Systems management for Coordinated Resource Recovery

by R. B. Bennett
W. J. Bitner
M. A. Musa
M. K. Ainsworth

Coordinated Resource Recovery is a Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™) function for providing consistency of changes to multiple resources in environments that include distributed applications. It provides a uniform solution for applications to the problem of resource consistency. Systems management of Coordinated Resource Recovery in VM/ESA (CRR) is the set of system services and interfaces that support both automatic and manual procedures for managing CRR installation, performance, and recovery, as well as resource manager and application participation. Much of systems management is focused on application recovery from occasional failures of the procedures for coordinating consistent resource changes. This paper describes several key aspects of CRR systems management, including the CRR recovery log, facilities for minimizing manual intervention when failures occur, performance considerations, and application participation in recovery.

When applications use more than one resource, they must ensure that resource changes are applied consistently, even when applications encounter failures such as the unexpected termination of a resource manager, loss of communications media, or other environmental error. Although most such failures are unusual, the effort to recover from them by restoring consistency between affected resources can be very significant, especially for the growing number of distributed applications. The Coordinated Resource Recovery function of Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™) provides an integrated and uniform solution to this problem. Coordinated Resource Recovery (CRR) in VM/ESA is described in Reference 1.

An important part of CRR are the aids for systems management that support CRR installation, availability, performance monitoring, performance tuning, and recovery. Systems management is most valuable when integrated into the system and fully automatic. After examining these automatic capabilities, we discuss and illustrate the more unusual situations where human intervention may be necessary to manage the system. A future challenge of CRR is to expand the automatic aspects of systems management while simplifying those tasks that cannot be automated. This approach encourages CRR use in a large variety of system environments and resources.

The first half of this paper describes the automated aspects of systems management and discusses the manual systems management tasks

that may need to be performed. The paper concludes with some considerations for future enhancements.

Automated resynchronization

Although failures during application synchronization point (*sync point*) procedures are very unusual, CRR in the VM/ESA system provides automatic recovery (*resychronization*) when such failures occur, completing the sync point to ensure that all resource changes are either committed or backed out. Resynchronization completes the sync point for each failed resource as the resource manager becomes available. Support personnel are kept informed through console messages and are alerted if intervention is required.

CRR builds on the base established by the Customer Information Control System/Virtual Storage Extended (CICS/VSE™) and Customer Information Control System/Enterprise Systems Architecture (CICS/ESA™) systems, as well as Systems Network Architecture Logical Unit (SNA LU) 6.2.[2,3] With the VM/ESA system, CRR is completely integrated and therefore does not introduce additional application environments dedicated to resource recovery processing such as are provided by CICS. While retaining a sync point manager for each execution environment (or virtual machine) integrated with the Conversational Monitor System (CMS), the recovery log and associated recovery server are centralized for each VM/ESA system. The full function of CRR is available to any application running under CMS in a VM/ESA.

The SNA LU 6.2 sync point architecture established a foundation for recovery from sync point failures. In the architecture, and to some extent in CICS, the recovery involves brief attempts to regain communications with participating applications (protected conversations) to complete the failing sync point, but forces commit or backout for sync point participants when they are not available to complete sync points normally. With CRR, both conversations and resources involved in sync point failures are normally automatically recovered through an independent, centralized recovery facility. Periodic and automatic attempts to resynchronize with sync point participants continue until resolution, with a provision for manual intervention in extreme cases.

During sync point recovery, portions of resources may be locked, pending the resolution of the sync point. Automatic recovery minimizes the time that these resources are locked and therefore unavailable to other users or applications. Automatic and centralized recovery facilities minimize administrative requirements. As distributed applications and network systems grow at rates that exceed the availability of qualified systems management professionals, automated recovery, such as that provided by CRR, is essential.

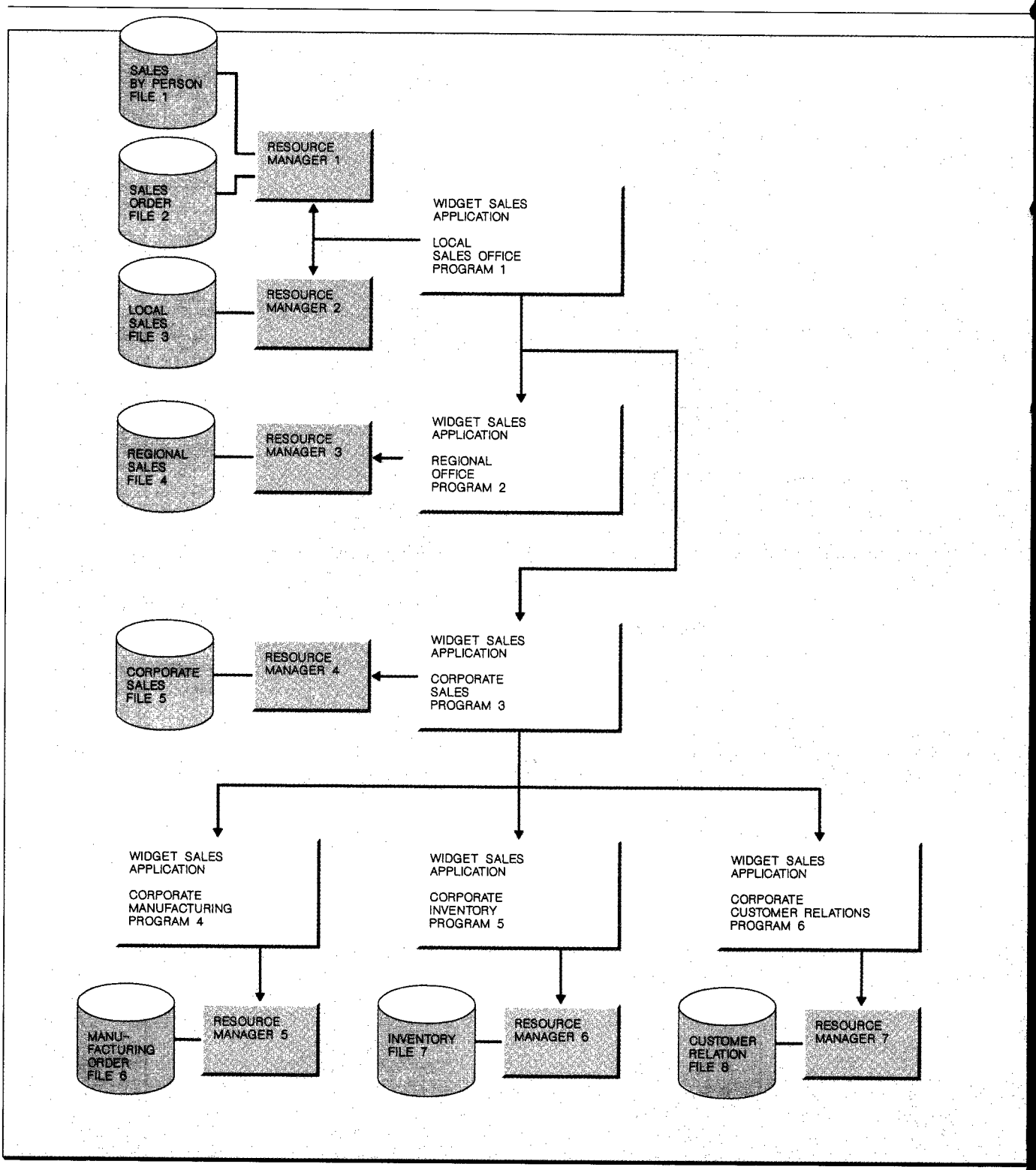
With networks of VM/ESA systems, the resynchronization work is subdivided between the recovery servers of the systems in the network. Each recovery server manages sync point recovery for the applications or portions of applications that reside in their system using the local sync point log information. This provides a reasonable balance between the efficiencies of centralization and the availability benefits of decentralization. Resynchronization work is performed asynchronously by assigning a separate task for managing the recovery of each participant of the sync point, permitting significant overlaps in recovery processing.

