cache separately from the processor. The cache had been designed as a single memory unit with redundant components and with a nonvolatile storage for mirroring modified data. Redundant processing units attached to the "single shared" cache provided the embedded software that delivered system function.

Following the decision to use SMPs as the processing engines, a choice was to be made between building a single shared cache as before, or using the SMP main memory as the read cache. Building the single shared cache would likely require that for each significant delivery of new SMP technology, the ESS team design a new cache that matched the new SMP capabilities. On the other hand, using the main memory of the SMP as the cache led to a partitioned cache. Each SMP has access to its main memory but not to that of the other SMP. Thus, internally, ESS would work better with two separate subsystems, each caching its own subset of the virtual disks and each attaching a subset of the physical devices.

Although the simpler environment is a shared cache, working to provide a new cache for each new ESS generation was redundant, given the pSeries development efforts. The pSeries already included a large volatile memory that was balanced with the rest of the technology in the SMP. Although a separate cache provides fast access, it cannot match the access speed of the SMP main memory. The decision to use the SMP main memory as the cache proved itself in three different generations of the product. The performance roughly doubled with each generation. This performance improvement can be traced to the capabilities of the completely integrated SMP, the processor speeds, the L1/L2 cache sizes and speeds, the memory bandwidth and response time, and PCI bus performance.

Adopting the main memory of the SMP as the read cache had many consequences. The two ESS nodes are nearly independent entities if one looks below the level of the host adapters. Each node supports its virtual disks and manages its physical disks. Although the nodes use the heartbeat mechanism and store data in the nonvolatile storage on the other node, these mechanisms affect system operation only in the event of node failures. The effects of workload imbalance between the two nodes was considered. Because of the amount of resource available in each node and the volume of work that would be going to each node in situations where the nodes might be stressed, our projections indicated that workload imbalance would not be a problem. Nearly

Figure 5 Future: ESS architecture with logical partitions




four years of experience with thousands of installed systems have proven this to be the case.

The split cache had consequences for failover and failback. Since the surviving node has no access to the cache of the failed node, it must store all virtual disks in its cache during the period of single node operation. The split cache also affected the decision that host adapters will be shared dynamically, as discussed next.

**Dynamically shared host adapters.** Shared host adapters were adopted for reasons of performance, functionality, reliability, availability, and serviceability. During the initial design we explored several methods for providing host attachment to the two nodes that make up an ESS system. The split cache decision had already been made. There were two opposing choices: (1) dedicated host adapters that attached to only one node, and (2) dynamically shared host adapters that attach to both nodes.

Providing dedicated host adapters represents a simpler design, but it has some undesirable consequences. Mirroring the write data would have required that the owning node receive the data from the host adapter and then forward it across the connection fabric to the partner node. The response time and the additional activity on the fabric made this a less attractive choice. Moreover, channel selection algorithms that rotate among the available paths from a host to a virtual disk would also have to be limited in order to avoid sending requests for a virtual disk to the non-owning node (such an alternate path is acceptable for error recovery but not for normal operation). Also, node failover would result in loss of one half of the server attachments, an unacceptable scope of failure. For fabric redundancy, an attaching host requires four paths to the ESS, two to each node. Dynamically shared host adapters mirror the write data by sending a copy to the read cache and a second copy to the nonvolatile storage. Thus, it is not necessary to "store and forward" the copy across the fabric. The dynamically shared host adapters also provide a level of switching which allows two paths between ESS and a host to provide full redundancy. This switching means that host attachments are able to be maintained on failover. Finally, all

paths can be used for both normal operation and error recovery.

**Both nodes active.** The decision to run with both nodes active was motivated by price/performance and therefore fairly obvious. Although the decision had a great impact on price/performance, it has not added significantly to ESS complexity.

**Device striping.** All ESS RAID offerings involve striping data and parity across the physical disks in the RAID array. Activity to a logical entity (virtual disk, data set, or file) thus tends to spread uniformly across the physical drives in the RAID. This distribution of requests means that accessing physical drives is faster and more efficient. Some RAID technologies (e.g., RAID 1 and RAID 4) do not stripe the data but rather map the virtual disks onto the physical storage viewed as a linear space. This may create increased contention if two or more particularly active virtual disks are mapped to the same physical device.

**Use of parallel SCSI devices in an SSA device fabric.** Whereas competitive products incorporate the latest technology on parallel SCSI devices before FCAL devices, ESS introduces the latest device technologies into the SSA fabric and thus delivers the most current device technology as early as possible. For the last two product generations, ESS has been delivered earlier than the two main competitive products.

## Future

The pSeries platform offers logical partition (LPAR) capability. LPAR enables an instance of an operating system (AIX, LINUX**, or OS/400*) to run on a full virtual machine that simulates the real hardware. LPAR also provides complete isolation between the many operating systems that may be running on a single SMP. This isolation means that one operating system does not interfere with the operation of another. One virtual machine could crash and reboot without impact on the other virtual machines running. IBM is looking to use the pSeries infrastructure to deliver LPAR capability within ESS. Storage-oriented middleware will then be able to run within ESS and provide an integrated, high-function solution (see Figure 5). For example, NAS gateways, Tivoli Storage Management, and Tivoli Storage Resource Management could be delivered in ESS LPARs. Over time, this capability could be enhanced with increasingly dynamic allocation of the resources available to the various LPARs and with virtualization of the communication paths between the LPARs.

