VM/ESA support for coordinated recovery of files

by C. C. Barnes
A. Coleman
J. M. Showalter
M. L. Walker

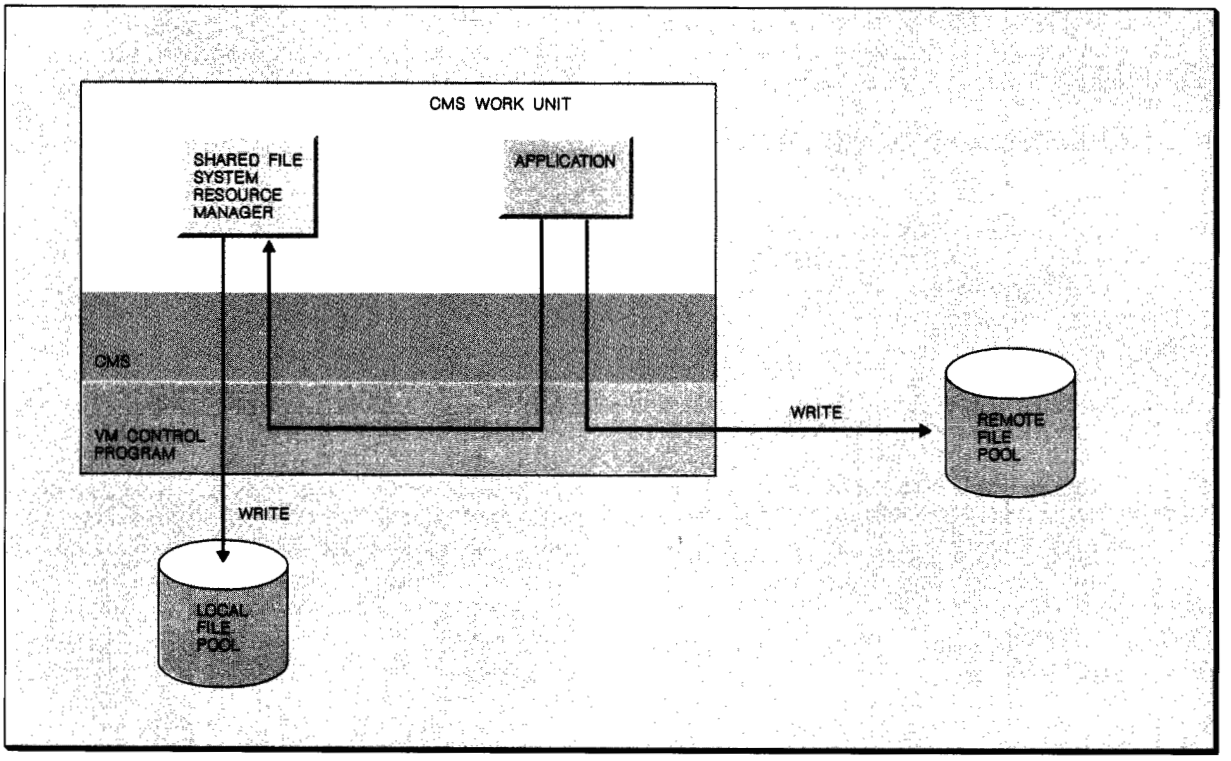
This paper discusses the concepts and facilities of the Shared File System (SFS) support for Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™) Coordinated Resource Recovery (CRR). It includes background information on limitations that lead to SFS support for coordination of file recovery functions. The level of support provided by the Virtual Machine/System Product (VM/SP) Release 6 SFS support is identified and contrasted with the support provided in VM/ESA. The paper contains an overview of the system structure and the rationale for the support and is a discussion from the overall perspective of the total system environment and system processing for resource recovery. After the concepts and structure of VM/ESA SFS support are introduced, the paper discusses the specific technology involved in providing SFS support for Coordinated Resource Recovery. This includes a discussion of specific facilities used by SFS and how SFS deals with certain conditions that can arise. In addition, this paper discusses the Conversational Monitor System (CMS) compatibility considerations that contributed to the design of SFS support for Coordinated Resource Recovery. This includes compatibility with prior releases and compatibility with the CMS file system support for minidisks. Finally, some of the future directions for file system support of resource recovery are identified along with some of the challenges that remain to be solved.

The Shared File System in the Virtual Machine/System Product (VM/SP) introduced the concept of recoverable files and application controls over committing or backing out changes to such files. The SFS support in Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™) and its support for Coordinated Resource Recovery (CRR) extends the concepts and support. Reference 1 contains a complete understanding of the recoverable file support introduced by the SFS in VM/SP, but some of the key concepts and facilities are summarized here for convenience.

A *CMS work unit* is a unit of processing on behalf of a Conversational Monitor System (CMS) application. Resource recovery functions are supported for certain resources processed by, or on behalf of, the application. CMS provides application services that allow CMS applications to control what processing is done for a particular work unit context. That is, an application may choose to separate its processing into multiple, distinct work units to effect different recovery processes for separate phases or instances of its processing.

A *recoverable resource* is any persistent application resource (data or object) that is enabled for resource recovery processing. For recoverable resources, all processing of the resource in the context of one CMS work unit is treated as an

Figure 1 CMS Shared File System use of recovery coordination without CRR



atomic unit of processing. Either all changes are effected, or none of the changes are effected. Not all resources available to CMS applications can (or need be) defined to be recoverable. Indeed, some resources, by their nature, should not be subjected to resource recovery functions.

A *recoverable file* is a file that is subject to resource recovery functions. With the CMS SFS, a file can be defined to be recoverable or nonrecoverable. Data files would typically be defined as recoverable files. Files that record status (e.g., print files or log files) would typically be defined as nonrecoverable.

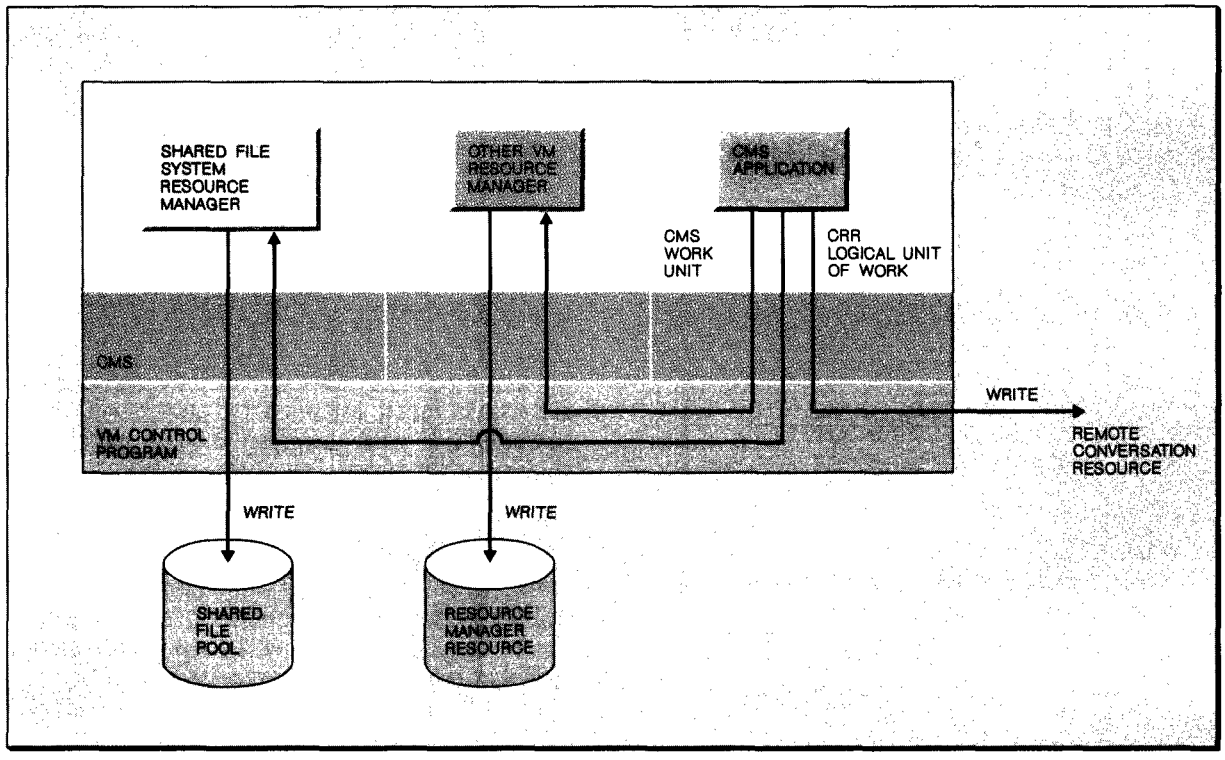
Commit is a CMS function that tells CMS to complete all changes made to recoverable resources that were made in the context of a specified (or implied) CMS work unit. For example, a CMS commit would cause the SFS to commit to permanent storage all changes to all recoverable files that were changed in the context of the CMS work unit specified by the commit.

