



Beyond Restart and Concurrency:

*z/OS System Extensions for
Restartable Batch Applications*

Beyond Restart and Concurrency:

z/OS System Extensions for Restartable Batch Applications

Introduction

This White Paper examines the challenges facing organizations with mission critical z/OS batch workloads. It presents restart capability for z/OS batch applications as a strategic investment in infrastructure and describes the benefits to be derived from restart enablement. It considers the requirements and characteristics of a generic restart solution along several dimensions to facilitate a rigorous evaluation and selection process.

A system-wide restart facility provides a fault tolerant execution environment for z/OS batch applications

As the enterprise extends its hours of operation steadily towards a "24 x 7" environment, there is less and less time to tolerate failures in z/OS batch production. A robust and reliable restart facility minimizes the impact of z/OS batch application failures and prevents them from escalating into business problems. Investing z/OS batch applications with restart capability is a proven means to leverage existing IT assets and optimize the applications already in place. A system-wide restart facility provides a resilient "safety net" for z/OS batch applications and an essential foundation for z/OS batch application processing.

This White Paper discusses the following topics in sequence:

- Description of a system-wide restart facility
- Implementing restartable z/OS batch applications
- Benefits of restartable operation
- Evaluating a restart solution

System-wide Restart Facility: Description

A z/OS software extension supports batch application restart on a system wide basis

A system-wide restart facility that is suitable for the organization's entire portfolio of z/OS batch applications is implemented as an extension to the z/OS operating system. It can track and manage the state of the z/OS batch application throughout its processing and provides an architected solution that is resilient, scalable and manageable. Such a facility creates a fault tolerant execution environment on which additional capabilities (described in subsequent sections) can be layered.

Checkpoint/Restart is an enabling technology that allows an application to restart from a checkpoint (near the point of failure) with all its resources in a consistent state. This requires that the restart solution:

- reposition sequentially accessed input and output files
- reposition DB2 cursors
- log VSAM I/O so uncommitted changes can be backed out if necessary
- save (selected areas of) application storage at checkpoint time and restore them at restart time
- inform the application as to whether an initial or restart run is in progress

The commit and rollback services provided by products like DB2 and IMS are only partial solutions. Successful restart requires a more comprehensive unit-of-work whose scope encompasses all the resources the application may access. Not just changes to DB2, IMS, WebSphere MQ and other RRS compliant resources -- but changes to application storage, VSAM updates and sequential file and DB2 cursor position as well.

The restart facility manages "orphan" resources and keeps them in sync with RRS compliant resources like DB2 and IMS

The system-wide restart facility functions as the resource manager for these "orphan" resources and keeps them "in sync" with resources managed by formal MVS subsystems like DB2 and IMS. When a "restartable-unit-of-work" is checkpointed, it establishes a point of consistency from which processing can successfully resume upon restart, as illustrated by the green arrow in Figure 1. Upon restart, this application can resume its execution from a checkpoint (near the point of failure) with all its resources in a consistent state. This eliminates redundant processing and ensures processing always proceeds in a forward direction. There is much less wasted processing because only work performed since the last successful checkpoint is backed out and discarded, not the entire run.

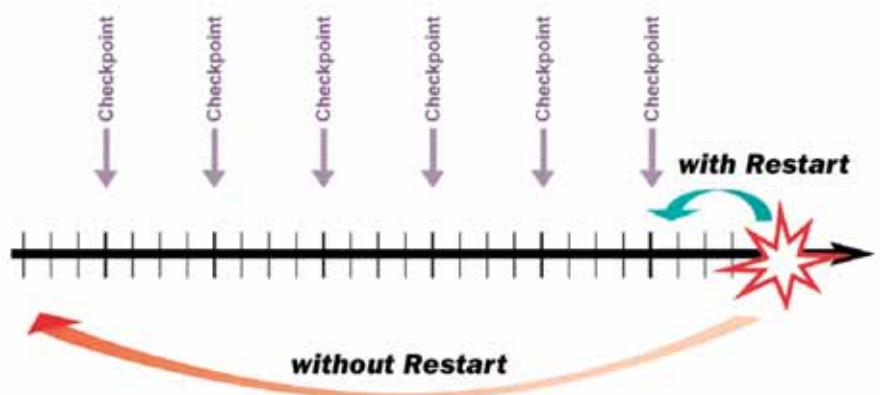


Figure 1 - Restart capability enables a z/OS batch application to restart from the last checkpoint

