Two decades of policy-based storage management for the IBM mainframe computer

by 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, M. W. Wood

Today, storage management vendors see the need and business opportunity for an enterprise-wide policy-based storage management solution for their customers. In the middle 1980s, IBM introduced the Data Facility Storage Management Subsystem (DFSMS) as a policy-based storage management solution for large mainframe computer systems. As an integral part of the operating systems OS/390® and z/OS™, DFSMS continues to be enhanced. This paper provides an overview of DFSMS and describes a few of its recent enhancements.

The Internet, e-business, business-to-business interactions, life sciences applications, and other new and extended uses of computing are creating an explosive demand for storage capacity. Using traditional procedures and tools to manage this storage growth requires an ever-increasing staff of storage administrators. More staff significantly increases the total cost of ownership of storage. Storage hardware and software vendors are turning to policy-based management technology to address these storage management needs and opportunities.

Large mainframe computer systems underwent rapid growth in storage and the accompanying escalating cost of storage ownership in the early 1980s. The IBM response for customers to this requirement was a policy-based storage management offering named the Data Facility Storage Management Subsystem (DFSMS). Today, DFSMS is a set of components that are integrated within OS/390* (Operating System/390) and z/OS* (the zSeries* Operating System). The four major components of DFSMS are: DFSMSdfp* (Data Facilities Product), DFSMShsm* (Hierarchical Storage Manager), DFSMSdss* (Data Set Services), and DFSMSrmm* (Removable Media Manager). These components and other terminology are defined in the Appendix.

The z/OS data, storage, and input/output architecture and the related terminology differ from the corresponding items in the UNIX** operating system and other nonmainframe operating systems. Here is a brief explanation of a few basic z/OS terms that are used in this paper: A z/OS data set is analogous to a UNIX file. A z/OS volume is a logical container for data. Data sets are stored in volumes. The data set to volume mapping is not one-to-one. Many data sets may be stored on a single volume. Parts of a single data set may be stored on multiple volumes. The content of a volume is stored or recorded on a storage device. There is a unique volume type for each unique device type.

For disk devices, z/OS views the space and content of a disk volume as mapping one-to-one to the space and content on the corresponding disk device type. When DFSMS was initially implemented, its view of a disk volume and disk device matched the actual data layout on the corresponding physical disk device. Through the years, disk and disk-storage-controller technology have evolved. However, z/OS con-

 0018-8670/03/$5.00 © 2003 IBM

tinues to use disk device type definitions that map one-to-one to z/OS disk volumes. Today's storage controllers support the z/OS disk device types and internally map them onto physical disks.

Prior to DFSMS, the user was directly involved in the data-set-to-volume placement. Since a volume mapped to a physical device, the user was required to choose a device type that met the requirements of the data. A fundamental principle that DFSMS introduced was the separation[1] of the logical view of data from the physical device characteristics. The DFSMS policy definitions based on this principle and the corresponding DFSMS policy-based storage management functions yielded a breakthrough reduction in the total cost of ownership of mainframe storage.

The next section in this paper is an overview of DFSMS policy-based storage management. The overview is followed by descriptions of the following DFSMS enhancements: DFSMShsm reclamation of tape storage media, DFSMShsm automatic reconnection of recalled data sets, DFSMShsm common recall queue, DFSMSdfp VSAM (Virtual Storage Access Method) record-level sharing, and DFSMSdfp data striping. It goes on to describe DFSMS support for business continuance and DFSMS disk copy services. Then follows a discussion of the various levels of DFSMS testing.

Overview of DFSMS policy-based storage management

The concept of policy-based storage management involves defining policies that allow the system to take over many storage management tasks that were previously performed manually.

DFSMS separates the logical view of data from the physical view of data. The logical view of data is concerned with what the data look like and what services the data require. The physical view is concerned with where the data actually reside. The policy types that specify the logical view of data are: data class, storage class, and management class. Storage group is the single policy type for specifying physical storage. An aggregate-group policy specifies a grouping of data for purposes of backup and recovery in case of a disaster. An additional policy called the base configuration is used to specify a system-wide set of default storage information. Following is a brief description of each of these policy types.

A *data class* policy specifies allocation[2] defaults for data. It supplies such information as space parameters and data attributes.

A *storage class* policy defines performance and the availability requirements of the data. It supplies attributes used for dynamic cache management, sequential data striping, and concurrent copy.

A *management class* policy supplies data migration, backup, and expiration and retention values. It provides automatic storage management and availability management capabilities.

An *aggregate-group* policy provides control information and data set lists to define an application level or other grouping of data. It contains lists of data sets and backup criteria that are used as input to DFSMShsm aggregate backup and recovery support (ABARS).

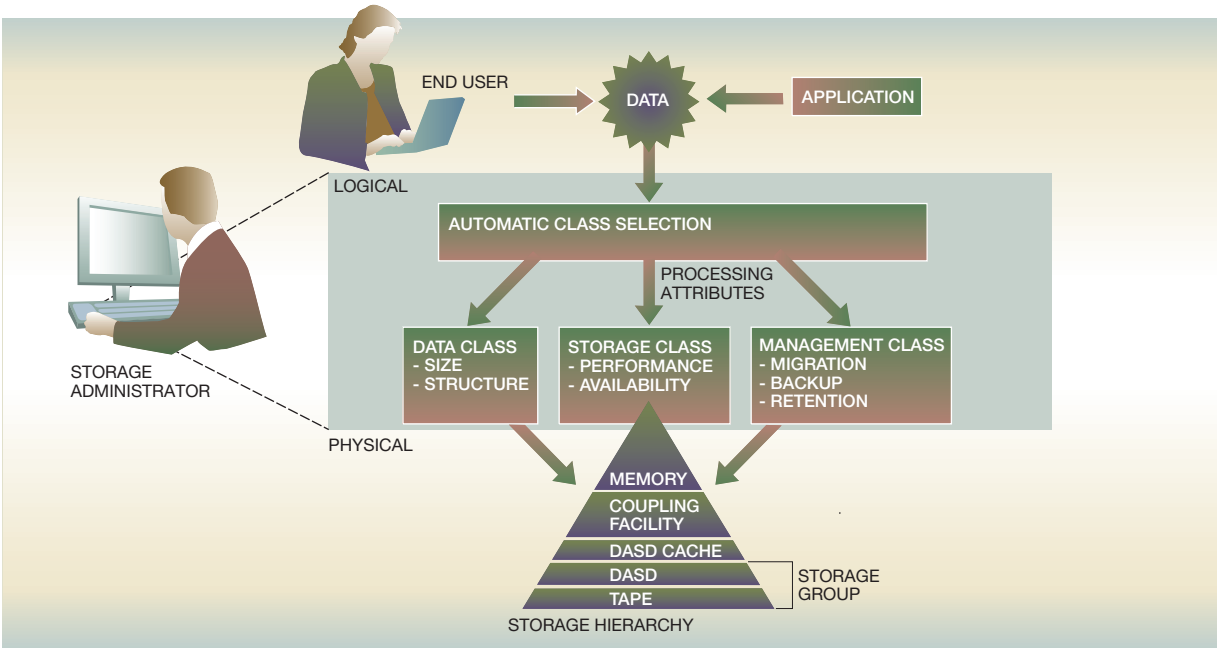
A *storage group* policy specifies the physical storage on which data reside. A storage group is a collection of disk volumes or a collection of tapes within a system-managed tape library. It allows a collection of storage devices to be pooled and managed collectively. It can also be used to define whether or not automatic migration, backup, and dump are allowed within the pool.

There is one *base configuration* policy per DFSMS configuration. It identifies the set of systems to which the configuration applies and contains a set of installation defaults.

An *automatic class selection*, or ACS, routine is a sequence of instructions through which the system associates DFSMS policies (classes and groups) with data sets. The selection of specific classes and groups is based on information from job control language specifications (used to describe the input/output requirements of a job) and other allocation parameters. ACS routines may use parameters, such as data set name, volume serial number, job name, and data set size, to assign policy instances to data sets.

There is an ACS routine for each policy type: data class, storage class, management class, and storage group. The ACS routines are invoked whenever data sets are created, either at initial allocation or as a result of operations that cause them to be reallocated or moved. As shown in Figure 1, the role of the ACS routines is to assign specific policies to the data set that is providing centralized administrator control over data set allocation. Once a DFSMS configuration has been created, it can be used as the set of policies that automate storage management across a z/OS sysplex. The next three subsections are brief

Figure 1 DFSMS policy-based storage management



descriptions of DFSMS policy-based disk volume selection, space management, and availability management.

Disk volume selection. DFSMS automates the function of selecting the specific disk volumes used for space allocation of a data set. DFSMS data set allocation goes through a number of steps to select a volume from the list of candidate disk volumes.[3] The list of candidates is made up of all volumes in all storage groups that were provided by the ACS storage group routine. Each candidate volume is placed on a primary list or a secondary list, or on both.

The primary list contains volumes that meet all criteria of the storage class and data class specifications of the data set and have sufficient space so that they will be below their high threshold after the data set is allocated. The secondary list contains volumes that do not meet all the criteria met by the primary list. All volumes that are on the primary list are also on the secondary list.

