V6R1 Journal Recovery Enhancements, Part 1

These improvements dispatch some key journaling conundrums

By Larry Youngren and Robert Andrews

V6R1 comes with a number of attractive journal enhancements that might help address concerns and frustrations that you currently have or have had in the past. Let's look at a few of the enhancements, exploring the background of the problems they address and discussing how V6R1 handles them.

A One-Way Street No Longer

Imagine that you're driving, and you come to a road sign advertising a scenic lookout. You follow the winding, one-lane road and come to the edge of a cliff overlooking a valley. Yes, the view is magnificent, but this isn't your final destination. You want to try other roads and view different scenery. Unfortunately, there's no room to turn around, and worse yet, there's a sign posted threatening severe tire damage if you try to back up. You feel trapped, don't you, and a bit misled?

That's the way some novice journal users felt in the past when they discovered that they had enabled journaling for far more files than they had intended and then wanted to change their minds. They felt that they had followed the signs for the correct route only to realize that they had traveled down a one-way street.

The problem stems from the fact that, up until V6R1, you could initiate journal protection while a file was actively being used, but until the file was closed, you could not switch directions and undo what you had accomplished. That is, you could not end journal protection for a file while it was actively being used. The STRJRNPF and ENDJRNPF commands simply did not behave in a symmetric fashion. What one command let you do while a file was open, the other did not let you undo until the file was closed.

It turns out that this lack of symmetry ensnared folks who were trying to set up journaling for a high availability (HA) product. They would rigorously assemble their list of critical database files to be replicated, enable journal protection for each, and then turn their applications loose, only to discover that they had put too many eggs in one basket, or worse yet, that they had errantly allowed work files to be journaled.

As soon as these people realized that the application was struggling because of their overzealous journaling or that the replication/replay process was falling behind, they tried to solve the problem by placing a fork in the road. That is, they wanted to cease routing all their files to the one master journal and instead split the traffic. In effect, they wanted to perform a U-turn and end journaling for half of their files so that they could restart journaling for these files to a second journal, thereby getting some performance relief via parallelism. Yet, with the application cranking away and the files already open,

the underlying ENDJRNPF command posted the "severe tire damage" sign. In effect, it said, "Sorry, not now, try again later."

This behavior was tremendously frustrating. You could enable new journal protection for a file while it was open but could not end journal protection until the file was closed. Hence customers who went down this one-way street had to either end their applications abruptly at an inopportune time (so as to reach a state that would allow them to cease journal protection) or wait until 2:00 a.m. to fix the problem.

Starting in V6R1, even if a file is open and actively being modified, you can end journal protection for that file. The consequence is that HA folks attempting to perform load-balancing adjustments (i.e., moving physical files from one journal to another) need not burn the midnight oil to make such adjustments. We call this our "end while open" support.

Do you need some special keyword or syntax to convince i5/OS to behave in this new fashion? No! It simply becomes the new default for the ENDJRNPF command. It's one of many enticing reasons to step up to V6R1.

"Are We There Yet?"

Many parents have felt like tearing their hair out after the umpteenth time of hearing that persistent question from the back seat during a cross-country trip. Yet it's natural for the young passengers to yearn to know the status and envision the completion of the trip. Journal users are much the same, especially remote journal users. They want to know the status of their long-distance transport of critical cargo.

To understand their motivation, a little background is helpful. In a logical replication-driven HA environment, a second copy of critical database files is often installed on a distant machine — your target recovery machine. In the event of a real disaster, you'd abandon the primary/source machine and switch users to the distant machine (out of harm's way). To trust the contents of this machine, however, its replicated files need to be refreshed regularly. That means that the delta occurring on the production machine needs to be captured and transported in realtime to the distant machine for replay. The capture step is easy; you merely need to enable journal protection for your critical files on the source machine. The transport step is more challenging. Adequate bandwidth and time are required to move delta information across these long distances. The underlying technology to accomplish this is called "remote journal support." Remote journal support appends groups of journal entries as bundles onto the designated communication line and then ships them off.