ESS continues to deliver improved price-performance with every generation by incorporating state-of-the-art technologies from servers, communication fabrics, and devices. Its function is also expanding over time as new function is added to the embedded software. Both its modular design and its code portability lead to improved stability over time. ESS implements policy-based storage management as well as other storage management solutions. In addition, the ESS team works with middleware, application, and services providers to offer integrated solutions that can be rapidly deployed. This combination, enhanced by LPAR capability, could radically alter the landscape of storage offerings.



## Cited references and notes

1. These systems date back to the late 1960s, when an System/360 was utilized in the Apollo 8 mission to the moon. For a discussion on the early System/360, see C. J. Conti, D. H. Gibson, and S. H. Pitkowsky, "Structural aspects of the System/360 Model 85," *IBM Systems Journal* **7**, No. 1, 2–14 (1968).
2. For a more in-depth discussion on cache history, see D. A. Burton and B. McNutt, "Storage control cache resource management: Increasing diversity, Increasing effectiveness," *IBM Journal of Res. and Develop.* **40**, No. 3, 331–340 (1996).
3. D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *International Conference on Management of Data*, Chicago, IL, June 1988, ACM, New York (1988), pp. 109–116.
4. For a good discussion of autonomic computing, see A. G. Ganek and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Systems Journal* **42**, No. 1, 5–18 (2003).
5. G. A. Castets, D. Leplaideur, J. A. Bras, and J. Galang, *IBM Enterprise Storage Server*, SG24-5465-01, IBM Corporation (September 2001).
6. Symmetrix Networked Storage Systems, CLARiiON Networked Storage Systems, EMC Corporation, http://www.emc.com/products/platforms.jsp.
7. Global Storage, Hitachi Data Systems, http://www.hds.com/products/systems/.
8. LUN, or logical unit number, is the physical ID of a device in a SCSI chain of devices.
9. RAID 5 offers independent actuators with data and parity spread across all drives, while RAID 10, the result of RAID 1 + RAID 0, offers data striping across several drives that are mirrored by arrays of drives.
10. SSA provides for high-speed access to high-capacity disk storage. In the mid-1990s, SSA was IBM's proposed ANSI standard for a standard high-speed interface to disk clusters and

arrays. At that time, SSA allowed full-duplexed packet-multiplexed serial data transfers at rates of 20Mb/sec in each direction. For an in-depth discussion of SSA, see I. D. Judd, P. J. Murfet, and M. J. Palmer, "Serial Storage Architecture," *IBM Journal of Research and Development* **40**, No. 6, 591–602 (1996).
11. A. S. Meritt, J. A. Staubi, K. M. Trowell, G. Whistance, and H. M. Yudenfriend, "z/OS support for IBM TotalStorage Enterprise Storage Server," *IBM Systems Journal* **42**, No. 2, 280–301 (2003, this issue).
12. The parallelism also benefits responsiveness by eliminating much of the need for queuing of I/O requests to volumes. Additionally, the distribution of function amongst the ESS components (the host adapter and the SMP) further ensures the best possible response times.
13. C. Brooks, M. Bedernjak, I. Juran, and J. Merryman, *Disaster Recovery Strategies with Tivoli Storage Management*, SG24-6844-01, IBM Corporation (November 2002).
14. A. C. Azagury, M. E. Factor, and W. F. Micka, "Advanced functions for storage subsystems: Supporting continuous availability," *IBM Systems Journal* **42**, No. 2, 268–279 (2003, this issue).
15. L. L. Ashton, E. A. Baker, A. J. Bariska, E. M. Dawson, R. L. Ferziger, S. M. Kissinger, T. A. Menendez, S. Shyam, J. P. Strickland, D. K. Thompson, G. R. Wilcock, and M. W. Wood, "Two decades of policy-based storage management for the IBM mainframe computer," *IBM Systems Journal* **42**, No. 2, 302–321 (2003, this issue).
16. M. Kaczmarski, T. Jiang, and D. A. Pease, "Beyond backup toward storage management," *IBM Systems Journal* **42**, No. 2, 322–337 (2003, this issue).

*Accepted for publication March 28, 2003.*

**Mike Hartung** *IBM Corporation, 9000 S. Rita Road, Tucson, Arizona 85744 (phoenix1@us.ibm.com).* Mr. Hartung is one of the lead architects responsible for defining the IBM midrange and enterprise storage strategy and products. An expert in highly available, high performance storage subsystems, he has an extensive patent portfolio in this area. Mr. Hartung is an IBM Fellow and a Visiting Professor at the University of Arizona. He received an M.S. degree in Electrical Engineering from Stanford University in 1970.