If the primary list of volumes is not empty, it is sent to the OS/390 or z/OS System Resource Manager (SRM) component to select a volume from the list. SRM prefers volumes that are not already allocated to a job or subsystem and that have the least I/O delay at the time of allocation. If there are no volumes on the primary list or if the system is unable to allocate space on any of the volumes from the primary list, DFSMS will attempt to select a volume from the secondary list. During this stage of volume selection, DFSMS does not make use of SRM. DFSMS groups the volumes into bands where the top band contains the most preferred volumes and the lowest band contains the least preferred volumes. This banding is done on the basis of the relative importance of different attributes. The volumes in the most preferred band satisfy user-required attributes to a higher degree than volumes in lower bands. After this grouping is done, a volume from the highest band is randomly selected.

Volume selection is followed by volume allocation, space allocation, and cataloging of the data set. If any of these subsequent processes fail, volume selection is repeated again and again until all candidates have been exhausted, at which point the allocation fails.

Space management. The DFSMS policies specify service-level objectives for available disk space. The goal is to avoid application failures caused by out-of-space

conditions. DFSMShsm is the DFSMS component that provides policy-based space management functions. Using storage group and management class attributes (policies), it automates space management at the data set level.

The DFSMShsm space management algorithms attempt to maintain free space within the disk storage groups. The primary method used to keep disk space occupancy within policy-specified thresholds is to migrate data sets from the DFSMS storage group disk space into a separate DFSMShsm-owned disk-tape storage hierarchy. The data sets to migrate are selected based on the policy-specified time period since last reference. The migration process is automatically invoked on a periodic basis. Although the data have been moved, the name of a migrated data set remains cataloged. If a migrated data set is subsequently referenced, DFSMShsm automatically recalls the data set into a disk storage group ready for application access.

In addition to data set migration, DFSMShsm frees disk space within storage groups by deleting expired data sets, deleting temporary data sets unintentionally left at the end of a program, releasing unused space allocated to individual data sets, and when permitted by policy, reblocking data during data set recall and recovery processing.

Other space management functions available in DFSMS include the release of over-allocated space when a data set is closed and defragmentation of space on disk volumes. The first may be specified as a management class attribute and is provided by DFSMSdfp; the second is provided by DFSMSdss. Defragmentation is used to consolidate free space on disk volumes.

Availability management. Availability management maintains recent backup copies of disk data sets. The purpose is to allow lost or damaged data sets to be restored to the most current backup level.

Using storage group and management class policy specifications, DFSMShsm provides automatic and periodic data backup. Both volume-level and data-set-level backup are provided. Volume-level backup expedites recovery when an entire volume of data is lost or damaged. DFSMShsm also supports command-initiated backup with DFSMS policies applied to the backup.

One function of DFSMShsm allows backup versions of the data to expire. This function is invoked manually to clean up excess or unwanted backup versions. Backup versions are deleted when the specified number of versions to keep has been exceeded, when a data set has been deleted, or when an authorized user issues a command to delete unwanted versions.

DFSMS provides an aggregate backup and recovery function. This function creates backup copies of a user-defined group (aggregate group) of data sets for recovery at another site or at the same site. Complete entities, such as applications, can be recovered in the event of a disaster.

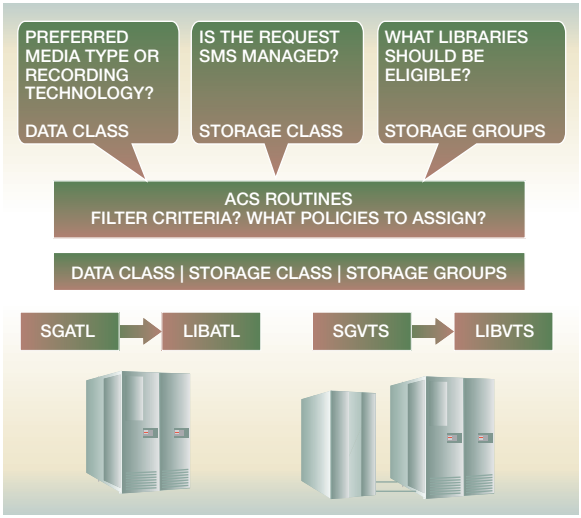
DFSMSdss is the DFSMS component that provides volume-level and data-set-level copy services. It exploits the advanced copy functions of disk storage subsystems. DFSMSdss works in conjunction with DFSMShsm to provide availability management functions.

Advantages of DFSMS policy-based storage management. Without policy-based management, storage-management-related parameters must be set individually for each volume. This operation requires the user to carefully place the data on volumes that satisfy requirements of the data sets. The user, or customer, must be aware of the functions and capabilities of physical device volumes. As the amount of data and storage grows, the customer cost of device-level and device-dependent management greatly increases. At some point, the cost becomes a serious inhibitor to growth. This increasing cost is the specific problem that DFSMS policy-based storage management addresses.

DFSMS policy-based storage management separates space, availability, and performance requirements for data sets from the physical device volumes on which they reside. There is no need to issue special commands to set the necessary parameters; all information is contained in the DFSMS policy configuration. DFSMS performs automatic space, availability, and performance management based on the data-set-level policies. Since data are managed based on data-set-level policies, data sets with different expiration, migration, and availability management requirements can coexist on the same volumes.

Policy-based storage management in a tape library environment. With the introduction of the IBM 3495 Tape Library Dataserver in 1993, DFSMS policy-based storage management was enhanced to include support for the IBM automated tape library dataserver. This support has been extended to include the IBM

Figure 2 System-managed tape



3494 TotalStorage* Enterprise Automated Tape Library, the IBM TotalStorage Virtual Tape Server (VTS), the IBM TotalStorage Peer-to-Peer Virtual Tape Server (PtP VTS), and the manual tape library.

Using the concepts of DFSMS policy-based storage management, the storage administrator can better manage the tape volumes and tape libraries in their environment. Each tape library consists of a set of tape volumes and the devices on which the volumes can be mounted. A tape library can be automated, with robotic devices mounting and demounting the tape volumes, or it can be manual, with a tape operator doing the mounting and demounting. A tape library can also be virtual such as the VTS or the PtP VTS with the library emulating virtual tape volumes and virtual tape devices. Each tape library can also contain heterogeneous media types and device types, all managed by the supporting DFSMS software and policy-based storage management.

The system-managed tape support manages the tape volumes but not the actual tape data sets. DFSMSrmm can be used to manage the tape data sets. When a volume is entered into a tape library, it can be associated with a storage group policy. Each tape library then contains volumes associated with one or more tape storage groups. As the first data set on a volume is allocated, the storage administrator, on the basis of filter criteria such as data set name, can direct through the ACS routines, for example, a storage class policy that indicates that the request is to be managed by SMS (the Storage Management Subsystem) and have one or more storage group policies, an optional management class policy, and an optional data class policy. The assignment of one or more tape storage groups then directs the request to one or more tape libraries. Optionally, a data class can be specified, which directs the allocation request to a particular type of media and recording technology, thus easily integrating new device technologies. This can be important if a customer's tape environment has heterogeneous devices in the customer's library environment and there is a need to direct a request to a particular media and device type. On the basis of the device that is allocated, the volume is then assigned to a tape storage group associated with that tape library. Requests to existing data sets are then allocated to devices within each associated tape library.

With system-managed tape and policy-based management, the storage administrator can easily direct or redirect the data of an application and be assured that the requested media and device type are allocated. The data may be directed to:

- A particular library type (automated, virtual, or manual)
- A set of one or more libraries, on site or at a remote location
- A specific device type within a set of one or more libraries
- A specific media type supported by the device

Figure 2 shows how policy-based management and storage group assignment can be used to direct the data set request to a particular type of tape library (automated or virtual).

The storage class policy can also be used to help the VTS better manage its tape volume cache. For instance, an initial access response time can be specified to let the library know that the data being written can be removed from cache sooner. This directive lets more critical customer data stay in cache longer for faster access (mount times). It may be particularly important if the volumes being written contain customer data that may never be accessed again (for instance, backup data).

Policy-based storage management has recently been extended to the library so that additional policy actions can be defined outboard (at the library itself). Again driven through the customer's ACS routines,

if the library supports outboard policy management, the assigned policy names (storage class, storage group, management class, and data class) will be sent to the library. Then at the library, policy actions can be defined and associated with those policy names. In this way the VTS and PtP VTS can enable policy-based storage management functions at the library analogous to functions that have been provided through host applications. For instance, one of these policy actions might indicate to the VTS library that the logical volume must be copied. With each VTS library containing more and more customer data and more and more tape volumes, it is increasingly important to enable customers to better manage their logical volumes and data. Since customers are familiar with policy-based storage management at the host, applying this concept to the library was a logical extension. It enables the same set of policy names to control both host and outboard policy actions.

DFSMSrmm policy-based removable media management. DFSMSrmm is the DFSMS component that records information about tape data sets and volumes and provides policy management for those data sets and volumes. Tape data set and volume information is made available by DFSMSdfp during data set creation and subsequent processing. Using management class and storage group names, and the attributes in its own retention and movement policy definitions, also referred to as vital record specifications (VRSs), DFSMSrmm automates retention and movement for all tape data that it manages.