A repositionable sequential access method can outperform QSAM by 10%

A system-wide restart facility can reposition sequential files (and sequentially accessed VSAM datasets) during restart using optimized, parallel I/O techniques that outperform the QSAM access method used by COBOL by roughly 10%. As a result, z/OS batch applications endowed with restart capability

exhibit run times close to native processing times. Moreover, from a wall clock standpoint, z/OS batch applications that fail and require restart finish processing sooner than their non-restartable counterparts (because they resume execution from near the point of failure, rather than from the beginning).

Implementing Restartable z/OS Batch Applications

Enabling an application for restartable operation can considerably complicate development. By some estimates, restart logic can double an application's size and significantly impair one's ability to comprehend and maintain it. Moreover, in the absence of a generic restart solution, each application must make its own provisions for restart.

Although restart poses numerous technical problems, a generic restart solution enables developers to delegate these responsibilities to reliable system software. Since the system-wide restart facility handles the mechanics of checkpoint/restart, developers can concentrate on defining the logical unit-of-work that successful restart requires.

Successful restart requires a Unit-of-Work whose scope encompasses all resources the application accesses

A system-wide restart facility makes it as easy as possible to endow both new and existing z/OS batch applications with restart capability by using the application's own commit logic (i.e. standard checkpointing requests like SQL COMMIT and IMS CHKP) to drive its more comprehensive checkpointing services. These extend the scope of a Unit-of-Work to encompass all resources the application accesses and ensure they remain consistent and 'in sync'.

It is the developers responsibility to identify the application's Unit-of-Work and request a checkpoint when it is complete. This is not a task that can be delegated to the restart facility since it knows neither the application's requirements, nor when application data is in a consistent state. What the restart solution can do is establish a point-of-consistency from which processing can resume if restart is required. The developer must also code a flow of control that resumes execution (in the event of restart) at a Unit-of-Work boundary.

Programmers code standard COBOL OPEN, CLOSE, READ and WRITE verbs to perform repositionable I/O

The Unit-of-Work can be comprised of one or more transactions since each transaction constitutes a point of consistency. Batch applications typically bunch multiple transactions within a single Unit-of-Work and process them within a loop. For the sake of clarity, the Unit-of-Work should be defined within the high level logic of the application's main source module. "Main" modules should always be "restart enabled" whereas subroutines need not be, unless they contain storage which must be saved and restored (a practice which is not encouraged).

A system-wide restart facility can process I/O to sequential files and VSAM datasets using standard compiler I/O statements. This allows COBOL programmers, for example, to code standard SELECT and FD statements and use familiar OPEN, CLOSE, READ, WRITE and START verbs to access both sequential files and VSAM datasets. In the event restart is required, the restart facility will automatically reposition sequentially accessed files when driven by a native compiler statement like OPEN.

Developers can optionally identify specific storage areas to be saved at checkpoint time and restored at restart time. Application storage can often be saved in its entirety (which requires no specification) unless it is extremely large and non-volatile.

Note: Some restart facilities permit the Unit-of-Work to be defined (and a checkpoint request to be triggered) external to the application source code (such as instream with the JCL). This can be useful if source code is unavailable or cannot be modified.

Benefits of Restartable Operation

A restart facility implemented as an extension to the operating system can improve the performance, availability and operational efficiency of the entire z/OS batch workload. It can also

- Enhance data integrity, availability and quality
- Increase business flexibility and improve service quality
- Complement existing business resiliency and DR processes
- Promote compliance with requirements for auditing and disaster recovery as mandated by Sarbanes Oxley legislation

The following sections examine these benefits in further detail.