Remote journaling on i5/OS comes in two varieties: synch driven and asynch driven. As the name suggests, the synch variety is synchronous. That is, the source side waits for confirmation that the packet placed on the wire has successfully reached the target side. The upside of this approach is that you always know where you stand: If you're still waiting, the data hasn't yet arrived.

The downside, of course, is that waiting slows down your applications, and few businesses can afford much idle time. Consequently, the vast majority of remote journal users have elected to configure the asynch variety of remote journal, the benefit being good performance. Applications on the production system barely notice that a remote journal connection is in place.

The choice of asynch transport behavior, however, prompts a nagging question: "I wonder how long it's really taking for my changes to reach the target system?" That concern has kept some folks up at night; they keep wanting to blurt out, "Is it there yet?"

Before V6R1, i5/OS offered no easy mechanism for garnering a reliable answer to this question, but with V6R1, i5/OS incorporates new remote journal tracking/reporting support. The new support not only lets you know how many bytes linger on the source side unsent (if any) but also how long they linger. Both of these new statistics provide insight regarding what's happening in the background.

A few bytes, every now and then, lingering unsent for a tiny fraction of a second for remote journal users isn't unheard of — particularly if the mode of transport selected was asynch. After all, that's what asynch means. It's when large quantities of bytes regularly remain unsent and linger in this state for extended periods that more intense investigation is called for. The real question is, "Do I truly have a traffic jam, and if so, what's causing it?"

Every application environment is likely to experience natural peaks and valleys in database activity and the corresponding journal traffic. The real concern is how many bytes of journal information pile up during a burst of activity and how long they remain unsent.

If the total quantity of unsent bytes continues to increase over time and the backlog continues to grow, you may have a communications problem worth investigating. Unsent bytes can suggest that you've sized your communication pipe inadequately for the peak volumes that your applications tend to generate that you have a "dirty" communications line — one experiencing an excessive quantity of retransmits (a practice that slows down the whole remote journal transport process)

Both possibilities deserve investigation. Before V6R1, it was difficult to sense that you had a communications problem. With the arrival of the new statistics, monitoring for that condition just got a whole lot easier.

To see the new V6R1 remote journal transport statistics, use the WRKJRNA command and select F14 and then option 5 on the remote journal. The Display Remote Journal Details screen (Figure 1) appears.

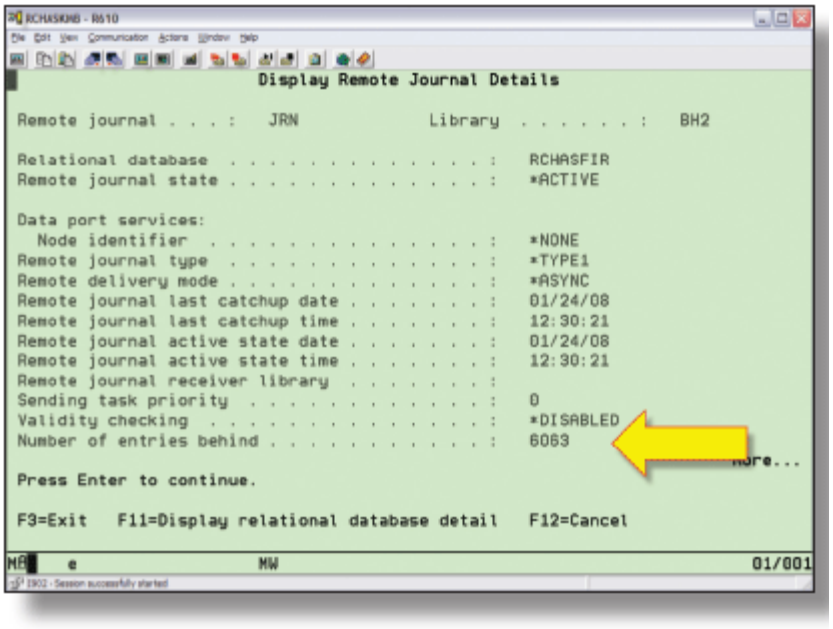

FIGURE 1
Display Remote Journal Details screen

Page forward to see the statistics values, as Figure 2 shows.