Backout is a CMS function that tells CMS to "undo" all changes made to recoverable resources in the context of a specified (or implied) CMS work unit. For example, a CMS backout would cause the SFS to restore all recoverable files to the state they were in before the changes done in context of the CMS work unit.

While SFS support introduced in VM/SP extended the capabilities of the CMS file system, it did have some limitations: SFS in VM/SP does not support updating multiple files in multiple file pools within the context of one CMS work unit. That is, SFS could only provide consistent file updates within a single file pool. While this is clearly an improvement over minidisk support, it does represent a limitation in single systems and in distributed (multiple system) environments.

With VM/SP Shared File System support, there was also no system-assisted method of coordinating the committing (or backing out) of CMS file updates with updates to resources managed by

Figure 2 CMS Shared File System participation in VM recovery coordination



other resource managers. Support in this area would have to be built into the application that was using both SFS files and the other resource.

Given the limitations of the VM/SP recovery support in SFS, the objectives established for SFS support in VM/ESA included objectives to overcome these limitations. In both cases, the Coordinated Resource Recovery support in VM/ESA plays a key role in overcoming the limitations. While other solutions (e.g., SFS unique recovery coordination facilities) could have been employed, the use of common system services for this provided a more robust solution.

Figure 1 shows a CMS file system application that is attempting to write to files in two different SFS file pools within the same CMS work unit. Such an application is not supported in VM/SP. The objective for VM/ESA was to support such a CMS application, which included: allowing writing to multiple (local) file pools within one CMS work unit, and allowing writing to both local and remote file pools on one CMS work unit.

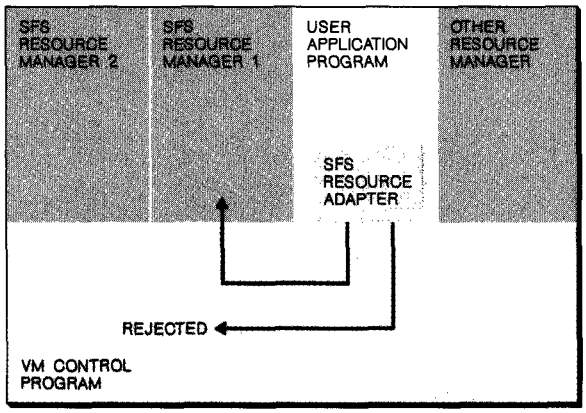
The support is intended to accommodate applications that perform synchronized broadcast of file (for example, library) updates to multiple systems, and that move data (or files) from one file pool to another (copy and erase coordination across file pools).

Through SFS use of the Coordinated Resource Recovery facilities for coordinating commit processing, updates to multiple file pools can be supported on the same CMS work unit.

Figure 2 illustrates the objectives for SFS participation in recovery coordination with other VM resource managers.

Figure 2 shows a CMS application that is attempting to write to multiple VM resources (local and remote) and shared files on the same logical unit

Figure 3 VM/SP Release 6 usage of multiple SFS file pools without CRR



of work. This "local" logical unit of work is the VM/ESA Coordinated Resource Recovery equivalent of a CMS work unit. The objective for VM/ESA was to support such applications by coordinating the write activity on the CMS work unit with the corresponding VM/ESA logical unit of work, such that the application does not have to code the logic to synchronize the file writes with the other application update activity.

The specific objective for SFS participation in recovery coordination was coordination of SFS with other resource managers that might participate in CRR. With the introduction of CRR support in VM/ESA, it is both possible and desirable to provide SFS support for CRR logical units of work, such that SFS can participate as a resource manager in a multi-resource VM application. This was desired for two reasons:

- To enable the use of SFS by resource managers wishing to effect participation in CRR by using the file system
- To allow applications to coordinate CMS file updates with updates to other resources managed by separate CRR-capable resource managers

By registering as a Coordinated Resource Recovery participant, updates to CMS files managed by the Shared File System can be coordinated with updates to other resources updated by the CRR application.

SFS participation in CRR is related to other SFS enhancements (as described in Reference 1). SFS participation in Coordinated Resource Recovery applications increases the need for SFS support for nonrecoverable files in order to support application logs, audit files, or other such files that are not supposed to be affected by coordinated rollback processing. Because the coordinated (CRR) commit involves committing resources other than SFS resources, the rejection of commit processing due to open files would be extremely undesirable. Thus, if SFS files are to be used by CRR applications, it is necessary that SFS remove this restriction.

What follows is a discussion of the structure of the SFS support, followed by considerations of compatibility with the system. The paper concludes with a discussion of future directions.

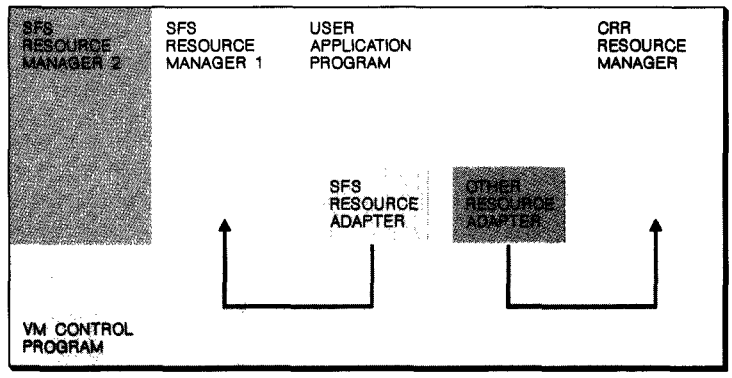
System structure of VM/SP

In VM/SP, SFS consists of two major parts, the SFS resource adapter that resides in the requesting user's virtual machine, and the SFS resource manager that resides in a separate server virtual machine. The SFS resource manager is sometimes called the SFS file pool server. The SFS resource adapter translates file system requests to advanced program-to-program communications (APPC/VM) with the SFS resource manager. Each SFS resource manager manages one SFS file pool of data. The SFS resource manager receives the APPC/VM communications sent by the SFS resource adapter and performs the requested file or directory operation. In performing the operation, the SFS resource manager provides other implicit operations such as authorization checking, locking for sharing, and logging for recovery.

Updates to multiple SFS files can be coordinated as long as they reside in a single file pool. Attempts to update files in multiple file pools in a single work unit are rejected by the SFS resource adapter (see the request to SFS resource manager 2 in Figure 3). The reason for this is that each SFS resource manager has its own independent recovery log. This enables coordination of updates within a file pool but not between file pools. There was no coordination of updates made to multiple file pools.

In addition, there was no capability to coordinate SFS updates with any other resource manager's

Figure 4 VM/SP Release 6 usage of multiple resource managers without CRR



updates. The SFS commit routine only committed SFS resources and no others. Even if other resource managers had a commit function, they had no way of participating in a coordinated commit. Each resource manager's commit would have to be done separately, allowing the possibility that one commit may succeed and another commit may fail (see the separate requests to SFS resource manager 1 and the other resource manager in Figure 4).

System structure of VM/ESA

Commit processing. In VM/ESA, the SFS resource adapter and the SFS resource manager still use APPC/VM conversations to communicate. In addition, they also communicate with a Coordinated Resource Recovery facility. The CRR facility supports coordination among multiple resource managers and protected conversations.[2] The SFS resource adapter communicates with the CRR synchronization point manager (or sync point manager) and the SFS resource manager communicates with the CRR recovery server.