The recovery server tasks operate on behalf of corresponding sync point managers to recover the failing sync point by re-establishing communications with affected resource managers, and when necessary, recovery server tasks in other systems. When attempts to establish these communications are delayed, the recovery tasks are temporarily suspended for a timed interval before attempting again. This minimizes use of system resources while attaining automatic recovery goals.

Resychronization includes a provision for notifying a "parent" recovery server task by dependent recovery server task or resource manager when a sync point order (commit or backout) is needed or a sync point response is ready. A recovery server that is suspended until the time for the next retry is immediately taken out of suspension when that participant initiates this notification. By using this notification, relatively long retry intervals are acceptable, minimizing resource consumption required by retry activities and eliminating the systems management task of tuning retry intervals.

The log that provides the recovery capabilities for CRR is logically in two parts containing sync point

Figure 1 An example of a traditional distributed application



information and a list of sync point logs (log names). The sync point log contains information concerning each active sync point, identified by a logical unit of work, such as the status of each locally known sync point participant. The log name portion holds the names of locally known sync point participants' sync point logs. CRR supports a procedure for exchanging log names with sync point participants in order to determine if any of the participants' sync point logs have been replaced. If the exchange indicates that the identity of a log has changed between the time of sync point initiation and sync point failure recovery, CRR maintains integrity by suspending recovery and avoiding use of the invalid log.

An example may help to illustrate the automatic resynchronization of CRR. Figure 1 shows a traditional distributed application called Widget Sales. It involves a local order entry portion (Program 1) with updates to several local files through two resource managers that could be either separate database repositories or completely different database types. Regional sales files are updated through a partner application (Program 2). The local sales transaction also initiates a corporate sales update activity (Program 3), which spins off parallel update activities for three other corporate resources (Programs 4, 5, and 6).

Figure 2 shows a distribution of the programs into five systems. A more complete example would involve multiple local and regional systems, omitted here for simplicity of illustration. Assuming a VM/ESA implementation, each program of the Widget Sales application executes in a separate virtual machine. Local System 1, for example, uses a virtual machine for Program 1 that updates Resources 1 and 2, each of which uses a server virtual machine for its respective resource manager. In this illustration, sync points are assumed to originate from top to bottom, forming a *sync point tree*.

Figure 3 shows the VM/ESA virtual machine structure for Local System 1. Specifically, the local sales application (Program 1) uses the Shared File System (SFS) component of VM/ESA, which implicitly uses the SFS *resource adapter* to communicate with SFS 1 and 2 (Resource Managers 1 and 2) in server virtual machines for updating Files 1, 2, and 3. The local application also uses VM/ESA services to initiate protected conversations with application partners that run remotely in regional and corporate systems. A CMS *protected conversation adapter* handles communication interfaces for the protected conversations. The distributed application uses CRR to ensure consistency of resource changes. Once the application has completed the local portion of the transaction and has propagated work to the "downstream" programs, it initiates a sync point. This invokes the local sync point manager to begin the two-phase commit procedure with its registered sync point participants. The sync point also propagates downstream to the regional and corporate systems. The *prepare-to-commit* (phase 1) and commit (or backout) (phase 2) sync point orders flow down through the tree and the responses flow up through the tree.

The status of each resource manager and conversation participant for this sync point (connected by solid lines in Figure 2) is recorded in a sync point log in the system whose program directly used that resource. For example, Corporate System 1 has a sync point log with the status of the current sync point that includes the sync point states of Resource Manager 4 and Programs 4, 5, and 6. The VM/ESA system also supports remote resources. If Program 3 used Resource Manager 6 (not illustrated), Corporate System 1 would record the status of that participant in its sync point log, even though the resource is remote.

Figure 4 shows that the recovery server, as the maintainer of the sync point log, receives sync point log information from each instance of a sync point manager in the local system, using a local log link. If a failure occurs that prevents normal completion of a sync point, the recovery server is automatically activated to pursue contacts with sync point participants, as dashed lines from recovery servers in Figure 4 show, to complete the interrupted sync point for the sync point manager. The sync point log determines the participants and their status. Individual recovery servers are responsible for only that portion of the sync point tree for which they have log information. This portion is shown in Figure 4 as the immediate leaf nodes of the tree connected by dashed lines to a particular recovery server.

There are several events that could start the recovery process. Figure 4 is the basis for exploring a few sync point failure scenarios:

