Saving Private Info

V5R4 delivers functionality and performance enhancements to the journaling facility.

By Robert Andrews

Journaling is a critical function for many System i5* customers. Journaling allows the System i5 platform to perform commitment control operations and object recovery in event of an outage, as well as provide detailed auditing records showing changes to the data. In this article, I'll discuss many of the journaling facility's changes and new features introduced in i5/OS* V5R4.

Maximum Objects per Journal

With V5R4, journals can now support up to 10 million objects per journal, up from 250,000 at V5R3. This is especially important for shops using popular ERP packages that often have libraries nearing the former 250,000-object limit and for shops using packages that tend to create new byte stream files on the fly, since such packages regularly attempt to create millions of such byte stream file objects. To allow a journal to grow beyond 250,000 objects, the new Journal Object Limit (JRNOBJLMT) parameter must be set to *MAX10M on either the Create Journal (CRTJRN) or Change Journal (CHGJRN) command. Once set to *MAX10M, the object limit can't be set back to *MAX250K. Also, once set to *MAX10M, the associated journal receivers can't be saved and restored to or remote journaled to a target system lower than V5R4. There are no appreciable performance problems related to simply setting a journal to be capable of growing to 10 million journal objects. However, the journal IPL recovery processing times for such a journal grow exponential, not linearly, when the number of journaled objects in need of recovery grows to a high value.

On the initial Work with Journal Attributes (WRKJRNA) screen, a summary of the number of items journaled and the maximum for that journal is shown in the right hand column. To help manage and effectively view the larger number of objects journaled, a new option was added to the WRKJRNA, Display Journaled Objects (F19) screen. Option 40 will now show a summary of the count of each object type that's journaled. Keep in mind some subtle internal journaled objects such as journal receivers and

some SMAPP-protected access paths aren't listed on this screen so the sum may not add up to the total on the first page.

QDFTJRN Enhancements

The QDFTJRN data area introduced with V5R3 is enhanced at V5R4. It allows users to control the automatic journaling of new database files when being created into a schema or library. With V5R4, the data area has been enhanced to support not only database files and SQL tables but also data queues and data areas. Also, instead of initiating journaling for such objects only at create time, these objects can be set to be automatically journaled when restored to or moved into the library or schema. When creating the QDFTJRN data area, the first 10 characters (positions 1 to 10) must be the library of the journal in uppercase and the second 10 characters (positions 11 to 20) must be the journal name in uppercase. What follows are sets of tuples of data types and operations. Each of the two elements of the tuples must be in uppercase and on a 10-byte boundary (positions x1 to (x+1)0 – i.e., 21 to 30). The first element is the data type: *FILE, *DTAARA, *DTAQ, *ALL or *NONE (prevents automatic journaling even in schemas). The second element is the triggering option: *CREATE, *MOVE, *RESTORE or *ALLOPR.

Change Receiver Threshold

Prior to V5R4, if the threshold of a journal receiver needed to be changed, a new journal receiver with the new threshold needed to be manually created with Create Journal Receiver (CRTJRNRCV) and then attached to the journal using CHGJRN command. With V5R4, this can be done at the same time using CHGJRN only. By using CHGJRN with a Journal Receiver (JRNRCV) parameter set to Generate (*GEN) and the new Journal Receiver Threshold (THRESHOLD) parameter set to a valid value, the newly generated journal receiver is created with the new threshold value instead of inheriting the old value. This saves the hassle of manually having to create the journal receiver first.

Minimized Entry Specific Data

When a journaled file is changed, one or two copies of that record are written to the journal. Often times, only one or a few fields are changed inside that record, which causes lots of additional space to be taken up by non-changed fields in the entry-specific data section. By turning on minimized journal entries support for files, the system saves only the information from fields that changed. However, this data was broken up on internal byte boundaries and compressed, making it unreadable to humans. This lack of auditability prior to V5R4 of the record changes often forced users to not use this functionality.

In order to make minimized entry specific data more attractive, a new style of minimizing is available in V5R4. The change allows the data to be broken on field boundaries instead of on internal byte boundaries. This will allow users to take advantage of only saving the information changed while still being able to read the journal entries that were written. To enable this new option, CRTJRN or CHGJRN should set the Minimize Entry Specific Data (MINENTDTA) parameter to *FLDBDY. If set on CHGJRN, a new journal receiver must be manually attached or created with *GEN on the JRNRCV parameter. Also,

once set to *FLDBDY, that journal can't be saved and restored to or remote journaled to a target system lower than V5R4.