Recovery and Restart Benefits

The failure of a critical z/OS batch application can halt activity, delay decisions and disrupt downstream users and processes that depend on the application's timely completion. Although failures and abends will inevitably occur, restart capability mitigates their impact with facilities that:

- provide standardized and simplified procedures for restart
- eliminate the time consuming back out of changes (which can take twice as long as the original processing)
- greatly reduce the need to back out and redo successfully processed work
- completely eliminate the need to back out and reprocess committed work

Restart capability minimizes the impact of z/OS batch application failure

- provide automated error handling and recovery
- reduce the need to back up shared files, DB2 tables, IMS databases and WebSphere message queues -- to the extent precautionary copies are made to allow job restart
- permit the same JCL to be used in both initial and restart runs (which eliminates the errors and delays associated with modifying JCL for restart)

Concurrency Benefits

For enterprises with high availability requirements, restart capability is not just insurance, but a prerequisite for concurrency

In the past, recovery from z/OS batch application failure meant restoring data to its original state from backup copies, correcting the error and rerunning the application from the beginning. This is no longer feasible in a shared data environment because restoring data to a point in time prior to running the failed batch application will overlay changes made by online transactions and other batch applications that were active while the failed application was running. Since these concurrently active processes access the same shared data as the failing z/OS batch application, their committed changes are wiped out by a point-in-time restore. In contrast, restart enabled applications are well behaved in a shared data environment. They promote concurrent access and allow batch and online processing to coexist. They also:

- release locks that impact concurrency and degrade response times
- reduce batch window requirements (or eliminate them entirely)
- enable z/OS batch applications to schedule and execute at any time
- reduce deadlocks, timeouts and "resource unavailable" conditions as a cause of batch application failure

Note: Committed changes become available immediately to other processes. In contrast, a failing application's uncommitted changes never become visible.

Automatic Unit-of-Work Retry

"Temporary" failures due to resource contention are conducive to automatic restart and Unit-of-Work retry. A system-wide restart facility permits SQL-CODE values such as -911, as well as selected IMS status codes and/or user abends to be predefined as eligible for "retry". When the restart facility detects a "retry" condition, it can back out data and processing to the previous point of consistency and wait a site-defined interval of time. It can then restart the application automatically -- without manual intervention or re-submission of JCL. This allows the application to retry the failing Unit-of-Work in its entirety and can further reduce timeouts, deadlocks and "resource unavailable" conditions as a cause of failure.

Checkpoint frequency represents a tradeoff between speed and concurrency

"Checkpoint pacing" facilities allow the commit frequency to be adjusted dynamically during execution -- without changing the application. Both the checkpoint frequency and interval may be modified based on factors such as elapsed time, number of records processed, time of day, heuristic code and/or operator request. Different checkpoint frequencies can also be pre-defined for separate shifts and take effect automatically.

The number of logical checkpoint requests received from the application before triggering a physical checkpoint can be varied as well. The physical checkpoint frequency in effect represents a tradeoff between speed (less frequent physical checkpoints) and concurrency (more frequent physical checkpoints). This allows applications to request a commit after every Unit-of-Work -- without concern for performance.

The ability to adjust the physical checkpoint frequency can be useful in a variety of situations. For example, does a batch application running beyond its batch window hold so many locks that it causes contention with online transactions? Rather than cancel it, the application can be directed to take more frequent checkpoints.

Workload Scheduling and Switching Benefits

By halting an application at a Unit-of-Work boundary, there are no changes to back out and no processing to discard

A system-wide restart facility provides additional benefits within the context of workload scheduling and process migration. Such a facility enhances z/OS job scheduling products that provide step restart capability by enabling a restartable z/OS batch application to resume execution from a checkpoint within a jobstep - something a job scheduler alone cannot do.