As the SFS resource adapter receives requests, it determines whether each request is for a new work unit or a new SFS resource manager (for example, a new SFS file pool). In either case, the SFS resource adapter provides a new registration with the sync point manager. If requests are made to two SFS file pools in one work unit, the SFS resource adapter registers twice. When a commit is issued (either by the application or implicitly by CMS), the sync point manager invokes the SFS resource adapter to perform its *two-phase* commit procedure (a prepare phase followed by commit phase). The sync point manager indicates that only a one-phase commit procedure is required if a single SFS file pool is the only resource involved in the commit. If the commit cannot be completed by all resource managers involved, the sync point manager tells the SFS resource adapter to roll back or backout the changes.

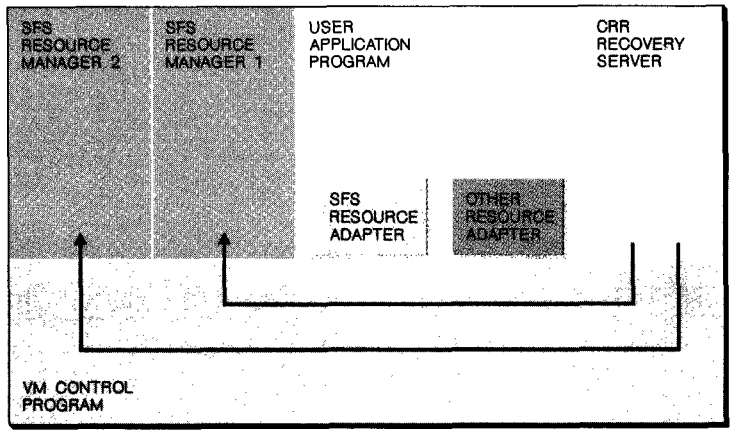
This same kind of processing is supported by an SFS resource manager connected by the Transparent Services Access Facility (TSAF) of VM or by APPC Virtual Telecommunications Access Method (VTAM) support. Only the CRR recovery server on the requesting user's system is used for logging during commit as long as the application is not using any protected conversations.

Resynchronization. If a failure occurs during commit processing, the state of the commit may indicate that resynchronization processing is required. Some examples of such failures are:

- A termination of the conversation between the resource adapter and its resource manager
- A failure of the resource manager
- A failure of the entire system

Resynchronization is required when the SFS resource manager has completed phase one of a two-phase commit procedure and has not been told whether to commit or backout. The CRR recovery server can determine this situation from the log information written during the commit processing. In this situation, the CRR recovery server automatically initiates resynchronization

Figure 5 VM/ESA resynchronization processing



when the failure has been resolved (see Figure 5). This does not involve the user virtual machine that made the original update requests. In fact, the user does not even have to be logged on. The CRR recovery server is able to invoke the resource managers involved in the commit because the sync point manager was given sufficient information by the resource adapter during registration. This information was provided to the CRR recovery server during commit.

Upon initiation of resynchronization, the CRR recovery server reads its log to determine the state of the commit. From this information, it can determine whether the work should be committed or backed out. Each SFS resource manager is invoked and told whether to commit or to backout.

This procedure works in a TSAF collection and between VM systems connected through a VTAM network. Since all the work was originally initiated from a single virtual machine, only a single CRR recovery server (the one on the requesting user's system) is involved in the resynchronization.

Invocation and interactions

The Coordinated Resource Recovery facility allows applications to participate in coordinated transactions that write data to more than one participating resource in a CMS work unit. This support is invoked implicitly as a result of a CMS request which is destined for an SFS file pool. A CMS work unit can consist of a series of related actions whose changes are treated as a single update. Each group of related actions is called a *coordinated transaction*. Within a coordinated transaction, data may be written to multiple file pools.

SFS participates in CRR by supporting the two-phase commit protocol for synchronization (sync) point requests. This is accomplished by the SFS resource adapter registering with the sync point manager when the first request is made to a file pool server associated with a work unit. In Figure 6, the SFS resource adapter would register SFS work unit x with file pool 1, work unit x with file pool 2, and work unit y with file pool 1 with the sync point manager.

Because the file pool is now registered, when either a commit or backout is issued by the application, the sync point manager drives SFS either to prepare and then commit (or backout if a failure occurs) or to backout. If a failure occurs during sync point processing, SFS participates in resynchronization by setting aside the logical unit of work involved in the sync point process (waiting for the second phase of the two-phase commit). Those logical units of work that have processed the first phase of a two-phase commit are identified as *prepared work*. Prepared work can either be prepared-and-connected work or prepared-and-not-connected work. Prepared-and-con-

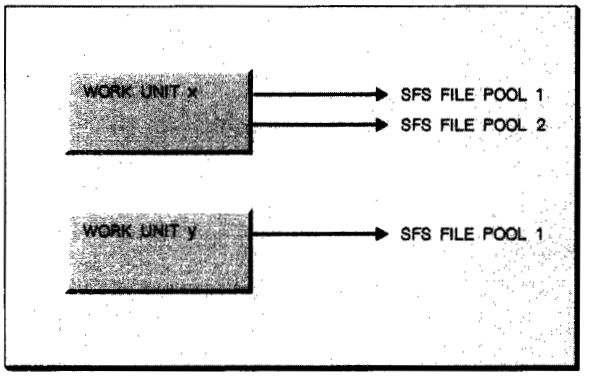
nected work has an active conversation from the user's virtual machine to the SFS file pool server, whereas the prepared-and-not-connected work has lost its conversation (is severed) from the user's virtual machine to the SFS file pool server and now requires resynchronization.

Ordinarily this prepared work is resolved during normal system restart and resynchronization operations. There may be times when the SFS operator needs to intervene and manually synchronize the prepared-and-not-connected work. For example, if for some reason there will be a long delay before resynchronization, then users will not be able to access files or directories that are locked, due to the prepared-and-not-connected work. SFS holds all locks for work that is not committed (including work that is prepared-and-not-connected). For these types of circumstances the SFS operator can manually synchronize the prepared-and-not-connected work by issuing either a FORCE COMMIT or a FORCE BACKOUT command. SFS remembers this heuristic action taken by the operator and when resynchronization direction is finally received, SFS responds according to the direction (commit or backout) given by the operator.

CRR registration. Upon receiving an SFS request for a work unit/file pool server pair that has not been previously registered, the SFS resource adapter registers that work unit/file pool server pair with the sync point manager.

If the registration is successful, the SFS resource adapter sends the request to the SFS resource manager. If, as a result of the request, a logical unit of work was not started in the SFS resource manager, the SFS resource adapter suspends the registration for that work unit/file pool server pair. This suspension leaves the work unit/file pool server pair registered but removes it from participation in coordination by the sync point manager. This saves calls to the SFS resource adapter exits when there is no active work in the SFS resource manager. If, however, a logical unit of work was started on the target file pool, a *recovery token* (a unique identification assigned to the logical unit of work by the SFS resource manager) is returned to the SFS resource adapter along with a recovery transaction program name (TPN). The SFS resource adapter then issues change registration to supply the recovery token and recovery TPN for the registry entry.

Figure 6 Work unit to file pool relationship

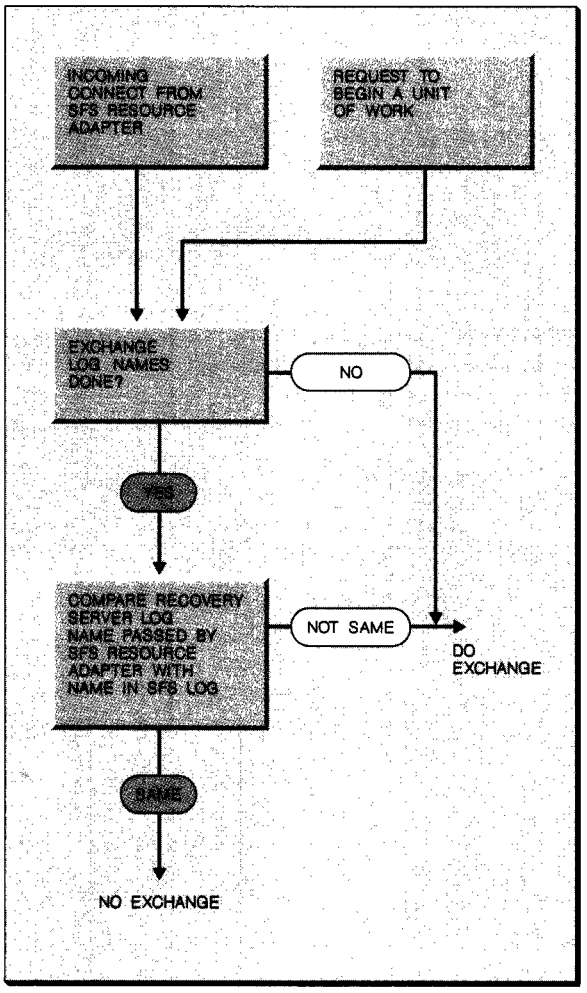


The SFS resource adapter always registers for the end-of-work unit exit. The SFS resource adapter exit may or may not unregister a resource when called for the end-of-work unit exit. If the work unit that is ending is a system work unit (that is, the work unit identification is in the range reserved for system use), the resource is not unregistered. If, however, the work unit that is ending is a user-requested work unit, the resource is unregistered. System work units are commonly used (and reused) work units, whereas user work units are used infrequently. Because of the infrequent use of user work units, it makes sense to tolerate the overhead of unregistering these work units. However, the overhead of the unregister function for system work units is eliminated by suspending them instead of unregistering them.