Figure 2 Systems partitioning of application

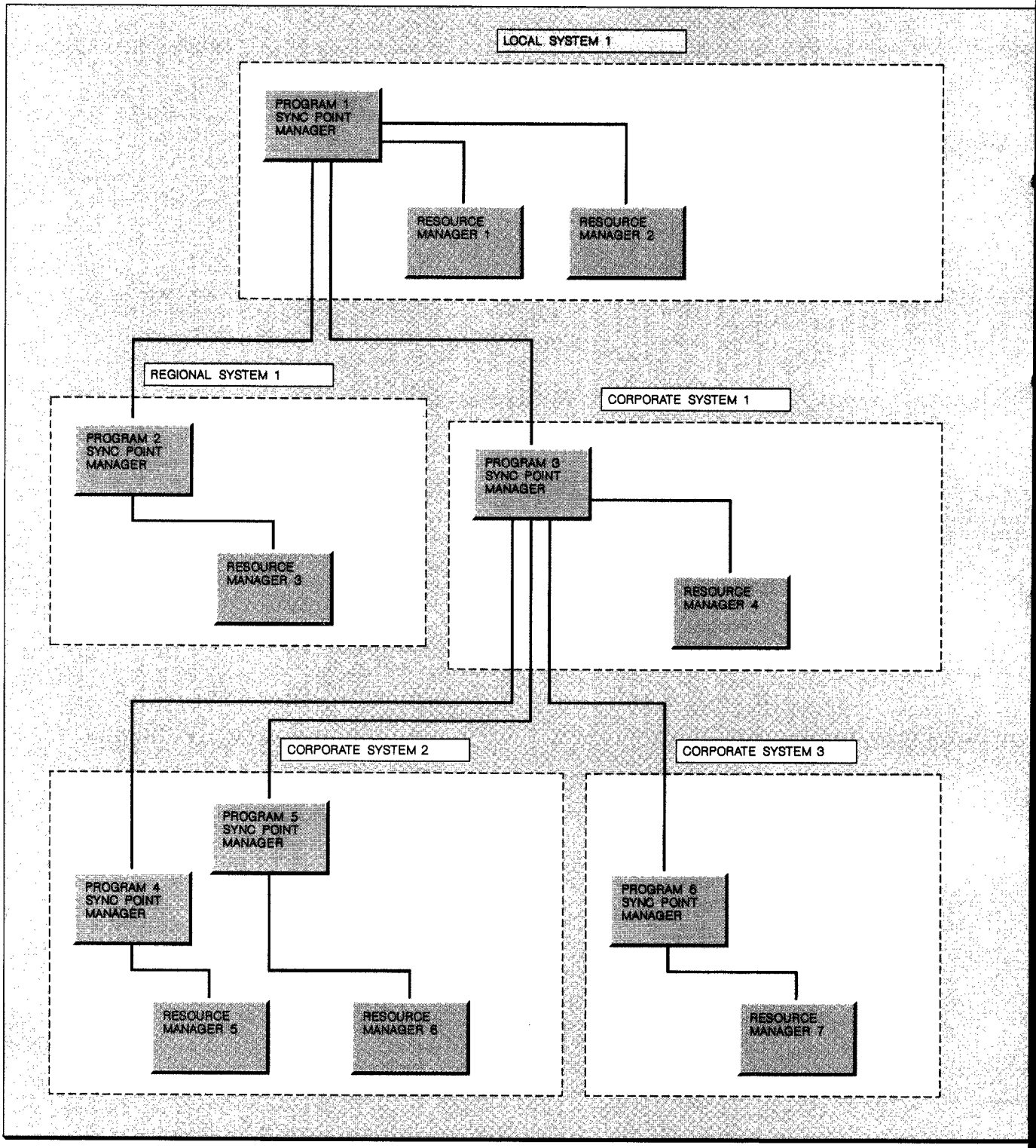
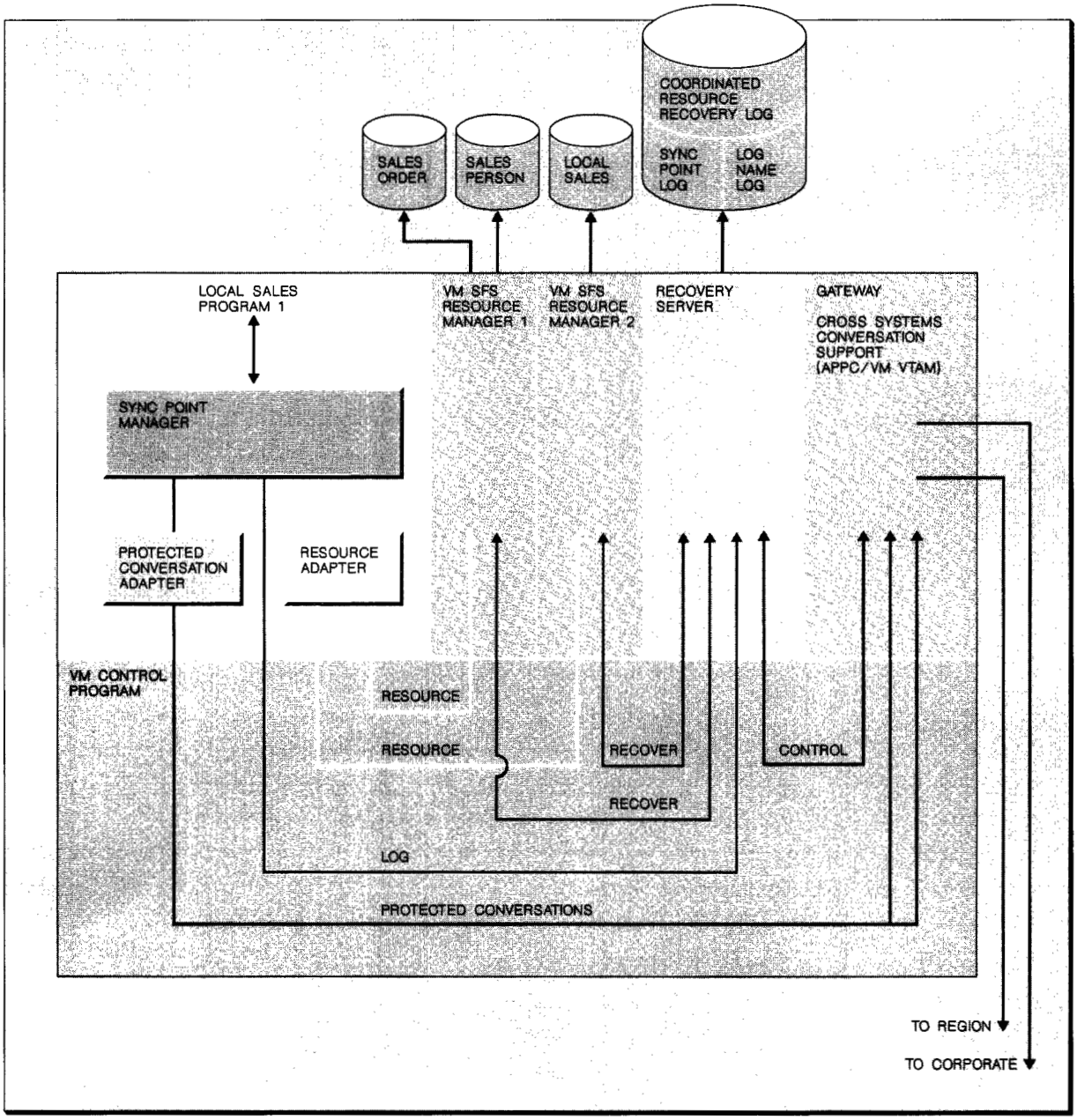
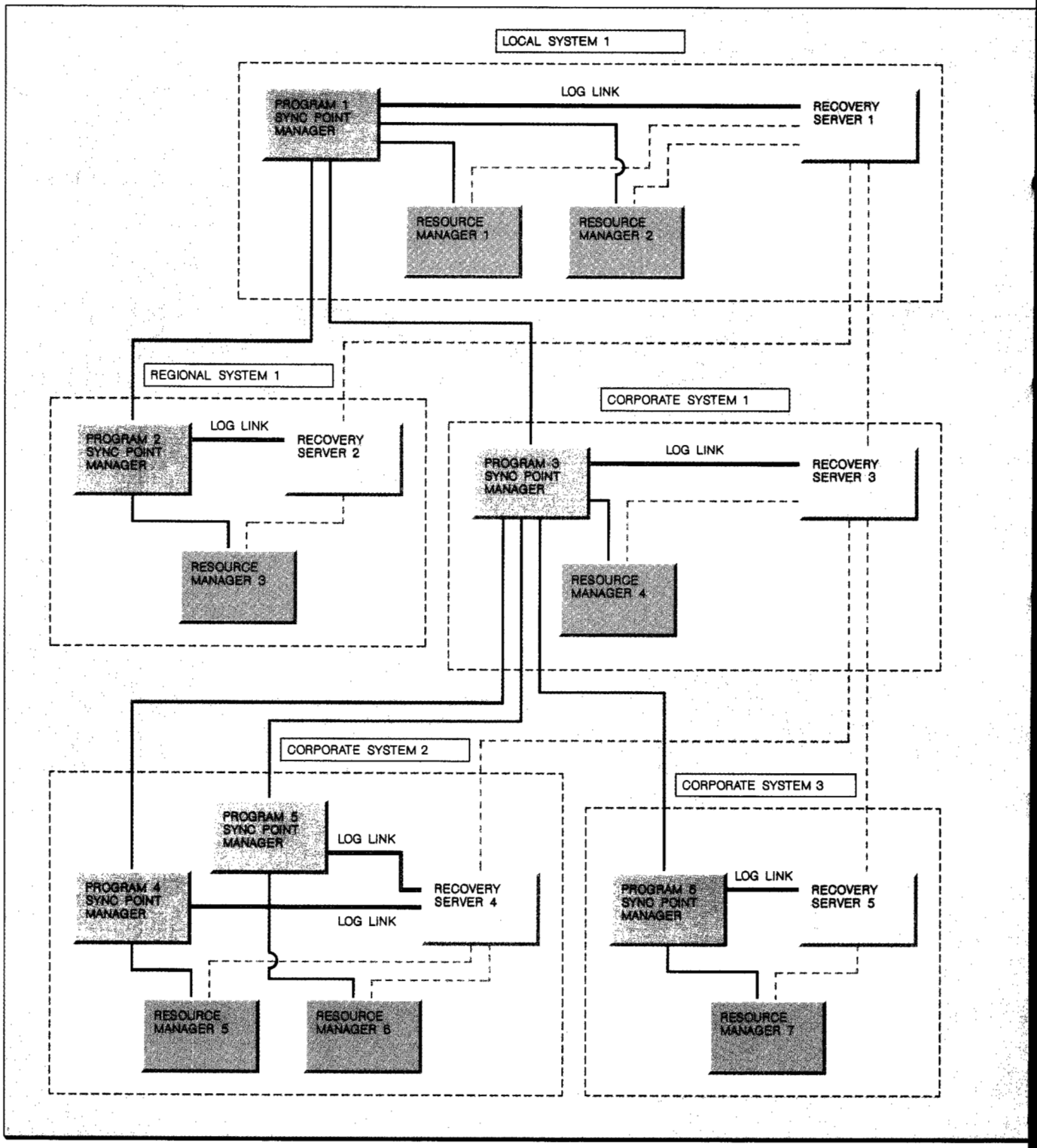