All new tape data sets are assigned default retention values. Optionally, a management class or VRS management value, or an override of the user-specified retention value can be assigned. For new data sets on system-managed tape volumes, DFSMSrmm can use the management class assigned through the ACS routines to select a VRS that contains the attributes used to manage the data set. Policies for tape data sets (removable media) are quite different from those required for disk data sets. Besides supporting movement through multiple libraries and storage locations, retention and movement decisions can be taken based on many different factors such as number of data set generations, days since last moved or created, and even catalog status. Complex policies can be created by combining different retention and movement decisions. For new data sets on nonsystem-managed volumes, DFSMSrmm also calls the ACS routines to enable a management class to be assigned.

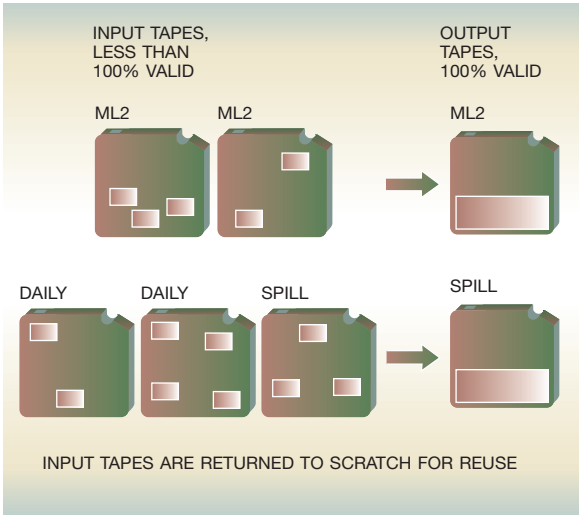
Policies can be defined for data sets and volumes. Usually they are defined for data sets, and DFSMSrmm manages the volumes based on the needs of all the data sets on that volume. A volume is retained until all the data sets on the volume have expired. In the case of virtual tape, DFSMSrmm uses retention information for all logical volumes on a physical stacked volume to determine where the exported stacked volume should be stored and retained.

DFSMSrmm management functions are performed by running utilities that include:

- Vital record selection: Each data set on a private tape volume is matched to the DFSMSrmm policies, and the retention and movement requirements are calculated. Each time this process is run, the policies are matched again, and any changes to policies or their attributes are implemented.
- Storage location assignment: Based on the results of the policy requirements calculation, the intended storage location for the volumes is determined, and movement of volumes is triggered. Volumes no longer retained by policy are returned to their home location.
- Expiration: Data sets no longer retained by policies are eligible for expiration. Expiration is at the volume level once all data sets on the volume have expired. Normally the intention of expiration processing is to return tape volumes to the "scratch" pool ready for reuse. However, DFSMSrmm also provides controls that enable volumes to be removed from the library or replaced. Before volumes are returned to the scratch pool, they go through a release process during which any required actions must be taken, including: relabeling, replacing the volume, owner notification, and data erase.

DFSMSrmm also manages the tape scratch pool. Policies can be defined that control how the scratch volumes are divided into pools and who can create new tape data in each of the pools. Allocation of new data sets to those pools can also be controlled by using the storage group ACS routine. For nonsystem-managed tape volumes, DFSMSrmm also calls the ACS routines to enable a tape storage group to be assigned and ensures that a volume from the correct scratch pool is mounted. Mount messages for scratch tapes can be modified by DFSMSrmm with the required scratch pool information. For system-managed tape volumes in an automated tape library, the library selects an appropriate scratch volume based on the media type requested with DFSMSrmm, ensuring that the volume selected is a scratch volume.

Figure 3 Recycle—consolidate valid data



DFSMSrmm extends system-managed tape support from volumes to data sets. By using the installation exits that are available at key processing points: cartridge entry, cartridge eject, change use attribute (scratch pool and private transitions), and volume-not-in-library processing, DFSMSrmm tracks key volume and data set information and ensures that correct values are available when required. When DFSMS policies are assigned to data sets, the policy names are recorded and can later be provided or changed by DFSMSrmm commands so that new policy names can be assigned and implemented at the library.

By directly calling the ACS routines to determine a storage group and management class assignment, DFSMSrmm can use storage group and management class for all tape volumes and tape data sets regardless of whether the volumes are system-managed.

DFSMShsm reclamation of tape storage media

Customers use management class policies to define to DFSMShsm how and when to back up and migrate data sets, when migrated data sets should expire, and how many backup versions to keep. It is also through these same management class policies that migration and backup versions are directed to tape, freeing disk space. DFSMShsm creates several types of tapes, among them migration tapes that hold migrated data sets and backup tapes that hold backup versions.

When the migration and backup tapes are first created, user data sets are stacked onto one or more tapes, fully utilizing the tape media. All data sets on those tapes are considered valid. As migrated data sets are recalled to disk or expire and as excess backup versions are "rolled-off," these valid tape data sets become invalid. A tape that once consisted of 100 percent valid DFSMShsm data now consists of both valid and invalid data.

As shown in Figure 3, reclamation, or recycle, is the process of moving valid migrated data sets or backup versions from multiple nonfull tapes and consolidating the data on one or more tapes, again, creating a 100 percent valid tape. The resulting empty tapes are returned to the scratch pool.

The recycle process implemented by DFSMShsm addresses three important problems associated with common reclamation techniques: prioritization of work, efficient use of tape media, and optimization of tape allocations.

Prioritization of work. In order to return tapes to the scratch pool as quickly as possible, three schemes are used: customer input, prioritizing the tapes for recycling from least full to most full, and keeping tasks busy recycling "good" candidates as soon as they are available.

Customers are given two options that allow them to limit the number of tapes that are chosen to be recycled: PERCENTVALID and LIMIT. PERCENTVALID tells DFSMShsm to recycle only those tapes whose percentage of valid data is less than or equal to the PERCENTVALID. This keeps DFSMShsm from recycling tapes with a high percentage of valid data. LIMIT tells DFSMShsm to recycle tapes until the specified number of tapes have been returned to the scratch pool,[4] thus allowing DFSMShsm to quit recycling tapes after that number has been returned to the scratch pool.

The DFSMShsm recycle process prioritizes the order of tapes to recycle based on the amount of valid data on a tape, optimizing the number of tapes returned to the scratch pool based on the least amount of data moved from the input tapes to the output tapes. Those tapes with the least amount of valid data are recycled first. Both PERCENTVALID and LIMIT use this prioritization.

Instead of delaying the movement of data until the percentage of valid data for each volume is known, the recycle process begins with "good" candidates, even if better candidates are found later. This method allows tapes with little valid data, say 10 percent, to be considered as good candidates and recycled while the total amount of work is still undetermined. The criteria for a tape to be considered "good" changes, based on making sure that data movement tasks are always kept busy with input tapes to recycle. If there are no current good candidates, then the recycle process will increase the percentage of valid data when considering whether or not a tape is a good candidate. If there are too many good candidates, then the recycle process decreases the percentage of valid data. After all the tapes have been analyzed, the list of work is sorted in ascending order based on the percentage of valid data.

Efficient use of tape media. When DFSMShsm determines how much valid data are on a tape, it takes into consideration how much valid data could be written to the tape given the current tape technology of an installation. For instance, if a standard capacity tape cartridge had been written by an IBM TotalStorage Enterprise Tape Drive 3590 Model B subsystem, it could contain 10 gigabytes (GB) of data. The same standard capacity tape cartridge written by a 3590 Model E subsystem can contain 20 GB of data.[5] The amount of data that a rewritten cartridge can contain is called the reuse capacity.

As an example, if a cartridge written by a 3590 Model B subsystem contains 70 percent valid data and the cartridge were to be rewritten with the same valid data by a 3590 Model E subsystem, it would only contain 35 percent valid data. At 70 percent valid data, the cartridge is not a good candidate for recycling, but at 35 percent it is.

DFSMShsm knows what tape technology is currently in use and calculates the percent of valid data on a tape cartridge based on the reuse capacity of that tape cartridge given the current hardware environment.

The recycle options, INCLUDE and EXCLUDE, are provided, allowing an installation to quickly move from one tape technology to another.

Optimization of tape allocations. Tape allocations are expensive in terms of overhead from a software perspective. Additionally, in a manual tape environment, deallocating and allocating each tape to be recycled puts a burden of work on the tape librarian. The recycle function has optimized tape allocations by considering the following: no tape allocations are required for empty tapes, a tape allocation is reused for subsequent input tapes, and the priority of which tapes to recycle takes into account the input device type required.

Although a tape allocation is not needed to recycle a tape with zero percent valid data, in the case where a recycle command is entered requesting a PERCENTVALID greater than zero percent, tape devices will always be allocated, anticipating the recycling of those tapes that contain some valid data. However, if customers need to return tapes to the scratch pool quickly, or have no spare tape devices, they can specify PERCENTVALID(0). The recycle function will analyze its tape inventory and recycle those tapes that no longer have any valid data, returning them all to the scratch pool without a single tape allocation or tape mount.

When a tape device is initially allocated, the volume to be mounted on that device is included in the allocation request. However, there will most likely be hundreds of additional volumes to be mounted and recycled. Rather than deallocating the device and then reallocating with the mounting of a new input volume, DFSMShsm enables subsequent tape volumes to use the same allocated device.

Finally, the tapes to be recycled may need to be mounted on different device types. Even though the list of volumes to recycle is sorted based on the percent of valid data, the volume at the head of the list is not necessarily always chosen for recycling. The list is scanned to seek volumes to be recycled on the tape devices that are currently allocated. If no other tape can be recycled on the current device, the device is deallocated, and the first tape volume in the list is selected.