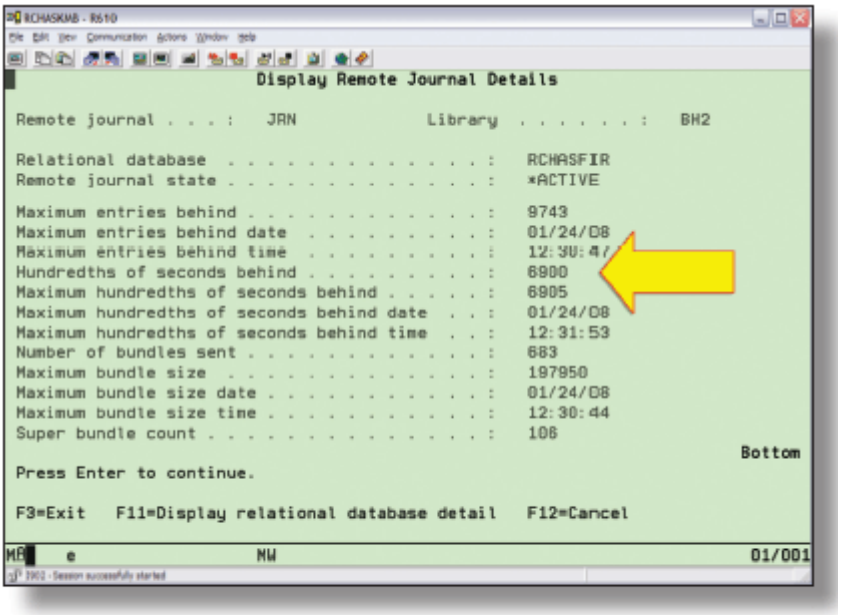

FIGURE 2
Display Remote Journal Details screen — statistics values

Pay particular attention to the Maximum entries behind field as well as the Hundredths of seconds behind field. The larger the values you observe in these fields, the more your source side is struggling to keep up. Also keep your eye on the Super bundle count field, which tracks the number of times the

underlying i5/OS software on the source side sensed that new bundles of journal entries were arriving as a burst and resulted in the operating system kicking into "super bundling mode" — a style of behavior under which the OS strings together consecutive groups of journal entries and sends them across the wire as a singular "super" bundle.

Having a low super bundle count suggests that your communication pipe is adequately sized and has plenty of room to spare. Seeing a higher value here doesn't mean trouble — yet. But it does suggest that you should keep an eye on the remote journal traffic to see whether it rises in the future to a point where bottlenecks begin to occur.

No Garbling Allowed

You may have noticed the next-to-last field on the screen in Figure 1: Validity checking. That is a new V6R1 feature added to the remote journal support.

If you ever played the game of "telephone" as a kid, you probably remember how you'd line up a set of classmates, whisper a phrase into the ear of the first kid in line, and ask him or her to pass it along. The last classmate was asked to state out loud the message. You probably howled with laughter as you discovered how your message had been garbled by the time it reached the end of the line.

Communication lines used for remote journal transmission, especially those that involve a configuration with lots of intermediate switches, can experience the same phenomenon: What went in clean can occasionally come out slightly garbled. Detecting that some transformation of the data has occurred during transmission was often a hit or miss proposition in the past. For short distances, ultraclean lines, and few physical switches, the problem rarely surfaced in remote journal environments. But for a few businesses, especially those that had recently modified their communication gear, we heard occasional stories of folks who were convinced that some garbling must have occurred.