Figure 3 VM/ESA structure of virtual machines for local system



1. Assume that the virtual machine hosting Program 1 fails during a sync point. This results in a failure of the communication link between Program 1 virtual machine and the Recovery Server 1 virtual machine, as well as a failure of the sync point manager for the failing virtual machine. This event causes Recovery Server 1 to begin to recover using the sync point log, involving any active sync points logged for the Program 1 virtual machine. To accomplish this, Recovery Server 1 must resynchronize with Resource Man-

Figure 4 Sync point tree and recovery servers



ager 1, Resource Manager 2, Recovery Server 2, and Recovery Server 3. When the recovery is complete, the results are reported to the Recovery Server 1 operator.

2. Another failure scenario involves a failure of Program 3 during a sync point. As a result, Recovery Server 3 resynchronizes with Resource Manager 4, Recovery Server 4, and Recovery Server 5. In this case, Recovery Server 3 must additionally report its results "upstream" to Recovery Server 1.

3. When a recovery server encounters a terminating failure, its log data are preserved. When the recovery server is restarted, it automatically starts recovery tasks for all sync points in progress at the time of the recovery server failure, represented by log entries in the recovery server's log. Assume that Recovery Server 4 fails or Corporate System 2 fails. When Recovery Server 4 is restarted, it recovers any sync points in progress for Program 4 and Program 5. The recovery server establishes a separate task to recover each participant in each of the sync points for which it has a log entry, recovering multiple sync points in parallel.

4. An application has the option to wait for sync point recovery to complete. If Resource Manager 2 fails during a sync point procedure and the sync point manager for Program 1 detects the failure when it drives the second phase of the commit procedure, the sync point manager remains operational so that it can send a request on the log link to Recovery Server 1, requesting recovery of Resource Manager 2. When the recovery is complete, Recovery Server 1 reports the result to the sync point manager and the application can continue. More typically, an application would not wait but would terminate, allowing resynchronization to resolve the failed sync point independently.

Automated performance optimizations

One important task for systems management is to optimize system performance. This includes minimizing the use of system resources such as central processor units, storage, and physical input/output operations. The objective is to automate systems performance management wherever possible, which is more difficult for CRR since it may operate in a large variety of environments. There are several areas where automatic performance optimizations are available with CRR, including asychronous functions and the recovery server log.

Asynchronous functions. Asynchronous functions are generally more expensive in terms of path length than synchronous functions, but asynchronous functions improve overall response time. The CRR recovery server and sync point manager automatically select the more efficient of the two methods.

The recovery server writes asynchronously to logs to overlap the processing for dual logging and asynchronously processes requests from sync point managers and resource managers. As explained previously, resynchronization includes extensive use of asynchronous processing techniques.

Sync point manager processing involves communications with the recovery server and one or more resource managers. While recovery server communications are synchronous to the sync point manager, resource manager communications are generally asynchronous. The following shows the impact on user response time for the sync point manager coordination of *n* resources:

For synchronous communication,

$$RT = \sum_{n}^{1} \text{time}(n)$$

where RT = response time.

For asynchronous communication,

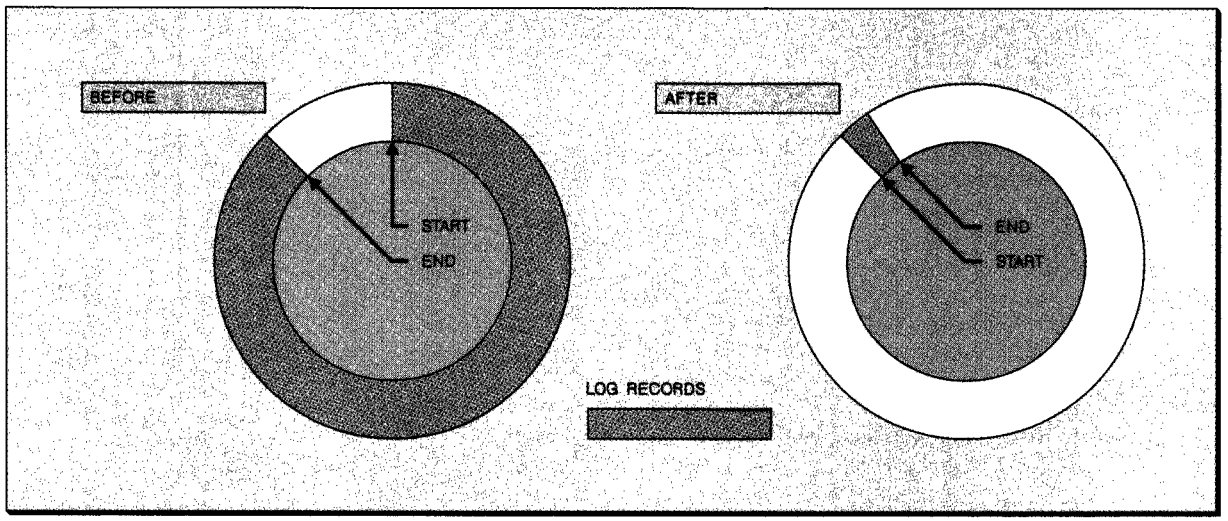
$$RT = MAX(time(1..n)) + n * x$$

where RT = response time and x = overhead for asynchronous functions.

Asynchronous functions let communication response time approach the maximum time to communicate with any one resource manager versus the sum of the time to communicate each resource manager. The communication time may be significant since applications involving CRR may span networks.

Recovery server log optimization. Because all two-phase commits require logging by the recovery server, automatic performance optimizations are

Figure 5 Ring buffer before and after checkpoint procession



particularly beneficial. These involve log compression to optimize disk space utilization and buffering or grouping of log data to reduce the amount of physical input/output (I/O) activity. Log optimizations prevent log I/O processing from becoming a bottleneck, even when sync point processing rates are high. Again, it is important that these performance optimizations do not result in additional manual intervention or jeopardize data integrity. I/O operations involving multiple data blocks are used throughout CRR processing to minimize physical I/O operations.