The recycle process is a time-consuming operation. By taking into account technology currently in use, prioritizing the workload, and making efficient use of the resources, DFSMShsm has been able to reduce the time necessary to reclaim space on tape media, returning empty tapes to the scratch pool.

DFSMShsm automatic reconnection of recalled data sets

The DFSMS management class policies include the specifications for data migration. DFSMShsm performs

the data migration based on the policies. It also performs the recall function if and when the migrated data are subsequently referenced.

The space management function of DFSMShsm reduces the cost of electronic data storage by ensuring that higher-cost per gigabyte devices, such as disk, are not occupied by data that are unneeded or that have not been referenced for a significant period of time. Obsolete data are deleted and inactive data are moved, or migrated, to lower-cost devices, such as tape. Migrating the data to tape allows the data to be automatically retrieved, or recalled, should the data be required at some future time, yet frees space on the disk device for other data sets. Although the deletion of data is nearly instantaneous, movement of data both consumes time and utilizes often scarce resources. Automatic reconnection of recalled data sets can be performed in approximately the same amount of time as data set deletion, yet allows the subsequent use of those data.

Normal migration and recall. When a data set is migrated to tape, its contents are copied to a tape, meta-data representing the migrated version are written to the DFSMShsm inventory files, and the original data set is deleted from disk. When the data set is recalled from tape, its contents are copied back to disk, and the meta-data are updated to indicate that the migrated version is no longer valid. The contents of the data set remain on the tape but are made inaccessible by the meta-data update, and the space they occupy is not made available for reuse until the entire tape is returned to the scratch pool. Standard remigration of the data set involves copying the data to a new location on the same tape or on a different tape, updating the meta-data to point to that new location, and deleting the original data set from disk.

Reconnection. Reconnection of a data set that has been recalled from tape, also known as Fast Subsequent Migration, is performed without copying the contents of the data set. Instead, the meta-data for that data set are updated to point to the old location of the meta-data on the migration tape, and the original data set is deleted from disk. The absence of data movement and the resulting reduction in tape mounts lead to significant savings both in time and system resources. A less obvious benefit to Fast Subsequent Migration is derived from the revalidation of data on the migration tape that was invalidated during the recall operation. This benefit reduces the need for the tape to be recycled, because the amount of valid data on it has been increased.

Clearly, reconnection to the old copy of the data set on the migration tape should be done only if the data set has not been changed since it was recalled. Information from the meta-data, the table of contents of the disk volume, and the system catalog is compared to verify that the data set has not changed. If the data set has been backed up since its recall, it is assumed that it has changed and will not be permitted to reconnect. In addition, recycling the migration tape on which the old migrated version resides prevents the reconnection of the data set as the existence of the version cannot be guaranteed.

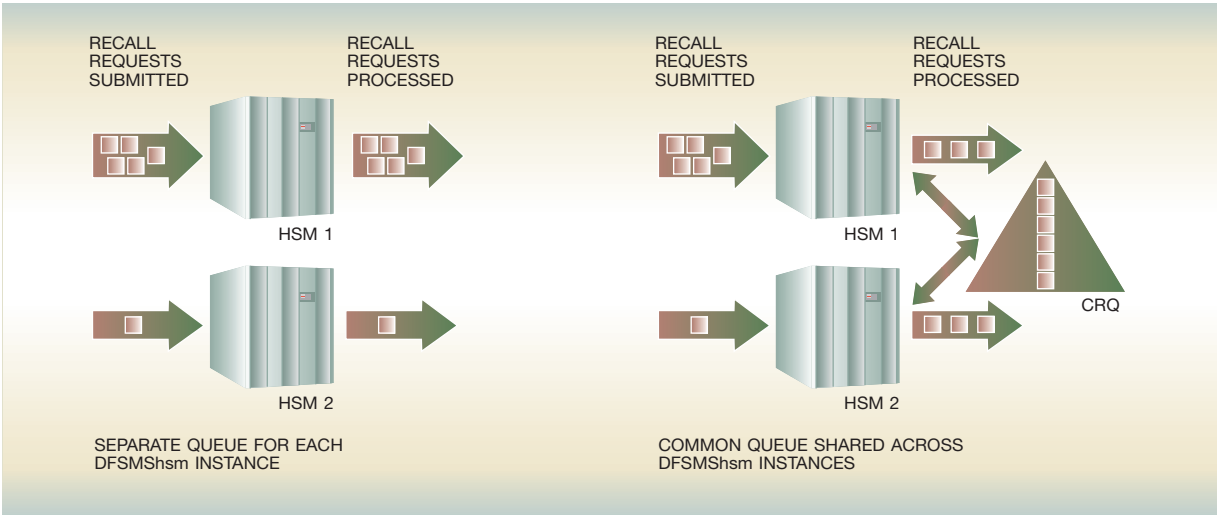
The ratio of data sets eligible for reconnection to those not eligible for reconnection will vary widely from customer to customer, based on the usage pattern of their data. It is essential that the overhead involved in performing the eligibility checks be minimized so that customers with small numbers of reconnectable data sets will not be penalized by these checks. To the extent possible, the eligibility checks for Fast Subsequent Migration are performed in conjunction with those done for standard migration. Storage administrator controls and an installation-wide exit provide the ability to customize the reconnection process to meet the individual needs of a customer.

Fast Subsequent Migration provides a quick means of freeing up space on disk while preserving the ability to automatically recall the data should the data be needed in the future. Eligibility checking is performed in such a way as to maintain data integrity, but generate little additional overhead. When enabled by the storage administrator, the function is performed without the need for user intervention.

DFSMShsm common recall queue

An important factor in the operational efficiency of z/OS computing environments that implement a DFSMS space management process is the performance of the DFSMShsm recall function. When an application references data that have been migrated to DFSMShsm-owned storage, that application must wait for DFSMShsm to recall the data back to primary disk storage. Minimizing this wait time by optimizing the throughput of the recall function aids in maximizing the efficiency of applications that reference migrated data. DFSMShsm provides this optimization through utilization of cross-system coupling facility (XCF) technology to provide a common recall work queue as depicted in Figure 4.

Figure 4 DFSMShsm common recall queue



A typical z/OS Parallel Sysplex* configuration consists of a single DFSMShsm host per z/OS image. When an application that is processing on a particular z/OS image references data that have been migrated, a request to recall the data is queued to the DFSMShsm host running on that same image. Each DFSMShsm host processes all of the recall requests on its unique queue without regard to the other concurrent recall activity that is occurring on other DFSMShsm hosts throughout the sysplex. This narrow perspective prevents sysplex-wide optimization of the recall function. Implementing a recall queue shared by all hosts in the sysplex overcomes this deficiency by enabling sysplex-wide workload balancing, priority optimization, efficient utilization of tape resources, and greater flexibility with sysplex configurations.

Workload balancing. The most significant feature of a common recall queue is that it enables the recall workload to be balanced across the entire sysplex. In a nonshared queue configuration, recall requests are assigned to a DFSMShsm host without regard to the ability or capacity of that host to process the assigned workload. The result is an unbalanced system in which one or more hosts may have more recall requests than they can individually process concurrently, while other hosts have unused request processing capacity. By placing all recall requests onto a common queue, workload distribution is optimized by only assigning work to a system that has the capacity and ability to perform that work. This technique enables all recall resources in the sysplex to operate at their peak task levels.

Priority optimization. One of the features of the DFSMShsm recall function is the ability to assign a processing priority to each request. This priority is used to determine the order in which each request should be processed with respect to other outstanding requests. A fundamental use of this priority scheme is to distinguish synchronous requests from asynchronous requests. All synchronous requests, those requests for which an application is waiting for the recall to complete before continuing, are selected for processing before asynchronous requests are selected. Placing all requests in the sysplex onto a single queue enables all synchronous requests to be selected for processing before nonsynchronous requests. This method is an improvement over a nonshared queue environment in which one DFSMShsm host may be processing asynchronous requests while synchronous requests are waiting to be selected on the queue of another host.

Tape efficiencies. When tape is used to store migrated data, a probability exists that concurrent recall requests will require data that have migrated to the same physical tape. The probability of this scenario increases as the storage capacity of tape increases with advancing technologies. In a nonshared queue environment, concurrent requests requiring the same tape may be queued to different DFSMShsm

hosts. When this occurs, the first host to allocate and mount the tape will be able to process its requests, whereas the remaining hosts will have to wait until the tape becomes available. A common queue configuration overcomes this wait by enabling the first host to process all the outstanding requests requiring that tape with a single tape mount, without regard to the system on which the request originated.

Flexible configurations. A common recall queue enables system configurations that would not otherwise be possible. Two such configurations are environments with recall servers and environments in which not all systems are connected to the tape subsystem.

Recall servers. In a nonshared queue environment, there must be one and only one DFSMShsm address space per z/OS image that processes the recall requests that are initiated on each image. The common recall queue overcomes this restrictive environment by enabling multiple DFSMShsm address spaces on the same image to process recall requests and by not requiring each DFSMShsm host to process recall requests. It may be advantageous to not have a particular DFSMShsm host process recalls if the dynamic nature of the recall workload would have a negative impact on mission-critical applications processing on the same image. DFSMShsm hosts whose primary responsibility is to process recall requests are referred to as recall servers.