Exchange log names. To connect to the CRR recovery server, the SFS resource manager needs the CRR recovery server's network logical unit (LU) name and TPN. Because the SFS resource adapter and the CRR recovery server are always on the same processor, the SFS resource manager can use the "connect back" locally known LU name from the extended data presented with the connection pending interrupt during communications. In addition, the TPN of the CRR recovery server is available to the SFS resource adapter through a routine, which it then passes on to the SFS resource manager.

The SFS resource adapter passes the CRR recovery server log name and the CRR recovery server TPN to the SFS resource manager with each request. If the log name is unavailable because the

Figure 7 Exchange log name flow



CRR recovery server is not running, the SFS resource adapter indicates to the SFS resource manager that this request is not participating in CRR (as zeros in place of the name).

When the first conversation is established between an SFS resource adapter and the file pool server, and at the beginning of each logical unit of work, the SFS resource manager is responsible for determining whether an initial exchange log name is required. If the information with the request indicates that the SFS resource adapter is at a level prior to VM/ESA, no exchange is required because that SFS resource adapter cannot participate. If the log name passed by the SFS resource adapter is all zeros, an exchange is not required since the zeros indicate that the CRR recovery server is not available and that the SFS resource adapter is registered as a sole writer to the log. The flow in Figure 7 illustrates the steps to determine if an exchange-log-names is required. As shown above, the SFS resource manager looks for the following:

1. An entry for the CRR recovery server in the SFS resource manager log. Because the SFS resource manager could be accessed by the SFS resource adapter on a particular processor through more than one LU, the SFS resource manager log could have multiple entries containing the same CRR recovery server log name, but each with a different LU name. If an entry is found, but the log name is different from the one passed by the SFS resource adapter, the associated CRR recovery server has started a new log. The SFS resource manager log is considered existing, but the SFS resource manager must initiate an exchange-log-names request to give its own log name to the new CRR recovery server log.
2. A local indication that log names were or were not exchanged. If the local indication shows that a log name exchange has occurred since the SFS resource manager was started, and the CRR recovery server log name received from the SFS resource adapter is the same as the log name saved in the log name table of SFS, then an exchange is not required. If none of the preceding tests indicates that an exchange log name is necessary, then the SFS resource manager proceeds with handling the SFS request. However, if an exchange log name is required, the SFS request is put on hold until log verification is finished. If an error results from exchange-log-names, then the SFS request is not performed and an error code is returned to the SFS resource adapter indicating that coordinated participation could not proceed.
3. The SFS resource manager acting on the response from the CRR recovery server and taking action as shown in Table 1. Note that the SFS resource manager participation in sync points must be denied until the error condition is resolved. The SFS resource manager operator should contact the recovery server operator to determine the reason for the mismatch. Every SFS request whose SFS resource adapter indicates this same recovery server will cause the same exchange-log-names sequence to occur until the problem is corrected.

Table 1 Initial exchange_log_names reply

Recovery Server's Log Status	Reply Function Status	Resource Manager's Log Status	Resource Manager's Actions
COLD or WARM	NORMAL	COLD	The resource manager saves the recovery server's log name.
COLD	NORMAL	WARM	The resource manager checks the log for prepared work that relates to the recovery server. If no records are found, the resource manager saves the new log name from the recovery server. If records are found, the resource manager issues an error message to the operator and deletes the SFS request.
WARM	NORMAL	WARM	The resource manager compares the log name. If the log names are the same, the SFS request is processed. If the log names are not the same, an error message is written and the SFS request is denied.
WARM	ABNORMAL	COLD or WARM	The resource manager issues an error message to the operator and the SFS request is denied.

The exchange-log-names sequence described here is the initial exchange required to ensure the logs match correctly before any new work is initiated. A similar exchange is required during resynchronization and is described in a later section in this paper.

Resource adapter exits. The SFS resource adapter registers and provides support for four resource adapter exits: precoordination exit, coordination exit, postcoordination exit, and end-of-work unit exit. At registration time, the SFS resource adapter passes to the sync point manager the entry point of the routine that will handle the exit function.

Precoordination. The SFS resource adapter processing for precoordination consists of tests to determine whether the work unit/file pool server pair is in the proper state to commit. For example, if an asynchronous request is outstanding for the work unit/file pool server pair, the work unit/file pool server pair will not be able to participate in commit processing.

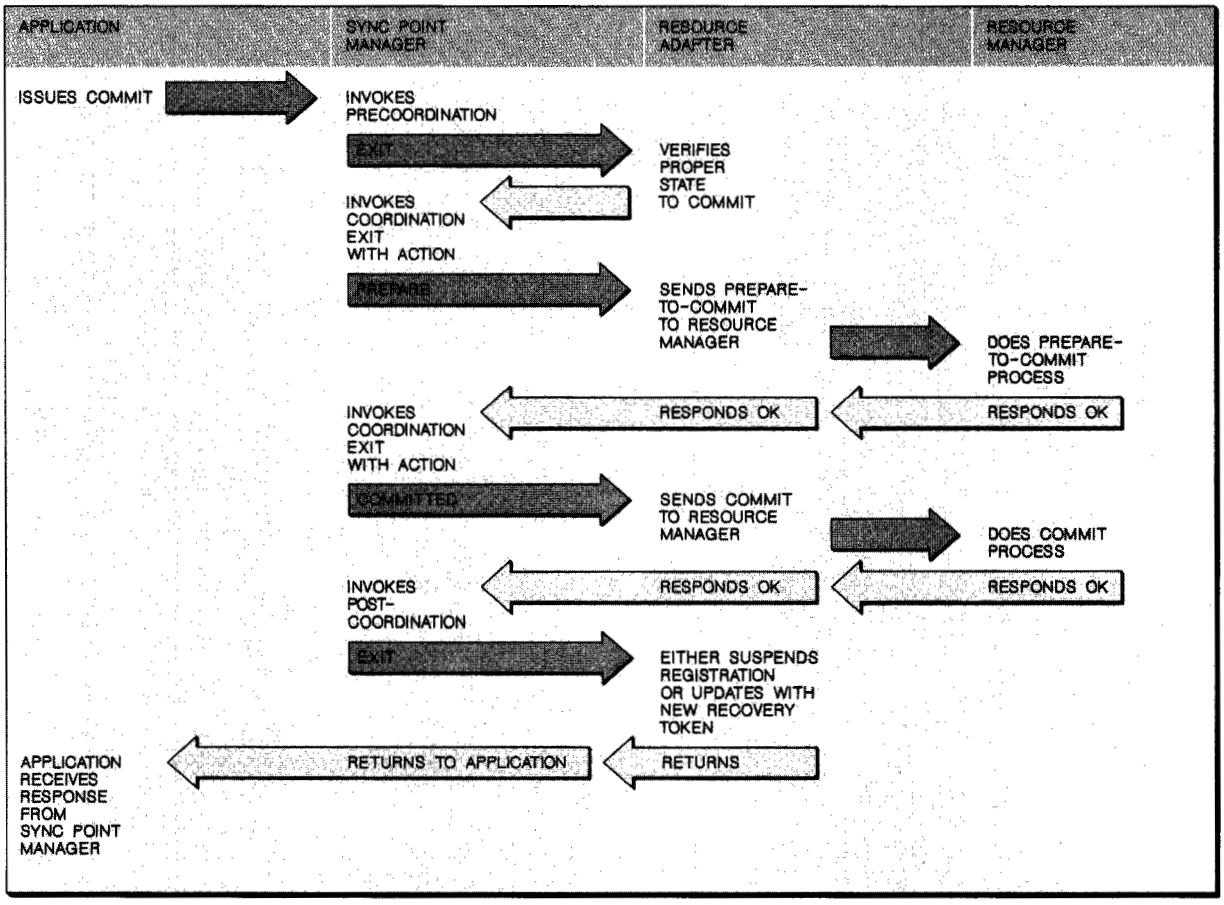
Coordination. The SFS resource adapter processing for coordination depends upon the value of the action parameter.