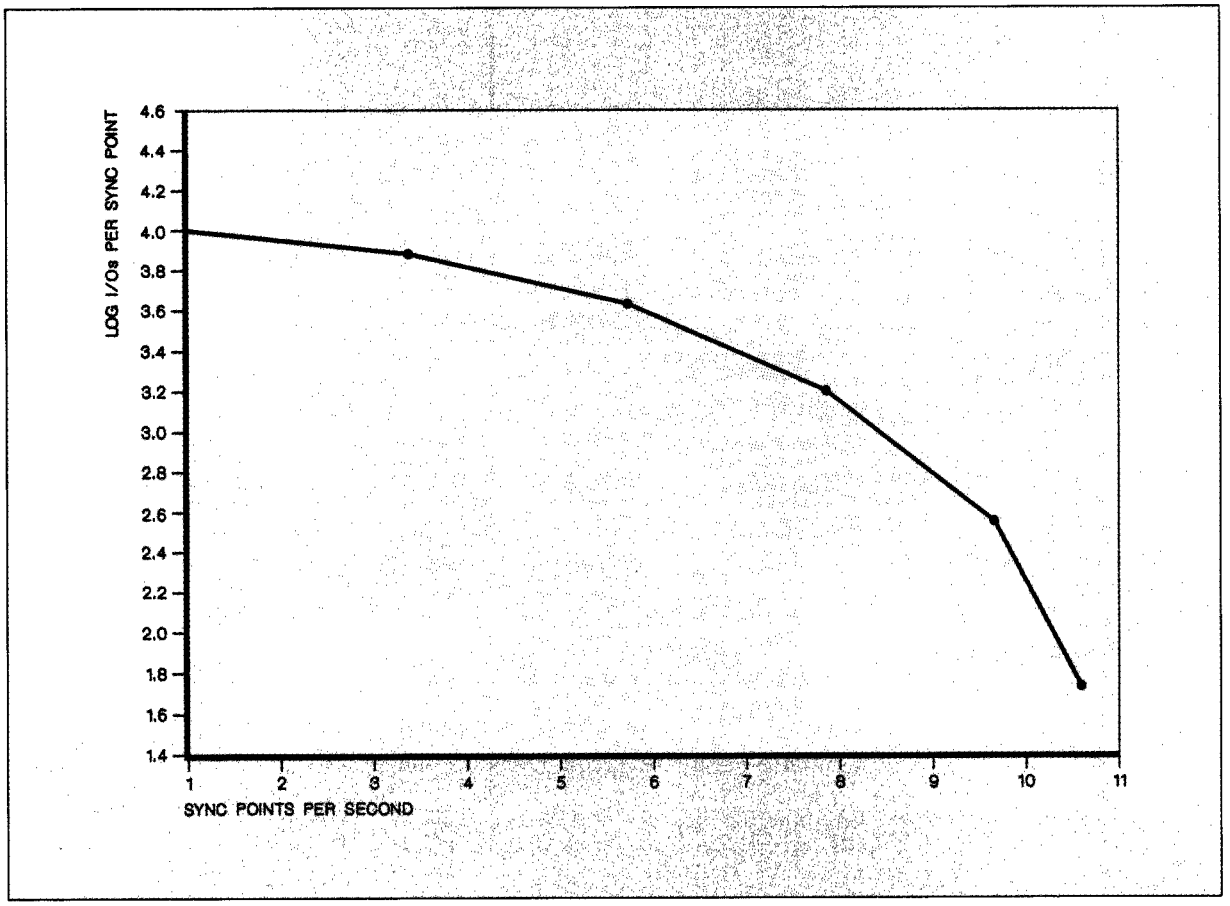
Log compression is a technique for avoiding operational disruption and intervention otherwise required to extend log disk space. Log compression uses a *ring buffer*[4] which is a wraparound method of recording log records on the log disks whereby the beginning of the log space is reused when the end of the area is reached. A checkpoint procedure is automatically initiated before the wraparound occurs, and summarizes all active sync point activity in a new checkpoint log record. Figure 5 shows the ring buffer conceptually, before and after checkpoint processing, and the log condensation that is possible.

The SNA LU 6.2 sync point architecture describes two types of log write requests, *forced* and *nonforced*. Forced log write data must be immediately written to nonvolatile storage (e.g., DASD). Nonforced log data are buffered in virtual storage until a forced log write arrives or the nonforced buffer is filled. This avoids the cost of physical output operations for every log write.

Physical I/O operations are also minimized by grouping data from multiple log write requests into a single physical output operation. Concurrent log write requests from the various sync point managers are collected by the recovery server into a common log write buffer, then written to the log disks as a group. Forced log write requests are not considered complete until the physical log write is finished, but nonforced log write requests complete without requiring an immediate physical write.

Figure 6 shows the benefit that log buffer management provides for physical I/O operations. As the rate of sync point processing increases, the physical I/O log activity per sync point decreases, due to an increase in the amount of log buffering and write grouping. The benefit of these performance optimizations increases with the rate of sync point activity because the recovery server is more likely to receive concurrent requests for log writes. Log output bottlenecks are unlikely except when log write activity approaches the maximum physical output rate of the log device, based on the current data blocking strategy. The graph is for a generalized case. The results vary depending on the environment.

Figure 6 Sync point rate versus physical I/O log operation



Automated availability

Dual logging increases availability of the CRR log by maintaining identical copies of the log on separate physical devices. When one disk becomes inoperable, the recovery server automatically continues by logging sync point data to the functioning disk. At failure time, the recovery server writes a checkpoint record to the working log disk. A message notifies the operator that a log disk has failed. With each subsequent checkpoint, the failed log disk is rechecked for operability. If found to be operable, CRR processing automatically resumes logging to both disks. Because the checkpoint summarizes all outstanding CRR activity to both log disks, manual intervention is not necessary to reactivate the log disk that had failed.

Limp mode, which is the state entered when the recovery server becomes unavailable, lets some applications continue to function. While in limp mode, applications that use resources coordinated by CRR may continue to run, but will get an error return code when attempting to use CRR. This allows for the continued operation of applications that can avoid use of CRR. While there is also some performance degradation associated with running in limp mode, overall availability is improved for those users that are not affected by the loss of the recovery server.

Operational intervention

We have discussed the systems management activities that are completely automated in CRR. To

Table 1 Systems management tasks for resynchronization

Task	Messages	Queries	Transaction Tags	Commands
Monitor the CRR portion of the system	X			
Determine that a problem exists	X	X		
Determine what application is having the problem			X	
Determine the original user of the application			X	
Determine which recovery servers are involved		X		
Determine the state of the resources (committed or backed out)	X	X		
Determine if manual intervention is required	X	X		
Force a commit or backout to occur				X

summarize, CRR automates all of the sync point activity as long as (1) the CRR components are physically available and (2) the protocols that are programmed between components are correct. Failures in these areas are expected to be rare. The remainder of this paper describes some of the manual systems management tasks that occasionally may be necessary.

Establishing or installing a recovery server is a straightforward task, requiring few manual decisions. Once the initial installation process is completed, the CRR recovery server is automatically started when the system is brought up. Although the recovery server shares a virtual machine with an SFS file pool server, the recovery server is installed on an SFS server base that is dedicated to the recovery server facility, enhancing both serviceability and performance. However, the flexibility exists for creating a server machine that acts as both a recovery server and an SFS file pool server.

When recommended installation procedures are followed, frequent CRR performance monitoring and tuning are not normally necessary. Occasionally monitoring may be needed to determine the CRR usage level. Both commands and standard VM/ESA monitoring facilities are available.