Tape subsystems. In certain environments, it may not be possible or desirable to connect all z/OS images to a tape subsystem. Without the common recall queue, applications on these images would not be able to reference data that had been migrated to tape. The common recall queue enables the DFSMShsm hosts on these images to be configured such that they place all of their requests onto the common queue but only select those requests that do not require tape. Thus, only those DFSMShsm hosts residing on images that are connected to the tape subsystem will process recall requests that require tape.

DFSMSdfp VSAM record-level sharing

The Virtual Storage Access Method (VSAM) record level-sharing (RLS) function was first introduced and shipped as a subcomponent of DFSMSdfp in 1996. The design intent of the RLS function was to expand on the DFSMS policy-based storage management implementation in order to address data sharing and recovery-related issues with existing VSAM data sets in a Parallel Sysplex.

Prior to VSAM RLS, applications or middleware (e.g., CICS*, the Customer Information Control System) accessing VSAM data sets were responsible for serializing the sharing of these data sets between the different system or application images within the sysplex. In addition to providing the necessary serialization, applications were also responsible for recovery scenarios in the event of an application or system failure. The existing VSAM method of access provided with DFSMS did little to contribute to the sharing or recovery of the files, other than to provide some file protection if limited access was required.

The requirement put forth to VSAM RLS was to take on responsibility for serializing access to the VSAM data sets and to play a role in providing data integrity and availability in the event of a system or application failure.

VSAM RLS addressed the sharing and recovery requirements by providing full read and write integrity to any number of applications (users) in the Parallel Sysplex through the use of the XCF function in OS/390 and z/OS. Additionally, VSAM RLS provides the CICS Transaction Server (CICS/TS), one of the largest VSAM users today, with a set of services to protect individual data records involved in a failure from access by sharing users. By retaining serialization for the "failed" data records, VSAM RLS enables CICS/TS to recover from the event while allowing sharing users access to the remaining data records in the data set.

The XCF function of OS/390 and z/OS offers a choice of allocating one or more specialized data structures in shared high-speed storage, known as a coupling facility (CF). The data structures, referred to as cache, lock, and list structures, can then be used for storing and retrieving information between the various systems within the sysplex.

VSAM RLS implemented two of the three XCF data structure types, namely the cache and lock structures, for serializing access between the shared VSAM files. The cache structure, which is associated with a particular VSAM data set via the DFSMS storage class policy of the data set, contains the latest copy of the records of the data set. The cache structure is allocated in the CF when a data set is first opened. Data blocks are then read into the cache from disk as needed. Local copies of the data blocks are main-

tained by each system; however, the XCF cross-invalidate function is used to inform individual systems whether a particular data block has been changed.

The XCF lock structure is allocated by VSAM RLS during system initialization by a set of routines collectively known as SMSVSAM. The lock structure is composed of two parts, the lock table and the record data table. The lock table maintains "record locks" for each of the individual data records currently referenced by SMSVSAM. The record locks may be held exclusively or shared by the user, depending on the type of activity performed on the data record. For example, an update of a data record will lock the record as being exclusive, prohibiting sharing users from accessing the record, whereas a read of the data record will lock the record as being shared, enabling sharing users to have simultaneous access to the record.

The record data table, in contrast, contains additional information about the individual record locks, examples of which include whether or not the record lock was involved in a failure (retained lock) and the name of the owning application of the record lock. By associating the retained record locks in the record data table with the CICS/TS owning instance, VSAM RLS can prohibit access to the specific records by other users who share the data set. The CICS/TS instance that originally held the record locks can then reobtain the locks and render a decision as to whether the original updates to the data records should be backed out or not.

Exploitation of the XCF cache and lock structures by VSAM RLS relieves VSAM users of the difficult burden of serializing access to their shared data and places the responsibility where it belongs: in the hands of the access method. Additionally, the ability of VSAM RLS to associate XCF cache structures to individual VSAM files via the DFSMS Storage Class policy provides its users with the necessary automation for managing their ever-expanding volume of data.

DFSMSdfp and data striping

Data striping is a storage accessing performance enhancement technique. Rather than place a file or data set on a single disk, the data are spread, or striped, across multiple disks. This technique enables the combined I/O rate and bandwidth of multiple disks to be used when the data are accessed. This technique can significantly reduce the elapsed I/O time for sequentially accessing large portions of large files or data sets. It can also support a higher random I/O rate.

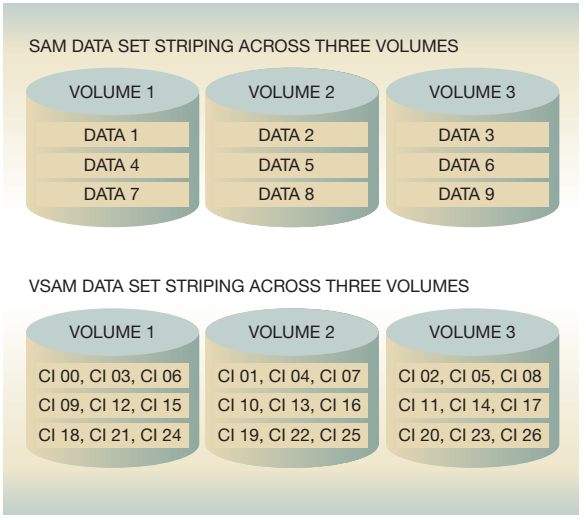
Data striping is supported as an optional feature of VSAM and SAM (Sequential Access Method) data sets that have the extended format attribute. The DFSMS storage class policy sustained data rate attribute of the data set and the number of volumes that are assigned to the data set determine the number of stripes. When performing sequential access to a striped data set, the objective is to perform I/O operations concurrently against each stripe[6] of the data set. The greater the number of stripes, the higher the sustainable sequential data rate.

Both SAM and VSAM use the same I/O driver for accessing striped data sets. Both access methods provide the buffering algorithms for direct or sequential processing and the information used by the I/O driver to determine how to locate and process the data in the data set. Part of this information includes information about the number of stripes used and the location of the data on the stripes. The I/O driver builds channel programs to access the disks and submits the channel programs to the z/OS I/O supervisor. For sequential access, a separate channel program is created to access each stripe, and the channel programs are run concurrently.

Striped VSAM data sets. For a VSAM data set, the data-to-stripe mapping is done at the granularity of a control interval (CI).[7] VSAM striping is done on a CI basis to provide efficiency for both random and sequential access to a key-sequence data set (KSDS). This method spreads the I/O processing as evenly as possible for the data being read or written. When reading data sequentially from a KSDS, VSAM does read-ahead processing using the primary index to determine the logical order of the data CIs and passes a request to the I/O driver to retrieve this set of data CIs. The I/O driver sorts the control intervals in ascending CI sequence and then sorts them by stripe. Next, it creates a channel program to access each stripe and submits multiple requests to the I/O supervisor to execute the channel programs simultaneously.

The maximum number of stripes supported for VSAM data sets is 16. VSAM data sets may be extended on the same volume, or if the space is not available on that volume, they may be extended to a new volume. When space is added to one stripe, all stripes are extended by the same amount of space. This multi-

Figure 5 DFSMSdfp striping



volume-per-stripe condition has been referred to as layering, in which a set of stripes resides on multiple volumes.

Striped SAM data sets. For a SAM data set, the data-to-stripe mapping is done at the granularity of a disk track. The parallel I/O function is used to perform striping of SAM data sets. It retrieves a full track from each stripe across all stripes allocated to the data set. One difference between striping for SAM and striping for VSAM is that a striped SAM data set cannot be extended to a new volume. Each stripe for a SAM data set may contain nearly a full volume of data, and the number of stripes that contain the data defines the maximum size of the SAM data set, because a stripe may not be extended to a new volume. The number of stripes for a SAM data set may be as large as 59 to allow the data set to be as large as 59 times the volume size.

Figure 5 shows simple examples of SAM and VSAM data striping. For each example, a striping level of 3 is used. Stripe 1 of the data set resides on disk volume 1, stripe 2 on volume 2, and stripe 3 on volume 3. For the SAM case, the data striping is done at the granularity of a disk track. Data 1 is the first track of data for the data set. Data 2 through Data 9 are the remaining tracks of the data set. For VSAM, the mapping of data to stripe is done at a granularity of CI. In the example, there are three control intervals of data per disk track. VSAM begins CI numbering for a data set at 0, rather than 1. The figure illustrates how CI 00 through CI 26 of the data set are striped across the tracks of the three disk volumes.

DFSMS support for business continuance

In addition to providing space and availability management, DFSMS provides functions that aid installations in making copies of production data for the purpose of business continuance. Having copies of these production data in a secure location separate from the primary data processing facility enables businesses to continue to function in the event of a catastrophic failure that might render their main processing center inoperable (in part or in full).

DFSMS provides various means of allowing installations to replicate production data into an alternate data processing facility or facilities. These facilities could be off-site tape vaults, off-site tape libraries, remote disk farms, or separate data processing complexes, complete with tape and disk storage facilities. Installations may opt also for a combination of different off-site data storage options available to them.

Installations can select a particular DFSMS business continuance solution to meet their business continuity needs, or they can mix and match solutions depending on the requirements of the individual applications or data.