- For action PREPARE, the adapter exit sends a prepare_to_commit SFS request to the SFS resource manager. Upon return from the SFS resource manager, the adapter exit interprets the return code and passes it on to the sync point manager.
- For action REQUEST_COMMIT and COMMITTED, the adapter exit sends a commit SFS request.
- For action BACKOUT or DEALLOCATE_ABEND, the adapter sends a backout SFS request to the SFS resource manager.
- For action PREPARE_TO_RESYNC, the SFS resource adapter causes the conversation for the given work unit/file pool server pair to be severed.

The adapter exit is called with action OK_BACKOUT if the exit responded with a backout indication to action PREPARE. For action OK_BACKOUT, the adapter exit simply sets the default response and returns to the sync point manager. Figures 8 and 9 illustrate the flow when an application issues a commit or backout.

Postcoordination. The SFS resource adapter processing for postcoordination consists of suspending the given work unit/file pool server pair if, as a result of the sync point that has just occurred, the work unit/file pool server pair does not have an active logical unit of work. If files or directories were open at the time of commit, the work unit/file pool server pair will have an active logical unit of work (commit without close sup-

Figure 8 Flow for commit



port), causing the SFS resource adapter to update the recovery token (the unique identification assigned to the logical unit of work by the SFS resource manager) using change registration.

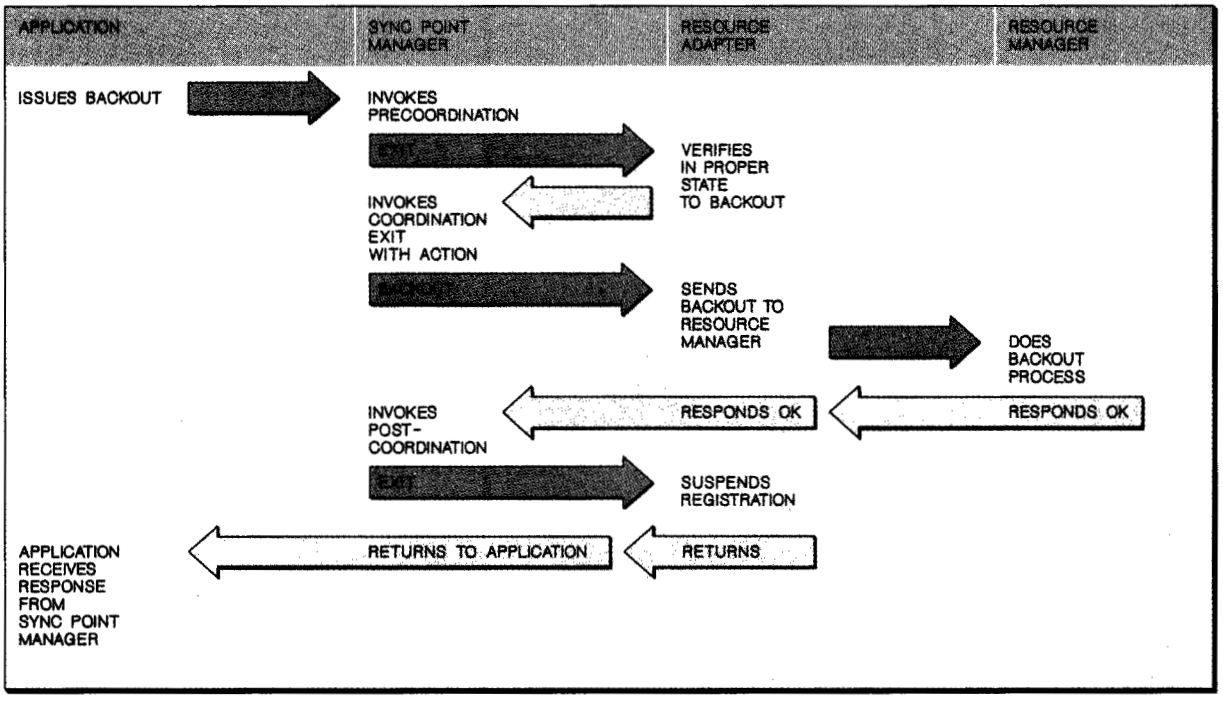
End-of-work unit. The SFS resource adapter processing for end-of-work unit consists primarily of cleaning up registration data for the specified work unit/file pool server pair, which may include unregistering for nonsystem work units. The end-of-work unit exit is executed at end-of-command, abend of the application, and when the application returns the work unit.

Commit with normal end. When the SFS resource adapter coordination exit receives control from the sync point manager with action PREPARE, the SFS resource adapter sends a prepare_to_commit to the SFS resource manager. The SFS resource adapter sends the transaction tag (set by the application) and the global logical unit of work identifier (input from the sync point manager when the coordination exit is invoked) to the SFS resource manager.

In order to support high concurrency, SFS does no updating of its catalog space until the end of the logical unit of work. Therefore, when a prepare_to_commit is received, it must now update all catalog information required by the logical unit of work and then update its log by writing a prepare_to_commit log record. This log record is used in the event of a system failure to rebuild the environment related to the logical unit of work.

During the prepare_to_commit, if no failures occur, then an "ok" response is given to the SFS resource adapter, which it returns to the sync

Figure 9 Flow for backout



point manager. Subsequently, the SFS resource adapter coordination exit should receive an action COMMITTED from the sync point manager. The SFS resource adapter then sends a commit request to the SFS resource manager.

When the SFS resource manager receives the commit request, a commit log record is written. All resources associated with the logical unit of work are released unless files or directories are open (commit without close support). For this case, a new logical unit of work is started implicitly and the locks associated with the open files and directories are held for this new logical unit of work. The SFS resource manager then responds to the SFS resource adapter with an "ok" response, which is passed on to the sync point manager.

The SFS resource manager still supports the one-phase commit process. If a commit request is received which has not been preceded by a prepare, SFS will update all catalog information required by the logical unit of work and then update its log by writing a commit log record.

Backout. Backouts can be initiated several ways. If the SFS logical unit of work was registered, then backout will be coordinated through the sync point manager. One exception to this case is nonrecoverable files. The changes for files with this attribute are committed (not coordinated) whenever the application initiates a backout, the application terminates with a failure, or any other error (such as logoff) that follows implicit temporary "closes" done by CMS for nonrecoverable files. Any recoverable files involved in the logical unit of work are part of a coordinated backout.

Certain errors that occur while processing an SFS request can cause an implicit backout condition. This condition is then signalled to the SFS resource adapter causing the SFS resource adapter to initiate backout processing. The SFS resource adapter first suspends coordination for the work unit/file pool server pair that is already backed out and then it invokes the sync point manager using the interface for the backout function.

In doubt and locks. When the SFS resource manager responds to a prepare_to_commit request indicating that a logical unit of work can either commit or backout (i.e., the resource is "prepared"), locks must be held on all resources used by this

Table 2 Resynchronization exchange_log_names request

Recovery Server's Log Status	Resource Manager's Log Status	Resource Manager's Actions
WARM	COLD	The resource manager does not save the log name of the recovery server and replies cold.
WARM	WARM	The resource manager compares the recovery server's log name. If the log names match, the resource manager ensures the unit of work is not connected and processes the compare_states request. If the log names do not match, the resource manager issues an error message to the operator and indicates an error to the recovery server.

logical unit of work, which is said to be in doubt (prepared). Until the second phase ends, any other logical unit of work that requests those resources held by this logical unit of work will have to wait for them. In order to prevent long delays to other units of work, the prepared logical unit of work that is still connected (and therefore not a likely candidate for resynchronization) will most likely end soon, so the requestor waits for the lock to be released. However, if the prepared logical unit of work is not still connected (resynchronization is required, and therefore a long delay may occur), then the requestor is immediately denied with a special error code indicating that a prepared logical unit of work holds the resource. Once the logical unit of work waits for the resource held by the prepared logical unit of work, if the state of the prepared logical unit of work changes from prepared-and-connected to prepared-and-not-connected, the waiting logical unit of work is immediately denied its request with a special error code.