CRR provides methods for simplifying operational intervention when problems occur, often avoiding operational disruption. Since CRR resides in an SFS file pool server, it is possible to designate any other SFS file pool server as an alternate recovery server. If there is an extended failure of the primary recovery server or its execution environment, control of the CRR log disks can be transferred to the alternate recovery server as necessary to restore the operation of CRR. As described earlier, CRR continues to operate when one of the two log disks becomes inoperable. CRR supports operational intervention for replacing the failing log without stopping recovery server operations.

The remaining topics address operational intervention considerations for resynchronization. Since resynchronization is normally automatic, this intervention is seldom necessary.

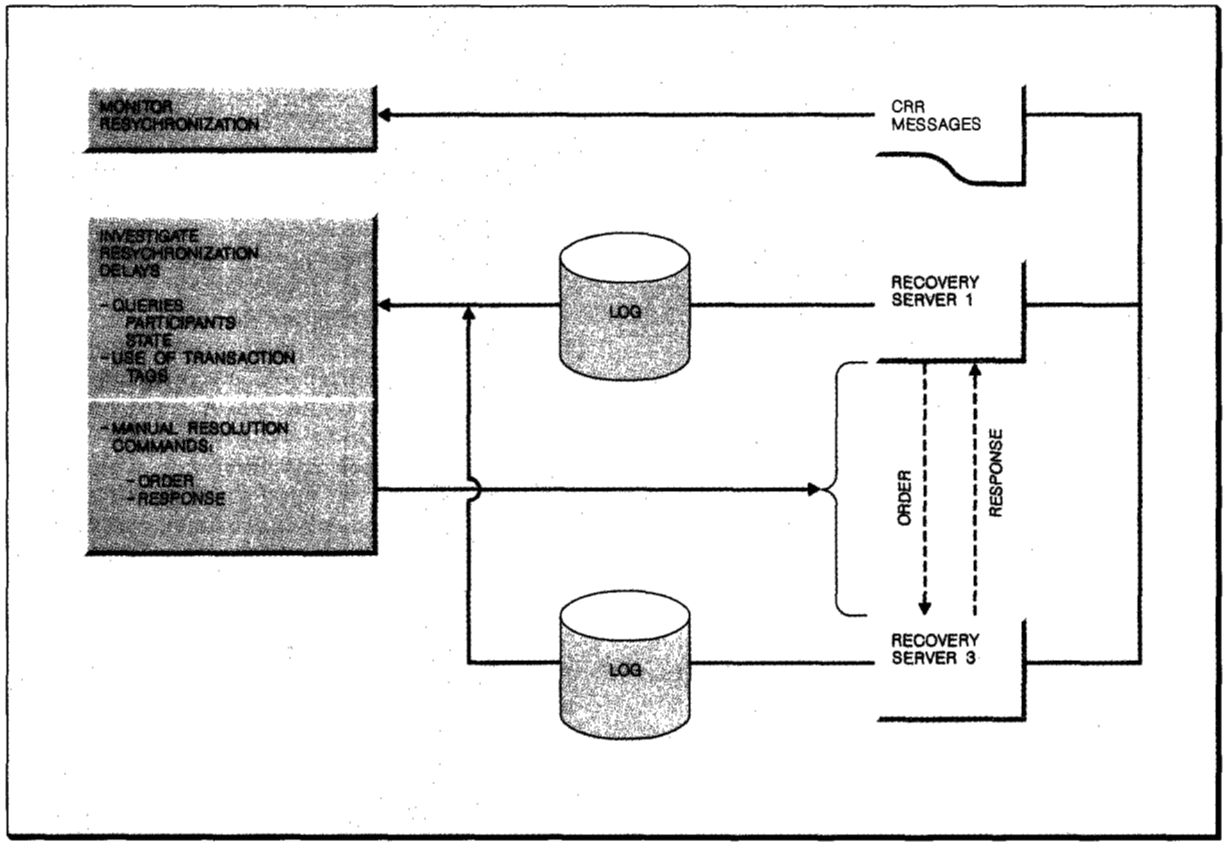
Monitoring resynchronization

Table 1 shows an overview of systems management tasks that may be required for managing unusual cases where automatic resynchronization is delayed or incomplete, along with the categories of tools provided by recovery servers for addressing them. Resource managers that participate in CRR must provide analogous facilities. Referencing Table 1 and Figure 7, messages, queries, transaction tags, and commands are discussed next.

Messages. Monitoring CRR by support personnel assures data integrity of the resources involved, especially when automatic facilities are stalled or otherwise cannot complete a coordinated sync point operation. Recovery servers provide a variety of messages for monitoring the execution of CRR. In some respects, a recovery server monitors itself, because it produces messages whenever abnormal situations arise.

Messages indicate when resynchronization is started for a particular logical unit of work and provide information for support personnel to monitor its progress. This information can be used by support personnel to assist in anticipation of unacceptable delays, prompting investigation

Figure 7 Operational intervention in resynchronization



and occasional intervention in the automatic resychronization process.

If delays in resynchronization are not recognized and addressed promptly, support personnel may receive phone calls or messages from users indicating that data resources are not available or applications are not functioning properly. Data resources locked by an unresolved sync point may not only hold up the sync point participants, but also other users of the affected resources.

In addition to potential delays in resychronization, messages may inform support personnel of *heuristic mixed* conditions. Heuristic mixed conditions occur when some participants in a logical unit of work are committed while other participants are backed out. With automatic resychronization, heuristic mixed conditions should not occur. The condition can occur when a participating resource manager violates recovery protocols or support personnel intervene with an inconsistent and inappropriate action. By screening of resource manager participation and proper training of support personnel, these conditions are normally avoided.

Queries. Along with the messages described previously, a set of queries for information in the recovery server log are available to support personnel where additional information is needed to determine the extent and nature of an unresolved sync point. This information includes identification of the state of the sync point participants, the logical unit of work, and the order (commit or backout) provided by the initiator of the sync point (when available). In addition, a *transaction tag* may be displayed, providing specific application information that can be extremely helpful in sync point resychronization.

The logical unit of work identifier can be quite long in order to accommodate network architecture uniqueness, and would be unwieldy for both reading and key entry in queries. To avoid this problem, CRR queries return a six-character token to represent the logical unit of work identifier locally. This simplifies subsequent queries where the token is substituted for the long logical unit of work identifier.

Queries only return information recorded in the local recovery server sync point log. To investigate logical unit of works that span multiple systems, additional queries are required, as described later.

Transaction tags. Recovery from sync point failures may be significantly simplified through the participation of application programmers. A small investment of time in the early stages of application development has potential for a huge payback in reducing the time for recovery of complex distributed applications. One of the challenges of recovery is associating a failure with the specific elements of the originating application, especially in network environments. CRR lets the application programmer associate application execution elements and sources, such as an invoice number or file name, with particular sync point occurrences. This can provide a distinct association between failure recovery and affected applications. Application participation in recovery simplifies and automates the work of systems management for CRR.

The information included in an 80-byte transaction tag can be generic to the application (e.g., the name of a file that contains the recovery procedure for this application), as well as unique to a particular stage of the execution of the application (e.g., transaction identifier, user account number, or sequence number).

When the transaction tag is provided, it is stored in the recovery server log, where it is available in messages and query responses, providing information to support personnel for associating sync point recovery situations with concurrent application conditions and states. This information makes it possible for support personnel to avoid having to contact application operators or programmers when manual intervention is required for the resolution of inconsistent sync points. In addition, transaction tags may provide support personnel with application directions for manual resolution of rare occurrences of inconsistent sync points. This information could be defined using prose or encoding, agreed upon between application and support personnel. For example, encoding in the transaction tag might indicate that:

- Support personnel should permit automatic resynchronization to be delayed for no more than 10 minutes before beginning manual intervention for this application.
- Support personnel should never manually resolve inconsistencies for this application without a complete understanding of the effect upon all sync point participants, because a heuristic mixed situation could cause intolerable resource repair work.
- Support personnel may intervene with resynchronization for this application by backing out a resource change, when it causes a resynchronization delay, because the particular resource is not especially critical for the application completion.