When discussing business continuance, two terms are commonly used, *recovery time objective* (RTO) and *recovery point objective* (RPO). RTO is described as how long it takes to recover the data of an application and have all critical operations up and running after an outage occurs. RPO is described as a point in time at which all backup data are current. In other words, they indicate how much data an enterprise can lose and still be viable. For instance, the requirement for a bank might be that all data must be current within 15 minutes, whereas for another organization it is only critical for data to be current within the last 48 hours. Within a particular enterprise there may be various RTO and RPO goals, depending on the criticality of particular applications.

In the early 1990s, customers working with IBM defined six different disaster recovery or business continuance approaches, which they referred to as Tiers. The Tiers are briefly summarized below and are depicted in Figure 6.

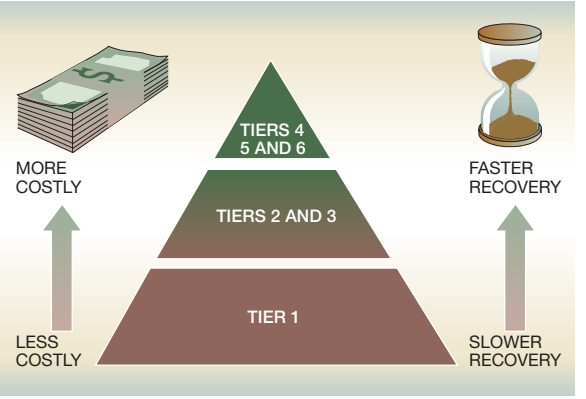
- Tier 0: No disaster recovery
- Tier 1: Physical transport of backup copies to an off-site location with no data processing capability
- Tier 2: Physical transport of backup copies to an off-site location with some data processing capability
- Tier 3: Off-site electronic vaulting, possibly an off-site tape library
- Tier 4: Two active sites with application software mirroring
- Tier 5: Two-site, two-phase commit. In this case, database logs are sent to the recovery site where programs apply the logs to databases.
- Tier 6: Disk and tape storage subsystem mirroring

Typically, installations will select lower tier solutions when their RTOs and their RPOs are longer and will select the upper tier solutions as the RTO and RPO objectives become shorter.

DFSMS has a number of solutions that fit into the Tier 1 to 5 business continuance requirements. They include the DFSMShsm full volume dump and restore solution, the DFSMSdss physical and logical data set and full volume dump and restore solutions, and the DFSMShsm aggregate backup and recovery support solution, known as ABARS. An enterprise may select one or more of these solutions to meet their business continuance objectives. ABARS is probably the most commonly used DFSMS solution for Tiers 1 through 5. ABARS allows installations to define an aggregation of data to be backed up and restored as a logical entity. ABARS maintains the point in time relationship of the data within a specific aggregate group, so that when the application is brought on line at the recovery location, the data are synchronized and the application can be started.

The scope of the aggregation of data that is defined to ABARS is typically that of a critical application or applications. ABARS will find where the data exist within the DFSMS storage device hierarchy, including user disk, user tape, DFSMShsm migration disk volumes, and DFSMShsm migration tape volumes and package those data into one to four output files that can be physically or electronically sent to an off-site location. ABARS will also collect the meta-data associated with the backed up data, such as catalog information, Generation Data Group base information, or DFSMShsm control data set records for migrated data sets, and restore that information along with the data.

Figure 6 Disaster recovery tiers

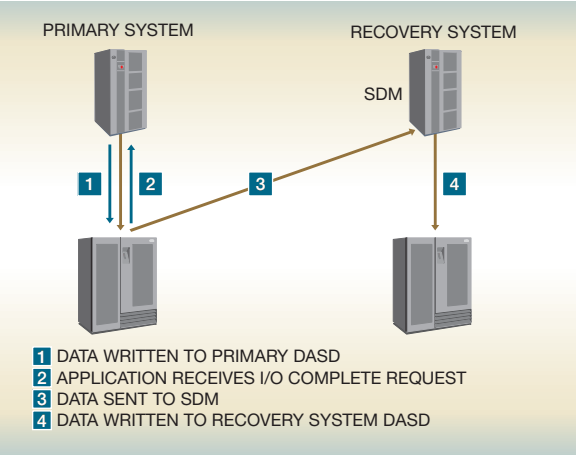


In addition to the software-only business continuance solutions, DFSMS contains a software function called the System Data Mover (SDM) that, when combined with the appropriate microcode on IBM, Hitachi Data Systems, or Amdahl disk storage subsystems, provides installations with an extended distance remote copy capability called XRC. XRC can copy (mirror) critical data over long distances with minimal impact to the primary application I/O activity. XRC supports short RTO and RPO objectives as well as providing an upper-tier disaster recovery solution.

XRC overview. As a host application issues a write I/O request to a disk on the primary disk subsystem that is part of an XRC configuration, shown in Figure 7, the XRC function intercepts the write I/O request and captures the information required by the SDM to create the write I/O operation on the recovery system. Asynchronous to the application I/O request, the SDM communicates with the primary disk subsystem and collects the updates, then journalizes them into consistency groups that are written to the recovery site target disks. A consistency group contains records that have their order of update preserved across multiple logical control units within an IBM TotalStorage* Enterprise Storage Server* (ESS), across multiple ESSs and across other storage subsystems participating in the same XRC session. The consistency groups enable writes to the secondary disk subsystem to be done in the proper order, maintaining I/O consistency to a specific point in time.

Typically, a single instance of the SDM can manage between 1200–1800 volumes, and up to five SDMs

Figure 7 XRC session



can be active on a single OS/390 or z/OS logical partition (LPAR). DFSMS also provides coupled XRC support that allows multiple SDM sessions to be coupled together so that installations can manage even larger numbers of volumes as a single session, enabling all volumes in the session to be recovered to the same consistent time. In an XRC configuration, the primary disk storage subsystem must be capable of running the XRC function; however, the target secondary disk subsystem can be any disk and need not be capable of running XRC.

As installations formulate their business continuance strategies, they can choose from a variety of DFSMS business contingency solutions. The choice of DFSMS solutions ranges from Tier 1 to Tier 6 and can meet various RTO and RPO objectives.

DFSMS disk copy services

In addition to the business continuance solutions mentioned in the previous section, DFSMS provides advanced disk copy services that allow installations to efficiently and reliably copy disk volumes or data sets from one disk image to another. The target disk images can then be used as source volumes for a subsequent dump of the data to tape. Besides the remote copy services functions, there are three disk copy services that are provided by DFSMS: Concurrent Copy, SNAP/SHOT*, and FlashCopy*.

Concurrent Copy. Concurrent Copy is a storage subsystem extended function that can generate a copy or a dump of data while applications are updating those data. DFSMSdss is the external interface to the Concurrent Copy function and works with the SDM to control the process. DFSMShsm also takes advantage of the Concurrent Copy feature when invoking DFSMSdss during its backup or dump processing.

Functionally, the data being dumped or copied are unavailable only for the time it takes to initialize a Concurrent Copy session for the data in question. The data are serialized while the initialization process takes place, which is normally only a matter of seconds, after which the data can be deserialized and the application can continue processing the data. The data copy process is said to be "logically complete" after the Concurrent Copy initialization process has completed. However, the data have not yet been physically written to the target media. After the data have been successfully written to the output media, the Concurrent Copy is said to be "physically complete." Concurrent Copy is an excellent tool for installations that need to reduce the time during which their data are unavailable while they make backup copies or dump copies of data sets or volumes.

SNAP/SHOT. The SNAP/SHOT function enables customers to produce almost instantaneous copies of data sets, volumes, and VM (virtual machine) minidisks—all without data movement. SNAP/SHOT is automatically initiated by DFSMSdss on storage subsystems with the SNAP/SHOT feature. This function is only applicable to the RAMAC* Virtual Array.[8]

Data are "snapped" (quickly copied) directly from the source disk location to the target disk location. This function is externalized in DFSMS with the DFSMSdss COPY command. This command is used to copy volumes, tracks, or data sets from one disk volume to another. DFSMSdss uses this method whenever the source and target data are located on like devices in the same partition on the same storage subsystem and no reblocking is required. With "native" SNAP/SHOT, the copy of the data is logically and physically complete as soon as the snap is complete.

Another way in which DFSMS exploits SNAP/SHOT is via a function called "virtual Concurrent Copy." Virtual Concurrent Copy uses SNAP/SHOT to provide a concurrent-copy-like function when the storage subsystem supports SNAP/SHOT but not Concurrent Copy. During virtual Concurrent Copy, data are "snapped" from the source volume to an intermediate location, and the data are gradually copied to the target location using normal I/O methods. The

data are logically complete after the data are "snapped" to the intermediate location and physically complete after the data are moved to the target media. The external interface to virtual Concurrent Copy is via the DFSMSdss product and is also exploited by the DFSMShsm product when it invokes DFSMSdss to perform data movement operations on its behalf.