A system-wide restart facility can gracefully halt an application at a point-of-consistency in its processing. For example, an application running beyond its batch window can be quiesced and resumed at a more convenient time. Since execution halts at a Unit-of-Work boundary, there is no wasted processing or time consuming back out of uncommitted changes.

An entire workload can be quiesced should a subsystem or LPAR be subject to a planned outage (such as a shutdown to conduct routine maintenance, upgrade hardware, etc.). In such a scenario, the restart facility can gracefully halt each restartable application within the workload at a Unit-of-Work boundary. Workload execution can then resume on another MVS image within the Sysplex and access another member of the DB2 data sharing group.

Restart is implemented in system software rather than separately in each application

Development Benefits

A system-wide restart facility shields developers from the complexities of restart and provides an easy to use methodology that simplifies application development. Such a facility

- Eliminates the need to implement restart logic separately in each application. z/OS batch programs are thus smaller, easier to understand, faster to develop and less prone to bugs.
- Handles the mechanics of checkpoint/restart so developers can concentrate on defining the logical unit-of-work that successful restart requires
- Provides a comprehensive set of services through a consistent, problem state API
- Requires only minimal changes to application source code (and often no changes at all)
- Supports applications written in COBOL, PL/I, C/C++ and Assembler languages
- Supports all RRS compliant resource managers (such as DB2, IMS, WebSphere MQ and Transactional VSAM)
- Supports restartable applications comprised of multiple load modules and/or program objects. These executables may in turn be comprised of multiple, independently written source modules.

Restart capability helps meet or exceed Service Level Agreements

Operational Benefits

A system-wide restart facility enhances operations by speeding up recovery times and minimizing the risk of delays and service disruptions. The enterprise benefits through restart facilities that:

- Provide a single system image and single point of control throughout the Sysplex
- Reduce or eliminate restart errors
- Orchestrate automatic recovery actions when failures occur
- Help meet SLA's with the ability to set (and dynamically change) batch execution priorities
- Enable the same JCL to be used for both initial and restart runs (which greatly reduces the possibility of error in the manual process of updating JCL for restart)
- Allow restartable applications to be tuned in accordance with established practices
- Provide monitoring and control facilities for the entire workload of restartable z/OS batch applications

- Permit database utilities that require exclusive control of a resource to run free of contention
- Predict completion times so "behind schedule" warnings can be issued and remedial action can take place
- Prevent batch applications that should be restarted from being cold-started by mistake
- Provide a system programming interface (SPI) for use by job schedulers, database utilities and other authorized programs

Choosing a Restart Solution

The restart solution should integrate seamlessly into your z/OS environment

In evaluating a system-wide restart facility, IT managers and technicians should consider whether the solution:

- Works without intrusive system "hooks" that can make z/OS service and upgrades problematic
- Provides "day-one" support for new z/OS releases and major subsystems like DB2, IMS and WebSphere MQ
- Provides forward and backward compatibility with all releases of IBM software whose IBM support status is "current"
- Ensures the restart environment will not "break" if service is applied or a z/OS upgrade occurs
- Scales from a single mainframe to the most complex, multi LPAR Sysplex configurations and data sharing environments
- Supports all RRS compliant resource managers -- including DB2, IMS and WebSphere MQ
- Supports batch jobs initiated under JES or WLM control
- Supports the latest DFSMS facilities like extended addressing volumes (EAV)
- Supports widely used SMS features like compressed and extended format datasets
- Interoperates with DFSMS Advanced Copy Services that create dynamic and "point-in-time" copies of data
- Limits access to authorized restart functions via standard RACF, ACF2 and TopSecret definitions

A system-wide restart solution provides additional features and synergistic benefits

In addition, evaluators should determine if the restart solution:

- Allows restart from near the point of failure - after abends, recompiles or system IPLs -- with all resources in a consistent state
- Implements restart functionality in system software rather than within the application program