In the Widget Sales example, a transaction tag may provide such information as the user name, the name of the application (Widget Sales), and the order number. Transaction tags can be passed from program to program along with the other data. In the Widget Sales application, a transaction tag may be established in Program 1 on Local System 1 and then passed to Program 2 and Program 3, until all the participants in the distributed application have information concerning the original sales order. In the case of a failure and a delay in resynchronization, this information could assist support personnel in associating inquiries from application users concerning delays in specific resychronization activities. It could also be very helpful in those rare cases where erroneous manual intervention causes a heuristic mixed situation. If this should happen, the transaction tag information may be used to identify specific resource elements that may need repair.

Commands. CRR includes commands to simulate normal sync point procedures to provide a commit or backout order as a surrogate for an unavailable sync point initiator, or a response as a surrogate for an unavailable sync point participant. These commands may be used when automatic recovery is stalled because of prolonged outages, requiring manual intervention to resolve

Table 2 Status of application data

Recovery Server	Resource Manager	Status	Data Affected
RS1	RM1	Committed	Sales by person
		Committed	Sales order
RS1	RM2	Committed	Local sales
RS2	RM3	Committed	Regional sales
RS3	RM4	Prepared to commit	Corporate sales
RS4	RM5	Prepared to commit	Manufacturing order
RS4	RM6	Prepared to commit	Inventory
RS5	RM7	Prepared to commit	Customer relations

failed sync points. Messages, queries, and occasionally phone calls may be necessary to collect information for formulation of these commands.

To illustrate manual resolution refer to Figure 4. Assume that there is a prolonged communications failure that prevents automatic recovery of a failed sync point in Recovery Server 3 from receiving a commit or backout order from Recovery Server 1, which represents a sync point initiator. Support personnel in Corporate System 1 determine from queries and communications with Local System 1 personnel that the required order from the sync point initiator is to commit, permitting support personnel to simulate the order of the initiator through a CRR command, and allowing the sync point recovery to complete in Corporate System 1. If there were an additional prolonged failure of the communications path between Recovery Server 3 and Recovery Server 4, another CRR command would be needed to simulate the response from Recovery Server 4.

Challenges of a distributed environment

CRR must address many environments from single system to complex networks of distributed applications. Even with a single system, multiple products and resource managers may be involved, possibly including the products of multiple vendors. When manual intervention is required, the skills of several support people may be required because each resource manager may have different external representations and recovery logs. Complexity grows accordingly as networks of homogeneous and heterogeneous systems are used, where an understanding of the entire scope of an unresolved logical unit of work presents a particularly difficult challenge. The goal of systems management in CRR is to make it possible to manage Coordinated Resource Recovery across many environments.

The complexities of distributed systems are best addressed by collecting CRR messages at a central site where queries are used to collect information and make adjustments for remotely located recovery servers. NetView® and VM/ESA facilities are useful in simplifying and consolidating these network management tasks.

To assure data integrity, support personnel must be able to determine the use and status of all resources for a logical unit of work in a distributed application. The logical unit of work for a particular application can be represented in several distributed recovery server logs.

Using the Widget Sales application as an example, messages can be issued from any one of five recovery servers or seven resource managers on five systems. Considering that there may be numerous local systems, the number of systems may be even greater. The possibility of multiple applications for support personnel to manage in a distributed systems environment dramatically compounds the situation and focuses on the importance of using transaction tags to aid in problem determination and correction.

The following example involves a communication failure between Recovery Server 1 on Local System 1 and Recovery Server 3 on Corporate System 1 after all sync point participants have been prepared to commit, i.e., have completed phase one of the two-phase commit procedure. Table 2 describes the status of a sample logical unit of work. Initially the support personnel do not have all of this status information. They may not even know what the Widget Sales application does.

There is no single message or query that gives the support personnel all of this information.

A complex example is chosen to illustrate that sufficient information is available to support operational intervention in the unlikely case where failures occur, and where delays in automatic recovery cannot be tolerated. The example is also the basis for exploring simplifications through the use of transaction tags and proposing some future improvements for support of operational intervention in the automatic recovery process. In the example, distributed systems are managed by central support personnel who must resolve a logical unit of work, assuming that it is not practical to wait for repairs that would permit the process to be completed automatically. CRR may be used in environments where support personnel are centralized, distributed, or a combination thereof. The example based on Table 2 follows:

1. A message routed to the central site indicates that resychronization was attempted, but there was a communications outage that prevented its completion. After being notified of three such retries at 10-minute intervals, support personnel decide to intervene. This decision is based on anticipated delays in fixing the outage, along with the priorities of the participating applications, possibly associated with pressure from users who are unable to access resources that are locked by the incomplete sync point.

2. Support personnel obtain CRR query results for the logical unit of work from the Recovery Server 1 log indicating that: Resource Managers 1 and 2 are committed; Recovery Server 2 is committed; and Recovery Server 3 is prepared to commit, but has not committed.

3. Support personnel query Recovery Server 2 for the logical unit of work providing a more complete picture of the resource managers involved in the logical unit of work. This query tells them that Resource Manager 3 is committed and no additional recovery servers are involved.

4. Support personnel determine that Recovery Server 3 is operational, therefore the failure is with communications facilities between Recovery Servers 1 and 3. If the communication link failure prevents a query from the central site, then a query at the local node would be necessary. Support personnel query the Recovery Server 3 sync point log for the logical unit of work indicating that: Resource Manager 4 is prepared to commit and Recovery Servers 4 and 5 are prepared to commit.

5. Support personnel query Recovery Server 4 for the logical unit of work indicating that: Resource Managers 5 and 6 are prepared to commit and no further Recovery Servers are involved.

6. Support personnel query Recovery Server 5 for the logical unit of work indicating that: Resource Manager 7 is prepared to commit.

7. With this information (see Table 2), support personnel are ready to initiate commands to override the automatic recovery process and avoid a heuristic mixed condition. Support personnel determine that Recovery Server 3 should be driven with a commit order to complete the sync point for the selected logical unit of work. This is accomplished with a CRR operator command for Recovery Server 3, manually supplying the commit order that would have originated from Recovery Server 1. The commit is automatically propagated to Resource Manager 4, Recovery Server 4, and Recovery Server 5.

8. The support personnel also initiate a CRR command that manually simulates the response from Recovery Server 3 to Recovery Server 1, which indicates that the logical unit of work is committed. This results in the automatic generation of messages that indicate the completion of resynchronization and the resolution of the sync point for the logical unit of work with a commit of all resources. This permits normal completion of the logical unit of work with minimal manual intervention and without waiting for repair of the failing systems communications facilities. Resources held up by the failure are promptly freed. This avoids the inconsistencies that can result when resources are freed by individual resource managers, unilaterally forcing sync point resolution without complete knowledge or cooperation of other participants and the appropriate system management support facilities.

The above example is useful for demonstrating a very complex recovery scenario for which there are possibilities and challenges for improvement. For example, a transaction tag could refer the support personnel to a Widget Sales application recovery procedure document that specifies exactly which queries are necessary, thereby simplifying the analysis. Another possibility is that a transaction tag could indicate that it is normally acceptable for the application to wait for automatic recovery to complete, avoiding the manual intervention except for extreme cases (outages in excess of some specific time period).