The good news before V6R1 was that the remote journal technology itself surrounded each set of journal entries and database row images sent down the communication wire with a wrapper. This wrapper included crucial information that the target end of the remote journal technology itself needed in order to determine what it had received and where to put the images that had arrived. If the garbling affected the wrapper (the so-called metadata), the resulting lengths and locations were often downright ludicrous, so the remote journal connection was automatically terminated until the problem was fixed. Hence, the presence of such garbling might get detected — if the garbled items were ones that i5/OS itself cared about. However, if the potential garbling affected a row image from a database file, it was far less likely in the past that the remote journal transport layer could detect the side effect, and the garbled consequence probably surfaced days later when your application attempted to display the database row on the target side. That was an unfortunate but infrequent occurrence.

What does all this have to do with V6R1? V6R1 provides a new optional remote journal option — an extra validation step. If you make this choice, every set of journal entries sent down the wire carries an extra cyclical redundancy check (CRC) value, substantially increasing the likelihood that garbling will be detected promptly.

Should every remote journal configuration enable the new validation behavior? Probably not. This new behavior is optional on purpose. It mandates that extra steps be performed on the source machine in order to generate the CRC value each time a new bundle of journal entries is sent, and it requires that corresponding steps be performed on the target side to generate a CRC value anew to compare with the one sent from the source. If the two CRC values agree, no garbling has occurred. Producing these extra CRC values time after time consumes CPU cycles. As a rough rule of thumb, you can assume that using the validation option probably adds an extra 3 to 5 percent of overhead on the source side and an equivalent amount on the target side.

The overhead isn't inconsequential. It's an overhead you probably want to incur during a shakedown phase if you suspect a dirty line or have recently modified your communication configuration and, hence, want some assurances that no hiccups are occurring. In fact, many shops will probably elect to toggle the CRC on and off as needed.

The syntax for managing this new setting is on the command CHGRMTJRN. You enable the validity checking CRC behavior with the VLDCHK keyword, as shown in this example:

```
CHGRMTJRN RDB(MYRDBNAME) SRCJRN(MYJRN) VLDCHK(*ENABLED)
```

Alternate Routes

In the summer of 1959, Larry's family decided to take a road trip from Illinois to Alaska and back. Larry's dad thought that it would be fun to travel 10,000 miles in three weeks, heat stew on the manifold of a 1952 Nash, pitch a tent in the rain, and travel the Alcan Highway (which at the time was not much more than a gravel road). Two punctured gas tanks and 11 flat tires later, they completed the trip and had helped Alaska celebrate its statehood.

They also had one of the scariest experiences of Larry's life. Somewhere in Canada, they came to a place where weeks earlier a high bridge had spanned the Peace River. Towering nearly 14 stories high, the bridge let cars pass from one side of the river to the other. However, days before they reached that gorge, a heavy rain had swollen the river and washed out the automobile bridge. For some strange reason, downstream a bit, it had not washed out the railroad bridge. White knuckled, following the directions of flagmen posted at both ends of the bridge, Larry's dad drove across that railroad bridge — a bridge that, as Larry recalls, had no sides and must have been three or four football fields long!

Frightening as it was, at least there was an alternate route to cross the river. Without that railroad bridge, the trip would have ended abruptly. It's nice to have a second route when your first one fails.

What does this story have to do with V6R1? It turns out that the communication line between your source and target machine is the Achilles heel of your HA strategy. When it's down, your target machine isn't staying in lock step with what's happening on the production side. You're at risk. That's the reason that, beginning in V6R1, your remote journal connection is no longer restricted to using only one line at a time. In fact, if you have them, you can configure up to four separate communication lines all servicing the same remote journal connection. This setup adds robustness to your logical replication/remote

journal HA strategy. If one line goes down or turns sluggish, the remote journal support can continue using the remaining lines.

To advise the remote journal support that it can use more than a single line, you tie into the same multiline Data Port services software that the i5/OS clustering support employs. To do so, you need to tell i5/OS that it can think of the source and target machine at the two ends of the wire as a cluster of machines. Having established this so-called cluster, you merely identify the cluster as well as the specific IP links on the CHGRMTJRN command. Let's walk though an example to help illustrate the concept.