Note that an interface is available to the user, called "set filewait on," which can be used to control whether or not to wait for resources and has no effect on the processing described above.

Resynchronization. When the CRR recovery server initiates resynchronization to ensure consistent completion of a sync point by all registered resources, an exchange-log-names and compare-states request is received by the SFS resource manager. The following summarizes the sequence of events.

The SFS resource manager receives the resynchronization request that consists of two parts. The first part is exchange_log_names and is processed as shown in Table 2. If the CRR recovery server has initiated an exchange_log_names request, it means that it had information on its log that indicated resynchronization work was pending. If the SFS resource manager finds that its log indicates that log names have never been exchanged with that CRR recovery server, an error situation exists because the logs are now out of sync. Log names can only be saved during an initial exchange.

If the log name exchange was satisfactory, the SFS resource manager processes the second part (compare_states) as shown in Table 3. When the SFS resource manager finishes either a commit or backout, a log record recording that fact is written to the SFS log. The logical unit of work is then immediately "forgotten." An assumption is made that any subsequent direction given by the CRR recovery server notifying to either commit or backout was the direction taken earlier and the response echoes the direction.

Processing the compare_states request also means that a record is written to the SFS log indicating the action (commit or backout), and freeing all resources held by the logical unit of work (including locks).

Resynchronization may be delayed because communication paths are down. During this delay, SFS holds all locks for work that needs to be resynchronized. The CRR recovery server will automatically establish connections to the SFS resource manager and drive the resynchronization process when communication paths are available. However, if an unusual situation occurs where the delay is long, or if the locks relate to critical resources, the SFS operator may need to manually complete the process using a command. SFS provides a FORCE command for this purpose.

Table 3 Compare_states response

LUWID state at resource manager	LUWID* State Sent by Recovery Server	
	Backout	Committed
	Resource Manager's Actions	
Backout or LUWID not found	Send "backout" reply	Send "committed" reply
Prepared	Drive backout and send "backout" reply	Drive commit and send "committed" reply
Committed or LUWID not found	Send "backout" reply	Send "committed" reply
Heuristic backout	Send "heuristic backout" reply	Send "heuristic backout" reply
Heuristic committed	Send "heuristic committed" reply	Send "heuristic committed" reply

*LUWID—logical unit of work identifier

The SFS operator, not knowing the commit or backout state of all other related resources, has a difficult decision to make when using the FORCE command, and commit or backout integrity is exposed. To minimize this exposure, SFS supports the following:

- An SFS operator command that displays information about the state of work. The objective of this command is to give the operator enough information so the administrator of the coordinating system can be contacted to determine whether this task should be committed or backed out.
- SFS remembers whether the operator did a FORCE commit or FORCE backout. Thus, when communication paths are re-established and the automatic resynchronization process is able to take place, SFS is able to continue its participation. If the operator made the correct commit or backout decision, the resynchronization process completes normally. However, if the commit or backout state is incorrect, error messages are displayed and SFS replies to CRR with formal heuristic return codes.
- The FORCE records created at the time of the operator FORCE command is not deleted until the automatic resynchronization process requests the state of this logical unit of work. If an SFS operator knows that no resynchronization will come from the coordinator because of a new log or cleanup from the coordinator site, the operator can delete all FORCE records and the log name associated with a particular logical unit by issuing an ERASE command. If the operator issues the ERASE command and subsequent resynchronization state exchange takes place, the response (nonheuristic) is supplied according to formal rules.

Compatibility

When the Shared File System was introduced to VM, it was understood that it would have to coexist with the CMS file system. Coexistence had to be considered from the interactive user's perspective and from an application's perspective. The key design point for both sets of users is that a Shared File System directory can be accessed, and once accessed, CMS disk (*minidisk*) file system operations and commands can be used to manipulate the data in the Shared File System directory.

In the minidisk file system, the user has to *access* a minidisk before it can be used. When the user accesses a minidisk, the file system is told where to place the minidisk in a linear search order. Each position in the search order is called a *file mode*. Since each file mode is represented by an uppercase alphabetic character (A-Z), the user can have up to 26 accessed minidisks in a search order.

Once a minidisk is accessed, the user manipulates the files on the minidisk through two methods. CMS commands can be used by the interactive user or issued from an application to manipulate

minidisk files. Examples of some CMS commands are RENAME, ERASE, and COPYFILE. For applications, the other method available to manipulate minidisk files is a macro interface, called the file system application programming interface (FS macro API). The FS macro API provides the basic functions of open, read and write, and close. It also provides other functions, to determine if a file exists and to erase a file.

Commit strategy. In VM/SP, the FS macro API was enhanced so that it would work on accessed SFS directories. When the file is in an SFS directory, using the FS macro API will result in CMS performing the operations using the default work unit. The commit strategy for the default work unit is different from the minidisk commit strategy.

In the CMS file system, data integrity and consistency are maintained on a minidisk basis. Stated another way, the minidisk is the commit scope, all files updated on a single minidisk are committed at the same time. In the FS macro API, there is no interface to commit files. A commit of the minidisk happens when the last file open for write on the minidisk is closed.

In the Shared File System, the commit scope is the work unit. All files updated on a single work unit are committed at the same time. SFS contains interfaces to commit changes on the work unit. When using the FS macro API to operate on files in a directory, the changes are made using the default work unit. In the VM/SP Shared File System, a commit strategy was created so that the commits of the default work unit would correspond to the commits of the minidisk for applications using the FS macro API. The commit strategy was that a commit is attempted whenever a file is closed through the compatibility interface. The SFS server will commit the work after it has closed the last file opened on the work unit. If there are still open files on the work unit, the SFS server will not commit any work. This commit strategy is not optimal, since files open for reading prevent the commit if SFS is involved, but do not prevent the commit if only minidisks are involved.

In VM/ESA, the Shared File System commit strategy for the FS macro API was improved with the introduction of the commit-without-close support for Shared File System. This support allows a work unit to be committed if files in an SFS directory are opened for either reading or writing. With this support in place, the commit strategy changes to attempt a commit whenever the last file open for update through the compatibility interface at the file mode is closed. Now, the commit strategy is the same for the FS macro API, whether the files are on a minidisk or in a Shared File System directory.

Another part of the mapping is that a commit is attempted when the last file open for reading through the compatibility interface at the file mode is closed. This commit is not required for compatibility reasons, since there is no concept of a commit of a read-only minidisk. Also, because the commit of the read-only work is handled by CMS, there is no compatibility problem for the application. Read-only work on a work unit must be committed to end the SFS server logical unit of work.

File attributes. In the CMS file system, the file mode is actually composed of two characters, the file mode letter (A-Z) and the file mode number (0-6). As was stated previously, the file mode letter determines the minidisk's placement in the file search order. The file mode number determines the attributes of the file. It is possible to have files with different file mode numbers but the same file mode letter. For example, a minidisk can be accessed as file mode B, and there could be files on the minidisk with file modes B0, B1, B2, and B3.

One of the attributes is the update-in-place attribute. This attribute is associated with file mode number 6. The update-in-place attribute means that the existing records of a file are written back to their previous location on a minidisk, rather than in a new location. An application would use update-in-place for several different reasons, such as:

- Avoiding the need to reaccess a minidisk when there is one writer to the minidisk with concurrent readers, as long as the updates do not change the number of blocks in the file
- Reducing the space utilization when there are many records in large files
- Avoiding out-of-space errors when updating a file
- Allowing multiple writers to update a preformatted file

In VM/SP, the user could create a file in an SFS file pool with a file mode number of 6, but the file would not have the update-in-place attribute. In VM/ESA, to address the requirements for file mode 6 files, the concept of file attributes was added to SFS. These file attributes are associated with a file, they are not associated with a file mode. A file in an SFS file pool has two attributes, the *overwrite attribute* and the *recoverability attribute*. There are two values for the overwrite attribute, "inplace" and "notinplace." There are two values to the recoverability attribute, "recover" and "norecover."