There is also an opportunity for the customization of network tools such as NetView for filtering messages and automatically gathering information through queries.

Future considerations for CRR

We have discussed several of the innovations of CRR involving automatic recovery, operational override that avoids heuristic mixed conditions for cases where automatic recovery is delayed by prolonged systems failures, and assistance for communication between applications and systems support personnel. Yet there are still some interesting challenges for future extensions of CRR.

Improve and expand recovery aids. As environments using CRR grow in complexity, tools are needed that can automatically monitor resynchronization, analyze the status of a logical unit of work across all affected systems, and report the results to a designated central facility when conditions justify operational intervention. It would also be helpful to assist in isolating the specific portion of a resource that is affected by an incomplete sync point. Recovery aids should be generalized to support many heterogeneous environments to avoid restricting the scope of CRR.

Expand the transaction tag capability. Currently the transaction tag is supported only by CRR in VM/ESA. Where an application updates data on other systems, the transaction tag information should be available on all systems involved with a logical unit of work identifier.

Extend CRR to heterogeneous environments. CRR was designed to support and participate in networks of heterogeneous systems with an open-ended set of resource managers. Although the SNA LU 6.2 sync point architecture provides considerable guidance for accomplishing this, there are some areas that are beyond the architecture, especially for resource manager participation. Although CRR in the VM/ESA system, along with SFS participation, has provided a model for extension to other environments and systems, it remains a challenge to maintain this uniformity as CRR participation grows to systems other than VM/ESA.

Provide audit trails. Many applications provide a complete record of changes in an audit trail. CICS provides this capability through an application program interface for journaling. Distributed applications have the same requirement, except that the audit trail for a single logical unit of work may be distributed across many systems. A tool for assisting the support personnel in problem determination would permit combining all of the audit trails for a single logical unit of work.

Summary

The data processing industry has experienced a rapid growth of capable resource managers to satisfy a variety of user requirements. Often applications are unable to complete their work without using a number of independently managed resources, requiring efficient coordination for consistent changes across the set of resources. With distributed applications, the challenges of coordination increase even more.

CRR in VM/ESA provides an integrated, consistent, and uniform solution to this problem and follows an architecture for extension to many other systems as well. The solution offers an alternative for applications that have previously had to develop their own individual solutions to the problems of resource coordination, at considerable expense and duplication.

We have examined several aspects of systems management for CRR in VM/ESA that reduce the need for manual intervention in recovery, availability, and performance optimizations. In addition, there are occasionally situations where intervention is still necessary while avoiding heuristic mixed conditions. There is also opportunity for application programmers and operators to participate by providing application-specific information to simplify manual involvement in recovery. Finally, there is a clear need for addi-

tional tools to assist in systems management and further automate the coordination of resources, especially as distributed applications grow in complexity.

Cited references

1. 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).
2. *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808, IBM Corporation; available through IBM branch offices.
3. J. P. Gray and T. B. NcNeill, "SNA Multiple-System Networking," *IBM Systems Journal* **18**, No. 2, 263–297 (1979).
4. J. Gray, P. McJones, M. Blasgen, R. Lorie, T. Price, F. Putzolu, and I. Traiger, *The Recovery Manager of a Data Management System*, Research Report RJ-2623, IBM Research Division, 650 Harry Road, San Jose, CA 95120 (August 15, 1979).

General references

C. C. Barnes, A. Coleman, J. M. Showalter, M. L. Walker, "VM/ESA Support for Coordinated Recovery of Files," *IBM Systems Journal* **30**, No. 1, 107–125 (1991, this issue).

M. W. Blasgen et al., "System R: An Architectural Overview," *IBM Systems Journal* **20**, No. 1, 41–62 (1981).

R. A. Crus, "Data Recovery in IBM Database 2," *IBM Systems Journal* **23**, No. 2, 178–188 (1984).

J. Gray, *Notes on Data Base Operating Systems*, Research Report RJ-2188, IBM Research Division, 650 Harry Road, San Jose, CA 95120 (February 23, 1978).

J. Gray, P. McJones, M. Blasgen, R. Lorie, T. Price, F. Putzolu, and I. Traiger, *The Recovery Manager of a Data Management System*, Research Report RJ-2623, IBM Research Division, 650 Harry Road, San Jose, CA 95120 (August 15, 1979).

B. G. Lindsay et al., *Computation and Communications in R*: A Distributed Database Manager*, Research Report RJ-3740, IBM Research Division, 650 Harry Road, San Jose, CA 95120 (January 6, 1983).

SAA Common Programming Interface Communication Reference, SC26-4399, IBM Corporation; available through IBM branch offices.

SAA Common Programming Interface Resource Recovery Reference, SC31-6821, IBM Corporation; available through IBM branch offices.

A. L. Scheer, "SAA Distributed Processing," *IBM Systems Journal* **27**, No. 3, 370–383 (1988).

VM/ESA CMS Administration Reference, SC24-5445, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Application Development Guide, SC24-5450, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Application Development Reference, SC24-5451, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Planning and Administration Guide, SC24-5445, IBM Corporation; available through IBM branch offices.

VM/ESA CMS General Information, GC24-5440, IBM Corporation; available through IBM branch offices.

R. Williams et al., *R*: An Overview of the Architecture*, Research Report RJ-3325, IBM Research Division, 650 Harry Road, San Jose, CA 95120 (December 2, 1981).

R. Bradley Bennett *IBM Data Systems Division, P.O. Box 6, Endicott, New York 13760.* Mr. Bennett is currently an advisory programmer in data management design for VM. He joined IBM in 1961 as a systems engineer at Syracuse, New York. In 1968, he transferred to the Field System Center at Syracuse as a complex systems design consultant, and in 1970 he transferred to Endicott where he has worked in the design and development of advanced systems, VSE/VSAM, and SQL/DS. He received an IBM Outstanding Innovation Award in 1987 for design work on the Shared File System of VM/ESA. He has received an IBM Second Level Invention Achievement Award for patent work. He graduated from Kenyon College in 1957 where he received a B.A. in economics.

William J. Bitner *IBM Data Systems Division, P.O. Box 6, Endicott, New York 13760.* Mr. Bitner joined IBM in 1985. He is currently a senior associate programmer in the VM/ESA performance evaluation department. In 1990 he received an IBM Outstanding Technical Achievement Award for his contributions to VM/ESA performance. Mr. Bitner received a B.S. in computer science from the University of Pittsburgh at Johnstown.

Mark A. Musa *IBM Data Systems Division, P.O. Box 6, Endicott, New York 13760.* Mr. Musa is currently a staff programmer in VM/ESA systems analysis, where he works on the functionality and usability of VM/ESA user interface designs. He joined IBM at Endicott in 1987 and worked in the VM/SP systems analysis area. He received a B.S. with majors in mathematics and computer science from the State University of New York at Brockport in 1976 and a M.S. in computer science from the State University of New York at Stony Brook in 1977. Prior to joining IBM, Mr. Musa worked for Texas Instruments in Dallas, Texas, and the Link Flight Simulation Division of Singer Corporation in Binghamton, New York.

Michael K. Ainsworth *IBM Data Systems Division, P.O. Box 6, Endicott, New York 13760.* Mr. Ainsworth joined IBM in 1984 after earning a M.S. in computer science from Indiana University in Bloomington. Mr. Ainsworth is currently a senior associate programmer and has worked on the design and development of the VM/ESA Shared File System, Coordinated Resource Recovery, and, most recently, installation and migration tools for VM/ESA.

Reprint Order No. G321-5425.