- Provides common and simplified operating procedures for restart
- Enables the same JCL to be used for both initial and restart runs
- Provides checkpoint pacing so applications can commit after every Unit-of-Work without concern for performance
- Allows failing programs to be changed and recompiled if necessary, and then restarted
- Supports automatic restart and Unit-of-Work retry after "resource unavailable" conditions like SQLCODE -911
- Repositions DB2 cursors automatically
- Enables applications without commit logic to take checkpoints at Unit-of-Work boundaries that are defined external to the application source code
- Exploits z/OS Parallel Sysplex capabilities to facilitate workload switching and process migration
- Provides facilities to monitor restartable applications, predict their completion times and govern their operation
- Allows restart support to be temporarily disabled or permanently removed should the need arise, on an application or site-wide basis
- Provides useful diagnostics such as compile date, compile options, record count and record contents at abend time
- Interoperates with z/OS job schedulers (such as those from IBM, CA and BMC) and provides additional function
- Fully supports diagnostic tools and debugging aids such as XPEDITER®, the IBM Debug Tool and Abend AID®

If a vendor supplied solution is being considered, evaluators should determine whether the vendor is committed to the solution's successful implementation and deployment. Does the vendor enhance the software with regularly scheduled releases and provide ongoing service and support of the highest caliber? Can the organization leverage the experience, methodology and best practices gained from the vendor's successful implementations?

Conclusion

As 24 x 7 access to data and applications increasingly becomes the rule and an unconstrained batch window becomes the exception, the enterprise requires a robust and reliable restart capability to:

- Ensure data integrity, availability and recoverability
- Minimize the impact of z/OS batch application failures and keep the enterprise on schedule
- Significantly reduce downtime and lower the risk of business disruption
- Shield the business as well as its partners and customers from the effects of batch application failure
- Improve batch job elapsed times

- Enhance the disaster recovery preparedness of the enterprise
- Promote concurrency, ease batch window constraints and improve business resilience

Restart enablement requires little or no change to source code

A system-wide restart solution provides a proven method for enhancing z/OS batch workloads in an evolutionary, non-disruptive fashion. With only minimal changes to source code, it avoids the risk, time and expense of a "rip-and-replace" approach. The nightly batch cycle becomes more predictable, reliable and secure -- and takes less time. The enterprise as a whole benefits from shortened recovery times and improved service levels without impacting existing processes, procedures or communities of users. The enterprise realizes significant operational benefits along with simplified development and maintenance.

With a growing recognition of the risks to which the enterprise is vulnerable, management must:

- prepare for problems, preferably before they occur
- cope with pervasive uncertainty, risk and an exploding number of contingencies
- envision disaster and failure scenarios that were unthinkable a short time ago
- prepare for both planned and unplanned outages
- devise procedures to detect, analyze and solve problems

A restart solution delivers fast, proven return on investment

Restart capability is consistent with all these objectives. It plays a critical role in leveraging the legacy portfolio of z/OS batch applications that are already in place. Restart is increasingly recognized as a simple and cost-effective solution with proven return on investment characteristics. Since timing can make a big difference in the ROI calculation, management must decide when to make such an investment: whether to act now or "wait and see". As restart capability becomes pervasive in the z/OS environment, its strategic rationale becomes increasingly evident and the business case more compelling by the day.

About Relational Architects International

Relational Architects International (RAI) has provided best of breed solutions for DB2 and z/OS since 1987. Its quality products and superb support have made it a trusted partner in z/OS enterprise computing. Smart/RESTART from RAI is a robust and reliable restart solution with a proven record of supporting mission-critical z/OS batch processing for the Global 2000. Over 15 years of development and refinement are invested in Smart/RESTART to make it the seasoned and mature solution it is today. To learn more about running your z/OS batch workload reliably, securely and with complete data integrity, visit www.relarc.com/smartrestart or call 1-800-776-0771.

z/OS, DB2 for z/OS, IMS and WebSphere MQ are software products of International Business Machines Corporation. Smart/RESTART and Smart/RRSAF are trademarks of Relational Architects Intl. Xpeditor and ABEND-AID are registered trademarks of Compuware Corp. Any other trademarks contained herein are the property of their respective owners.