The value of "inplace" for the overwrite attribute for SFS files maps to the update-in-place attribute for minidisk files. The allowable combinations of overwrite attribute values and recoverability values are shown in Figure 10.

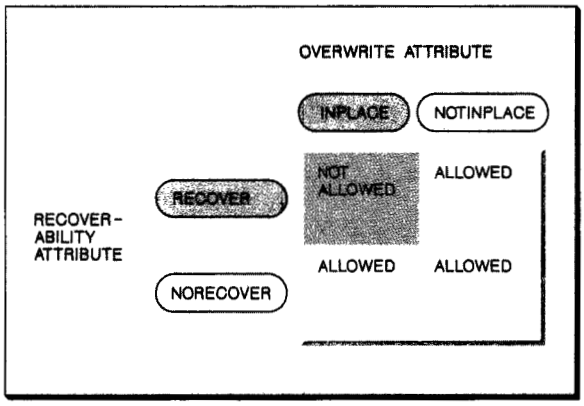
The recoverability attribute determines what happens to changes made to the file if the application issues a backout. Files with the "norecover" value for the recoverability attribute will have their changes committed whether the application issues a commit or a backout, or the application fails with an abnormal termination. This support is useful to applications that are creating files that do not need recovery support, such as log files or audit files.

In VM/SP, if an application wanted to work with recoverable and nonrecoverable files, the application would have to use two work units. All the recoverable work would be on one work unit and all the nonrecoverable work would be on the other work unit. The recoverable work unit would be committed or back out as necessary. The nonrecoverable work unit would always be committed. The introduction of the recoverability attribute simplifies the application's design, since updates to recoverable and nonrecoverable files can be on the same work unit.

Concerns. An application programming interface (API) was introduced in VM/SP to allow the application programmer to manipulate data in the Shared File System. The API was provided as a set of routines, which are packaged as part of library services. The support necessary to manipulate the routines was also introduced.

Commit option support. The API support for Shared File System provides interfaces to do such

Figure 10 Overwrite and recoverability attributes



things as file manipulation, directory manipulation, locking, and authorization functions. The API provides two different methods to commit the changes associated with a work unit. The work may be committed by invoking a commit routine or the work may be committed by specifying a commit option on a routine that supports it. Routines that represent typical commit points for Shared File System applications support the commit option.

The application programmer has a choice to make when closing a file, because there are two ways to commit. The first method is to specify the commit option and the second method is to specify the nocommit option along with a separate invocation of the commit routine.

One of the advantages of specifying the commit option on a SFS function invocation is better performance, because there is only one interaction with the Shared File System server, since the request to commit is "coupled" with the other request. If the commit option is specified, this is reflected by setting a commit indicator in the request that is sent to the SFS resource manager, which carries out the function specified in the request and then commits the work.

In comparison, specifying the nocommit option and following with a commit invocation results in two interactions with the Shared File System server. First, a request is sent to the SFS resource manager to close the file. Then, the sync point manager will perform the commit. Performing the

commit involves the sync point manager notifying the SFS resource adapter, which in turn will send the commit request to the SFS resource manager.

In VM/ESA, it was a requirement that specifying the commit option on an SFS routine resulted in a coordinated commit. It was also known that not every commit had to be a coordinated commit. That is, if an application is coded to the VM/SP SFS API and then run on VM/ESA, it will not require any coordinated commits, since it is dealing with a single write resource manager. For performance reasons, it is better if coupled commits can stay coupled if coordination with multiple resource managers is not required.

Coupled commits are still allowed. When processing the commit option, the SFS resource adapter checks whether there is any other resource in use for the work unit. If there are other resources, such as a protected conversation or another SFS file pool, active for the work unit, the coupled commit performance optimization is not allowed. The SFS resource adapter sends the original request without the commit indicator to the SFS resource manager. Once that request completes, the SFS resource adapter performs a coordinated commit. If there are no other resources active for the work unit, the coupled commit performance optimization is allowed. The SFS resource adapter sends the original request with the commit indicator set to the SFS resource manager.

Atomic operation support. In the SFS, some operations are *atomic*. Atomic operations have the following special rules: there can be no outstanding work when the atomic operation is started, and there is no outstanding work once the atomic operation completes. Following an atomic operation with a commit or backout request is meaningless, there is nothing to commit or back out. Atomic operations are not coordinated by CRR. That is, an atomic operation commits its own work. This commit is not a coordinated commit; only the work done by the atomic operation is committed.

In VM/SP, atomic operations are allowed only when there is no outstanding work for the work unit when the atomic operation is started. In VM/ESA, the rule for when atomic operations are allowed is that there can be no outstanding work for the affected file pool for the work unit. An atomic operation must be directed at a particular file pool. The status of other file pools on the work unit does not determine whether an atomic operation is allowed for a particular work unit.

This change in the rules for atomic operations should not cause any problems for applications written under the VM/SP rule. Those applications should continue to operate successfully; however, this change should simplify the development of applications under VM/ESA.

Future directions and challenges

Directions for distributed data. There is mounting interest in the industry on the general topic of distributed data and their various forms. One of the major challenges is finding ways to access data across an enterprise that is comprised of a network of unlike systems, each with its own unique file system capabilities. As an illustration of this, see Reference 3. This referenced document provides a survey of distributed data capabilities on a number of IBM systems (including VM), and provides some insights into the requirements and direction for distributed data.

While a number of IBM systems provide distributed file capability, few of the systems provide access to file data on other (unlike) systems. For example, SFS remote file access support only works between interconnected VM systems. If a VM application needs to gain access to a Customer Information Control System (CICS) control file on MultipleVirtual Storage (MVS), this requires a different facility. If a VM application requires access to file data on Operating System/400® (OS/400®), or Operating System/2® (OS/2®), the support provided is limited to file transfer services.

The solution to this problem will be defined protocols for file access across the various diverse systems. The direction for distributed file access across Systems Application Architecture™ (SAA™) platforms is the Distributed Data Management architecture.[4] In addition, there are a number of published distributed file access protocols that exist and are relatively popular in the industry. In Transmission Control Program/Internet Protocol (TCP/IP) networks, the SUN Network File System (NFS™) protocol and variations of it are quite popular. In the context of DOS or OS/2 local area network environments, the Server Message Block protocols are quite popular. In the

context of international standards, Open Systems Interconnection (OSI) defines the file transfer, access, and management (FTAM) protocols for file access.

All of the protocols mentioned include some concept and semantics of recovery processing associated with file access. Most of the protocols for this are somewhat limited in scope. It should be noted, however, that the FTAM protocols include support for the standard of the International Organization for Standardization (ISO 9804) for commitment, concurrency, and recovery (CCR), which has concepts similar to those of the SNA architecture for resource recovery. This reflects industry interest in applying resource recovery capabilities on file data.

With the introduction of Shared File System support for Coordinated Resource Recovery in VM/ESA, IBM now has two SAA platforms that support a form of coordinated file recovery in the context of their distributed file support (CMS and CICS in the MVS environment). A logical next step for IBM would be to define the architecture for protocols to coordinate file recovery across unlike file systems.

In this context, it is important to note that SFS support for coordinated file recovery conforms to concepts and architecture defined for SNA resource recovery.[5] Similarly, the CICS recovery support for file control function shipping provides support that is consistent with the SNA architecture. It seems feasible that the recovery capability provided by CMS and CICS could be extended to the broader scope of distributed file access across unlike file systems. It would seem as if this could be done in the context of either (or both) SNA architecture for resource recovery or OSI architecture for CCR.

Cooperative processing considerations. One particular form of distributed data that deserves special attention is the distribution of data between a workstation and a host. File access between a workstation and a host is driven by one of two basic requirements, (1) file serving for the workstation and (2) workstation access to host data.

File serving for workstations involves host server functions that emulate the functions of the workstation's file system. For workstation file serving, there does not appear to be any need for resource recovery support. Few, if any, workstation file systems support the concept of recoverable files. This, in turn, means there are not many workstation applications that require commit (or backout) processing. However, one might expect this to change in the future.