When retrieving data that's been minimized on field boundaries, the data can be converted to be human readable or left in its compressed form. When using the Display Journal (DSPJRN) command, the data is always converted to human readable. On the Retrieve Journal Entries (RTVJRNE) command, Receive Journal Entries (RCVJRNE) command and the Retrieve Journal Entries (QjoRetrieveJournalEntries) API, there's a new Format Minimized Data (FMTMINDTA) parameter that determines if the data is converted or not. The default is to not convert the data because this is more efficient. However, if people or programs that can't read the compressed data will be processing the data, the parameter should be set to *YES to force the conversion.

Recovery Ratio

When a change is made to a physical file that's journaled, two items are actually updated--the physical file itself and the journal receiver currently attached. Because the physical file can be recovered from the journal, the system doesn't waste time by ensuring both changed items are pushed out to disk. Instead, the system ensures that the journal entry makes it to disk immediately and allows the changed physical file to remain in main memory. This leaves a gap between the most current journal entry and the state of the physical files on disk. Remember, the changed physical file only exists in main memory. The wider this gap is, the longer the system will need during IPL and iASP vary-on to recover the physical files that weren't pushed to disk in case of a system crash where main memory wasn't preserved. The recovery ratio sets the maximum size for this gap between journal entries and the physical file on disk per journal. Prior to V5R4, this could be set only at a system level and defaulted to 50,000. This meant that all journals on the system would have the same value. However, since this value is per journal and not per object journaled, not all journals worked well with the same setting.

At V5R4, the recovery ratio can now be set on a per-journal level. In addition, the default has been raised from a 50,000-entry gap to a 250,000-entry gap. This means that each journal on the system will need to review, at most, the last 250,000 entries to recover any files that may have been open just prior to a crash that lost main memory. The higher the recover ratio is set, the less frequently the physical files will get pushed to disk. This will result in a faster runtime but opens up the potential for slightly longer IPLs and iASP vary-on recovery processing time should the system crash and the open files need to be recovered. As a general rule of thumb, raising the recovery ratio by 10,000 shouldn't increase an IPL that's recovering files by more than 30 seconds, and usually much less. The set of system tasks responsible for maintaining this gap are named JOREC-***. If performance reports show these tasks taking a large percentage of CPU, it's possible that the recovery ratio is set too low, causing frequent disk writes.

Soft Commitment Control

In the past, both interactive and batch applications have been able to make use of journal caching to dramatically increase runtime performance when there's a high level of journal activity. When a

journaled database file is changed, the journal entry and the file itself must be written to disk. As stated earlier, the system ensures the journal entries are written to disk before the operation is considered complete. However, when journal caching is activated, the system builds up several of these journal entries in a 128 KB buffer before pushing them out to disk. Without caching, if the record size is 2 KB and there's a before and after image, a write was performed for only about 5 KB of data. When caching is on, the system attempts to fill a 128 KB buffer before performing the disk operation. In this example, the number of disk writes is reduced about 25 times, and application throughput is increased.

When commitment control is added on top of the journaling, the system must ensure that the journal entries representing completed commit transactions are written to disk. This means that prior to V5R4 at every commit, regardless of how much of the buffer is filled, a disk write operation must be performed to assure the matching journal entries reach disk. If an application has many small commit blocks, the performance benefit normally associated with journal caching is defeated. In order to address this problem, at V5R4 a new concept of soft commitment control was introduced. It's especially appropriate for batch jobs housing many SQL statements, each of which might establish a separate narrow commit transaction.

As with normal journal caching, when soft commitment control is activated, the system caches up multiple tentative journaled database changes before issuing the disk write operation on behalf of the journal receiver. However, while this slightly opens up the possibility that a few recently completed transactions would be lost if the system were to crash such that main memory wasn't preserved, the risk is quite low. The reason that the risk is low is due to various items. First, most crashes preserve main memory contents in which case no transactions will be lost. Second, a background SLIC-provided sweeper task regularly flushes the journal cache during dormant periods further reducing the likelihood of losing transactions. Notice that all files involved in a soft commit transaction will still maintain the transaction's integrity since a committed transaction either completely makes it or is completely lost.

Soft commitment control isn't the default. In order to activate this new functionality, the QIBM_TN_COMMIT_DURABLE environmental variable must be set to *NO. This can be set at the job or system level via the Add Environment Variable (ADDENVVAR) command.

See For Yourself

Overall, there are many enhancements to the journal facility in V5R4. These enhancements allow for better functionality and improved performance. For more details on these enhancements, visit the iSeries Information Center (http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp).

Robert Andrews is a staff software engineer with IBM Global Services. He's been with IBM for five years and is currently working in the iSeries Software Support Center for DB2 UDB on iSeries. Robert can be reached at robert.andrews@us.ibm.com.