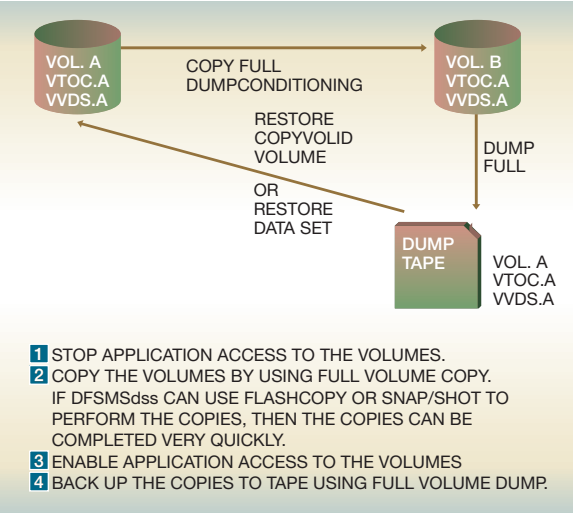
FlashCopy. FlashCopy provides a point-in-time copy of data for backup and recovery operations. Currently this function is only supported on ESS. Both the source and target volumes must reside on the same logical subsystem. DFSMSdss automatically invokes FlashCopy when performing a full volume copy operation and the copy operation is for data on a subsystem that supports FlashCopy functions such as ESS.

Functionally, the data being copied by FlashCopy are unavailable only long enough for DFSMSdss to initialize a FlashCopy session, which normally only takes a few seconds to complete. In contrast with SNAP/SHOT and Concurrent Copy, using FlashCopy allows the data on both the source and target volumes to be available immediately after the initialization process is complete.

As shown in Figure 8, installations can also utilize FlashCopy to produce a copy of a volume in seconds and then use DFSMSdss to dump the copy to tape while applications are accessing the data on the original volume.

Installations can even take this operation a step further by requesting that DFSMSdss condition the target volume during the copy operation so that the target volume, when dumped, will look as though it were dumped from the original source volume of the copy operation. For example, if a full volume copy is performed with DUMPCONDITIONING of a volume called VOL001 to a volume called VOL002 and then a full volume dump of VOL002 is performed, the dump data set looks as though it were created by performing a full volume dump of VOL001. Additionally, if a user knows that he or she only wants to create a FlashCopy so that the target volume can be immediately dumped, DFSMSdss provides a NOCOPY option that prevents the ESS subsystem from performing a physical copy of the data. Performing the physical copy uses subsystem resources and can impact the performance of other I/O operations that are issued to ESS. There is also a FlashCopy WITHDRAW option that tells DFSMSdss to withdraw the FlashCopy relationship after the volume is dumped. This withdrawal frees the subsystem resources that are used to maintain the FlashCopy relationship.

Figure 8 DFSMS disk copy services



DFSMS testing

Just as the storage management products have grown and evolved over the years, the test philosophy for these products has changed from a function-based focus to a solutions-based approach, assessing the quality of the DFSMS software products working with storage product hardware, the other z/OS software stack, and the database product offerings. As customers have come to rely more on integrated solutions that work together, rather than just individual pieces, the focus of testing storage products has had to evolve into a more solution-oriented approach to support this need. This approach has required software testing to rely heavily on interaction with the storage hardware teams. Testing has also expanded to leverage cross-site environments to verify database products, and the DFSMS products work as part of the business solution for storage management. The following subsections expand on this evolution of software testing.

Unit test. The DFSMS test cycle starts with unit test. This test is the initial verification that the new and changed code within a module (software part) is error-free. It is performed by the software developer who wrote the code. The focus at this point is on the

individual parts. This test is done in a Time Share Option test or VM guest environment so that the test variables can easily be controlled to ensure the code is performing as designed.

Development verification test. After unit test, the test cycle continues with a development verification test (DVT). This phase of testing is also performed in development and brings together the code from multiple developers. This test is the initial verification that the new and the changed code can be integrated and the result is error-free. DVT is performed by the individuals involved in the development effort and is also done in a VM guest environment so that the test environment can be controlled and verified.

Function verification test. The next phase of testing is a component test, usually known as the function verification test (FVT). Here the set of modules that comprise a component or function are verified. Areas such as external interfaces (e.g., panels, commands, messages), component interfaces, hardware and software interfaces, and application program interfaces are verified. The focus is on nonmessage event recording (e.g., SYS1.LOGREC records, SMF [System Management Facility] records), RAS (reliability, availability, and serviceability) characteristics and error diagnosis, shared paths (multitasking), and shared resources (files, locks, queues, etc.) in order to ensure function completeness. FVT is most often run in a VM guest environment similar to unit and development verification test.

System verification test. In the early years of DFSMS, only unit test and FVT were performed before the code was used in internal production or early customer support tests. As the interrelationship of the various DFSMS components grew, the need for additional internal testing became apparent. System verification test (SVT) was started to fill this requirement. SVT focuses on validating a given product in predefined system environments with a variety of user-oriented workloads and scenarios. Often this test is the initial exposure of the new software on the actual hardware. The focus is on load, stress, recovery, migration, and usability. It includes test case streams, predefined scenarios, and shared hardware resources.

The initial basis of DFSMS system test was grounded primarily in software validation for a specific software product set. The next step was to expand this testing to encompass joint test efforts with the storage hardware product teams. This expansion was a key step toward gaining the efficiency and effectiveness that solution testing of storage products offered. It relies heavily on the interaction of the software and hardware teams.

A good example of joint interaction is the testing for copy services products. The interrelationship of XRC, FlashCopy, and Concurrent Copy with the latest ESS hardware makes it imperative that changes be tested from a solution perspective. By working with the ESS hardware team, copy services functions are run across multiple z/OS images to a wide variety of ESS hardware. Another area that has leveraged this approach is the DFSMS system-managed tape support, including the latest 3590 devices and VTS hardware. By performing software testing jointly with the hardware team, tests can run in a sysplex under multiple system levels to all types of library configurations. With this approach there can be multiple users of the system and hardware, allowing for not only the planned testing, but a large volume of unpredictable interaction. This dynamic environment provides a better opportunity to flush out many customer-related problems for both the hardware and software components.

Internal production. The solution-based testing continues with pre-GA (general availability) installations of storage product software and hardware on software development production systems. These systems support code development, library control, program compilations, reports, and other new release activities. The testing provides firsthand experience to the development team from the results of their latest efforts.

Integration test. Integration test is the last internal validation of DFSMS prior to GA. It runs in parallel with the early support program. This testing is a partnership that brings together storage products, server products, and database products in a large, robust, dedicated environment with various processors and ESS storage devices. It runs a pseudo production workload with the latest GA subsystems, focusing on IMS* (Information Management System), DB2* (Database 2*, the relational database management system), CICS, and VSAM RLS data sharing. This test experience is documented in the *Parallel Sysplex Test Report*, available on the Web as a PDF file.[9]

Solution level combined IBM database and DFSMS test. This combination is a solution test launched early in 2002 to improve validation of DFSMS and storage hardware with the latest IBM database prod-

ucts under stress and duration. Customer-like activities defined by product teams allow the interaction of GA and pre-GA storage and database products to find problems before these solutions reach a customer.

Consolidated service test. This test effort was launched to provide a single, consistent, installable maintenance recommendation across the z/OS software stack for software and hardware products. The test group is comprised of a cross-laboratory, remotely disbursed team with the goal of reducing confusion and potential business outages caused by conflicting service levels and products failing to work together consistently. The test includes the recommendation of a tested maintenance level for z/OS, OS/390, and key subsystems including CICS, IMS, DB2 and MQSeries* (messaging and queuing series). A quarterly report with the recommended maintenance level is available.[10]

Conclusion

DFSMS is a policy-based storage management solution for OS/390 and z/OS environments. It has been extended over the last two decades to support and exploit new server and storage technology. The policy-based functions of DFSMS have also served as a conceptual base for some of the policy-based storage management functions provided by the IBM Tivoli Storage Manager product. They are also likely to have an influence on future storage and storage management products. This paper briefly described a number of recent optimizations that improve DFSMS internal efficiency and enhance its data access performance, data sharing, and data recovery capabilities. DFSMS provides a number of functions in support of business continuance, including the System Data Mover with its exploitation of the extended distance remote copy storage hardware function. DFSMS has extended its ongoing commitment to quality by participating in and leading the movement beyond product-level testing to solution testing.

Appendix: Definitions of some terminology

catalog A repository that keeps track of data set location. Cataloging a data set simplifies subsequent retrieval of the data set with only the data set name being required. Specific volume and device information can be omitted for the data set request.

Data Facility Storage Management Subsystem (DFSMS) An operating environment that helps automate and centralize the management of storage. To manage storage, the Storage Management Subsystem provides the storage administrator with control over data class, storage class, management class, storage group, and automatic class selection routine definitions.

DFSMSdfp A DFSMS functional component of z/OS that provides functions for storage management, data management, program management, device management, and distributed data access.

DFSMSdss A DFSMS functional component of z/OS used to copy, move, dump, and restore data sets and volumes.

DFSMShsm A DFSMS functional component of z/OS that provides both availability and space management.

DFSMSrmm A DFSMS functional component of z/OS that manages removable media.

extended format The format of a data set that has a name type (DSNTYPE) of EXTENDED. The data set is structured logically in the same way as a data set that is not in extended format, but the physical format is different.

IBM TotalStorage Enterprise Storage Server (ESS) An IBM high-end storage subsystem designed for midrange and high-end environments. ESS provides large capacity, high performance, continuous availability, and storage expandability.