Workstation access to host data, on the other hand, involves a workstation application performing functions on host data which may include access to recoverable files. In this context, the distributed file access services at the workstation must support, or otherwise honor, the recovery protocols for access to those files. If the workstation application attempts to access recoverable file data on multiple hosts, then presumably the workstation services would need to include facilities for initiating some form of coordinated recovery of resources.

There would appear to be two approaches for addressing this requirement. If the workstation only accesses "like hosts," one could envision placing host-specific file recovery services and perhaps even host coordinated recovery services on the workstation. On the other hand, if the workstation needs to deal with multiple "unlike hosts" (with unlike file systems), then one would expect a more universally usable approach with an architecture defined.

In either case, one of the concerns that arises relative to recoverable file access from the workstation, is whether or not a workstation is an appropriate place from which to control resource recovery. Controlling commit and backup from a workstation is not particularly difficult to imagine; however, the prospect of a user workstation supporting and controlling the resynchronization process would seem difficult to achieve. After all, a CRR recovery server logically operates independently of the user or application environment.

In this context, one might expect functions like those of the CRR recovery server to not actually reside on individual user workstations. Instead, these components might be expected to reside on a locally attached (for example, local area network attached) server (either another microprocessor or one of the hosts).

Considerations for device recovery. The CRR support in VM/ESA covers the functions required for *dynamic backout* support and resynchronization.

Dynamic backout refers to the ability to perform commit and backout processing without interrupting service to applications. CRR does not cover synchronization of *forward recovery* or backup and restore processes. The requirement for SFS participation in CRR has no direct implications on Shared File System facilities for backup and restore. The backup and recovery facilities for SFS files will continue to be supported as in VM/SP.

However, the concept of retaining work unit data consistency across direct access storage devices (DASD) restore operations is not new. It is supported by transaction oriented resource managers such as Structured Query Language/Data System or those found in CICS/VSE™ or CICS/MVS™. The support is effected through logging of resource updates and forward recovery processing on DASD recovery.

This section briefly explores the possible future enhancements that might be entertained relative to recoverable CMS files, particularly those that might be participants in CRR applications that also manipulate resources that are subject to forward recovery.

Backup and restore procedures. Traditionally, CMS files have been protected by backup and restore facilities. Unless specifically provided by the application, forward recovery support is not provided. In the context of work unit based commit and backup processing, restore procedures offer an opportunity for application data to be restored to an inconsistent state from the application's point of view. (Restoring data to an inconsistent state could happen with or without Coordinated Resource Recovery or the Shared File System commit support in VM/SP. It can also happen with minidisk files.)

Historically, CMS applications have dealt with this by recommending that all data for the application be stored in the same unit of DASD allocation (that is, a minidisk), such that it would tend to be backed up and restored as a unit. This procedure works quite adequately for the CMS minidisk environment. The situation is less clear in the SFS environment, however, and the CRR environment further complicates the picture.

Relative to the Shared File System environment, an approach that may be pursued is a logical extension of the procedures used for minidisks. The concept is one of file aggregates. A file aggregate is an application-defined collection of files, independent of either their physical placement on the disk or the logical placement in file directories. By providing backup and restore services that operate on file aggregates, an application could protect itself from inconsistent application data as a result of a restore by defining the appropriate file aggregate for the data.

In the context of CRR environments, the situation becomes somewhat more complex. That is, in order for the file aggregate backup and restore procedure to work, it would require that the file aggregate construct be honored across multiple file pools. On a single system with a single backup tool, this could probably be effected without much trouble. However, in the context of multiple file pools across multiple systems, this becomes much more difficult, particularly if the systems are autonomous from a systems management point of view.

In the multiple systems case, some degree of coordination among multiple instances of backup tools is implied. Minimally, they would have to recognize and support common definitions of the cross system file aggregates. A backup or restore operation initiated on any one of the systems would imply a need to initiate the corresponding operations on other systems that own files in the file aggregate. Within a limited configuration of multiple systems, this is probably reasonable. In the context of a loose confederation of systems tied together in a wide area network, however, this may not be practical.

Forward recovery facilities. Another approach that could be taken to assure consistency of application data across DASD recovery procedures would be to support forward recovery for file data. With this approach, all updates to recoverable files would be logged such that on DASD failures, the file data could be restored through the last successful work unit. If all distributed file components provide this level of support, file data across multiple systems could always be kept in a consistent state from the application's point of view.

This approach seems valid and is demonstrated through CICS/MVS-based facilities[6] that support file control recoverable files. However, this level

of support can be expensive in terms of the logging required and the processing required to restore the files in the event of a failure.

In the context of CMS files, such support was considered more support than was needed. With CMS files and CMS file applications, there are two significant considerations. The first is file "replace" versus file "update" activity, where many CMS file applications do not update records of a file. They do whole file replace operations. This would result in some very imposing logging requirements, should SFS attempt to provide forward recovery support. The second consideration is user facilities for archive and retrieval, where CMS files are subject to end user initiated, file-level archival and retrieval functions. Functionally one can think of these as user-driven backup and restore functions. This, of course, has the potential of destroying any application-level consistency that logging may be attempting to effect.

These factors lead one to the conclusion that forward recovery, if provided, should not apply to all CMS files, or even all CMS recoverable files. That is, the customer would want to be selective about what files were subject to forward recovery processing. Similarly, one would expect that files subject to forward recovery processing would probably not be subject to user-driven archival and retrieval functions.

Cited references

1. R. L. Stone, T. S. Nettleship, and J. Curtiss, "VM/ESA CMS Shared File System," *IBM Systems Journal* **30**, No. 1, 52–71 (1991, this issue).
2. B. A. Maslak, J. M. Showalter, and T. J. Szczygielski, "Coordinated Resource Recovery in VM/ESA," *IBM Systems Journal* **30**, No. 1, 72–89 (1991, this issue).
3. *Concepts of Distributed Data*, SC26-4417-0, IBM Corporation (December 1988); available through IBM branch offices.
4. *DDM Architecture General Information*, GC21-9527-1, IBM Corporation (February 1988); available through IBM branch offices.
5. *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808, IBM Corporation; available through IBM branch offices.
6. *CICS VSAM Recovery/MVS Guide*, SH19-6584-1, IBM Corporation (June 1989); available through IBM branch offices.

Cherie C. Barnes *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Ms. Barnes joined IBM in 1973 in the Field Engineering Division. She transferred to Endicott, New York, in 1979 where she worked on DOS/VSE and SQL/DS. In 1986 she transferred to the development team that worked on the design of the Shared File System and on Coordinated Resource Recovery in the VM operating system. She is currently an advisory programmer in VM/ESA design for CMS data management. Ms. Barnes received an IBM Means Service Award in 1975 and 1983, an Outstanding Innovation Award in 1987, and a First Level Invention Award in 1990.

Andrew Coleman *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Mr. Coleman joined IBM in 1979 after receiving a B.S. in mathematics from Wilkes College. He received an M.S. in advanced technology from the State University of New York at Binghamton in 1984. He has worked on SQL/DS test and CSP development and has participated in the design and development of Coordinated Resource Recovery. Currently, he is an advisory programmer working in CMS data management design in the VM operating system. Mr. Coleman received an Outstanding Technical Achievement Award in 1990 for his work on Coordinated Resource Recovery.

James M. Showalter *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Mr. Showalter received a B.S. in mathematics from Clarkson University in 1967 and joined IBM that year. Subsequently, he received an M.S. in computer and information science from Syracuse University in 1979. He has worked on DOS/VSE, OS/VS1, VSAM, QMF for VSE, SQL/DS in VSE and the VM operating systems. Most recently, he worked on the Shared File System and Coordinated Resource Recovery in the VM operating system. He is currently a senior programmer in VM/ESA system design. Mr. Showalter received Division Awards in 1980 and 1983 and a First Level Invention Award in 1990.

Michael L. Walker *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Mr. Walker received an M.S. in mathematics from the University of Illinois in 1970 and joined IBM that year. Early in his career he worked with technology and advanced systems projects in Poughkeepsie, New York, in the areas of database, data communications, and storage management products. He has been at IBM Endicott since 1979 when he joined the SQL/DS development team. More recently, he has held positions in VM planning and VM strategy, specializing in data facilities and SAA support in the VM/ESA system. He is currently a senior programmer in the VM strategy and technologies department. Mr. Walker received awards in 1981, 1987, and 1989 for his work on SQL/DS.

Reprint Order No. G321-5426.