KSDS A VSAM key-sequenced data set that has an organization in which records are sequenced on a key field.

logical volume In the context of a virtual tape server, logical volumes have a many-to-one association with physical tape media and are used indirectly by z/OS applications. They reside in a virtual tape server on a stacked volume (physical tape media) or on exported stacked volumes.

logical subsystem The logical functions of a storage controller that allow one or more host I/O interfaces to access a set of devices. The controller aggregates the devices according to the addressing mechanisms of the associated I/O interfaces. One or more logical subsystems exist on a storage controller. In general, the controller associates a given set of devices with only one logical subsystem.

Peer-to-Peer Virtual Tape Server (PtP VTS) A virtual tape server (VTS) configuration with two interconnected virtual tape servers maintaining a copy of the tape volume in each VTS. This dual-copy capability is transparent to the user and host processor resources.

spill backup volume A volume owned by DFSMShsm to which all but the latest backup version of a data set are moved when more space is needed on a disk daily backup volume or all valid versions are moved when a tape backup volume is recycled.

stacked volume A volume that has a one-to-one association with physical tape media and which is used in a virtual tape server to store logical volumes.

stripe In DFSMS, the portion of a striped data set that resides on one volume. The records in that portion are not always logically consecutive. The system distributes records among the

stripes such that the volumes can be read from or written to simultaneously to gain better performance.

striped data set In DFSMS, an extended-format data set consisting of two or more stripes. Striped data sets can take advantage of the sequential data striping access technique.

sysplex A set of z/OS systems communicating and cooperating with each other through multisystem hardware components and software services to process customer workloads.

Virtual Storage Access Method (VSAM) An access method for direct or sequential processing of fixed and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative-record number.

virtual tape server (VTS) This subsystem, integrated into the IBM TotalStorage Enterprise Automated Tape Library (3494), combines the random access and high-performance characteristics of disk with outboard hierarchical storage management and virtual tape devices and tape volumes.

virtual volume A tape volume that resides in a tape volume cache of a virtual tape server. Whether the volume resides in the tape volume cache as a virtual volume or on a stacked volume as a logical volume is transparent to the host.

VSAM record-level sharing (VSAM RLS) An extension to VSAM that provides direct record-level sharing of VSAM data sets from multiple address spaces across multiple systems. Record-level sharing uses the z/OS coupling facility to provide cross-system locking, local buffer invalidation, and cross-system data caching.

Cited references and notes

1. J. P. Gelb, "System-Managed Storage," *IBM Systems Journal* **28**, No. 1, 77–103 (1989).
2. "Allocation" describes the function of z/OS in which resources such as volumes and devices are assigned to fulfill a data set request.
3. The disk volume can be a logical volume within a disk subsystem.
4. LIMIT is defined as the *net* number of tapes that have been returned to the scratch pool: the number of input tapes returned to the scratch pool less the number of new output tapes written.
5. For enhanced capacity tape cartridges, the capacities double with an IBM Tape Drive 3590 Model B being able to write 20 GB and a 3590 Model E being able to write 40 GB of data on a cartridge.
6. A stripe is the portion of the data set that resides on one volume.
7. CI is a VSAM logical disk block. It is the minimum number of data bytes transferred by a VSAM disk access. The size of a CI is an exact multiple (1 or more) of the disk block size for the type of device in use (e.g., an IBM 3390 direct access storage device).
8. The name RAMAC was originally derived from random access method of accounting and control and was given to the first IBM disk storage system. It is now the name of an IBM product line consisting of a multiple-disk storage subsystem.
9. *Parallel Sysplex Test Report*, Latest Edition, z/OS Integration Test, IBM Corporation, http://www.s390.ibm.com/os390/support/os390tst.
10. Consolidated Service Test and the RSU, IBM Corporation, http://www-1.ibm.com/servers/eserver/zseries/zos/servicetst.

General references

DFSMS, IBM Corporation, http://www-1.ibm.com/servers/storage/software/sms/.

R. F. Kern and V. T. Peltz, *IBM Storage e-Infrastructure for Multi-Site Data Availability*, White Paper, IBM Corporation (November 16, 2001).

J. P. Strickland, "VSAM Record-Level Data Sharing," *IBM Systems Journal* **36**, No. 2, 361–370 (1997).

z/OS DFSMSdfp Storage Administration Reference, SC26-7402, IBM Corporation.

z/OS DFSMSdss Storage Administration Reference, SC35-0424, IBM Corporation.

z/OS DFSMShsm Storage Administration Guide, SC35-0421, IBM Corporation.

z/OS DFSMShsm Storage Administration Reference, SC35-0422, IBM Corporation.

z/OS DFSMS Implementing System-Managed Storage, SC26-7407, IBM Corporation.

z/OS DFSMS Object Access Method Planning, Installation, and Storage Administration Guide for Tape Libraries, SC35-0427, IBM Corporation.

z/OS DFSMSrmm Guide and Reference, SC26-7404, IBM Corporation.

z/OS Internet Library, IBM Corporation; see http://www.ibm.com/servers/eserver/zseries/zos/bkserv.

Accepted for publication January 14, 2003.

Lyn L. Ashton *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (ashton@us.ibm.com).* Ms. Ashton joined IBM in 1978, working in the Tucson Product Test and Assurance Laboratory. In 1990 she transferred to the Tucson Programming Center as a software developer. She is a Senior Technical Staff Member and is currently a DFSMShsm architect.

Edward A. Baker *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (ebaker@us.ibm.com).* Mr. Baker is an advisory software engineer in the SSG Software Strategy and Architecture department. He joined IBM in Tucson in 1979. From 1979 to 1983 he was an MVS application programmer supporting manufacturing applications. In 1983 he transferred to the DFSMShsm development organization as a system programmer. From 1998 to 2001 he worked in the IBM Product Introduction Centre in Hursley, United Kingdom, coordinating early support programs for DFSMS software and IBM tape hardware products. In 2001, he returned to Tucson and joined the SSG Software Strategy and Architecture department as an architect for the DFSMShsm and DFSMSdss products.

Arthur J. Bariska *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (bariska@us.ibm.com).* Mr. Bariska is a senior software engineer in the DFSMS development laboratory. He joined IBM at Poughkeepsie, New York, in 1974, working as a systems analyst and later a manager in the Materials Management organization. In 1982, he transferred to Tucson, holding management positions in the Materials Management organization and the Computer Operations Center. He moved to the DFSMS development laboratory in 1989 as a programmer in the test organization. He has been a member of the DFSMS Systems Test team since 1993.

Erika M. Dawson *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (brosch@us.ibm.com).* Ms. Dawson joined IBM in 1987 and has spent the majority of her career working in the Object Access Method (OAM) component of DFSMSdfp on the system-managed tape support. She is a senior software engineer currently working as a development team leader and product architect.

Ruth L. Ferziger *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (ruthf@us.ibm.com).* Ms. Ferziger is an advisory software engineer. She joined IBM at San Jose, California, in 1984 as part of the team that developed the storage management subsystem (SMS). She is currently the development team lead and designer working on future VSAM RLS-related enhancements.

Stanley M. Kissinger *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (kissingr@us.ibm.com).* Mr. Kissinger is a senior software engineer in the DFSMS development laboratory. He joined IBM in 1985 and has spent his entire career in Tucson. From 1985 until 1997, he worked in technical support in the Level2 support center. From 1997 to the present time, he has worked as both a developer and architect for DFSMShsm.

Terri A. Menendez *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (terriam@us.ibm.com).* Ms. Menendez is a senior software engineer. She is currently working in VSAM RLS development.

Sanjay Shyam *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (shyams@us.ibm.com).* Mr. Shyam is a senior software engineer. He is currently a software architect for the SMS component of DFSMSdfp.

Jimmy P. Strickland *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (jstrickl@us.ibm.com).* Mr. Strickland is an IBM Distinguished Engineer. During his career, he has held various technical leadership positions, among them, technical lead for the initial release of VSAM and technical lead for the initial releases of the common DB2 and IMS lock manager component IRLM (IBM Resource Lock Manager) and the IMS Logger. He was a member of the team that invented the IBM System/390® Parallel Sysplex technology and developed its architecture. Currently, he is a storage software architect.

Dave K. Thompson *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (dkt@us.ibm.com).* Mr. Thompson is a senior software engineer in DFSMS at the San Jose Programming Laboratory. He joined IBM at Riverside, California, in 1966. From 1966 until 1969, he was a software customer engineer working from the Riverside branch office. In 1969, he transferred to the San Jose development laboratory where he performed development work for VSAM and design and development work for Media Manager.

Glenn R. Wilcock *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (wilcock@us.ibm.com).* Mr. Wilcock joined IBM at Tucson in 1992. Since 1992, he has worked in both DFSMSdss and DFSMShsm development. He is an advisory software engineer currently working as a development team leader for DFSMShsm.

Mike W. Wood *IBM Global Services, MP14, P.O. Box 31, Birmingham Road, Warwick CV345JL, United Kingdom (mikew_wood@uk.ibm.com).* Mr. Wood is a consultant IT architect currently working in the DFSMSrmm development team. He joined IBM at Croydon, United Kingdom, in 1976 as a computer operator. From 1978 to 1990 he was an MVS™ system programmer, and in the latter years he was the UK MVS systems architect. Since 1990 he has driven the development of the IBM Removable Media Manager (DFSMSrmm).