Let's say that you want to establish a remote journal connection between a pair of machines, and you also want to provide secondary IP paths. First, you create a local journal on the source side and then add a matching remote journal, identifying the target machine housing it by supplying a relational database (RDB) name.

After the remote journal connection is established, you use the CHGRMTJRN command to activate it. In V6R1, this command is enhanced so that you can specify that the remote journal traffic should use the underlying Data Port transmission software. To help the remote journal express its desire to tie into the Data Port transmission layer of microcode, you need to define a cluster and all nodes involved.

A word of caution: If there are only two machines, life is simple. If there are more than two machines, keep in mind that a system can be a member of only one cluster. This limitation means, for example, that if you have three systems set up such that both systems A and B do remote journaling to a common backup system C, and you want to use Data Port services, all three systems need to be defined as members of a single shared cluster. To create such a cluster and define the nodes therein, you can use Navigator for i5/OS or the CRTCLU command, naming the cluster, the nodes, and the IP addresses as Figure 3 shows.

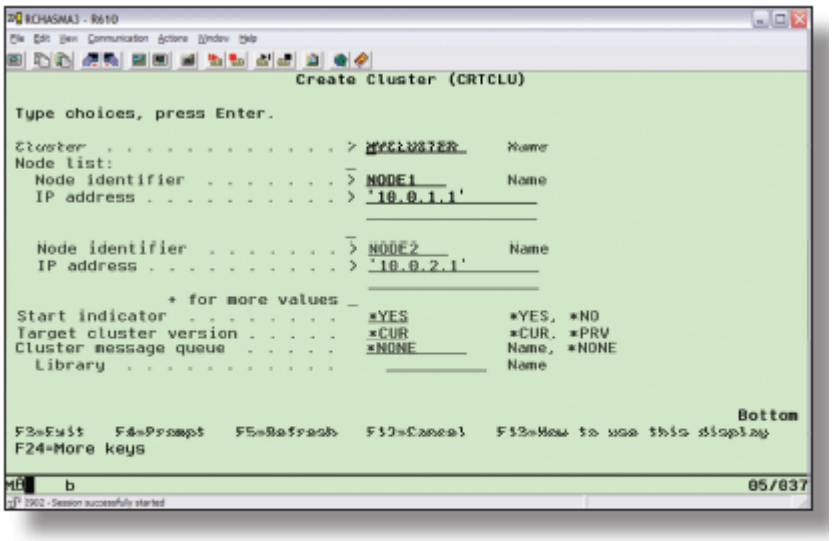

FIGURE 3
Create Cluster screen

Although you eventually might want to establish multiple paths between system A and system B, you must take one step at a time. The CRTCLU command asks you to provide only the first IP address. This first IP address establishes the Data Port link, which you can then use to capitalize on the other lines (up to four). After the cluster and a single end point on each node have been defined, it's time to activate the cluster nodes. Each node needs to be activated with Navigator or the STRCLUNOD command. These active nodes then can respond to the remote journal request, which we show next, and which wants to add more paths.

For our example, we have two systems, each with four network interface cards (NICs). The first system is on the 10.0.1.X network, and the second is on the 10.0.2.X network. Let's assume that the adapter's IP addresses end in 1, 2, 3 and 4 for our four lines, respectively. We create a cluster called MYCLUSTER with two nodes, NODE1 and NODE2. The commands look like this:

CRTCLU CLUSTER(MYCLUSTER) NODE((NODE1 ('10.0.1.1')) (NODE2 ('10.0.2.1')))

Remember, at this point we specify only one end point on each node. We then have to start each node in the cluster:

STRCLUNOD CLUSTER(MYCLUSTER) NODE(NODE1)

STRCLUNOD CLUSTER(MYCLUSTER) NODE(NODE2)

OK, the cluster activation is done. Now we turn our attention back to the remote journal itself. After clustering is started and the matching nodes are active, we can start the remote journal connection and define all the end points for Data Port services. We do this with the CHGRMTJRN command, as Figure 4 shows.

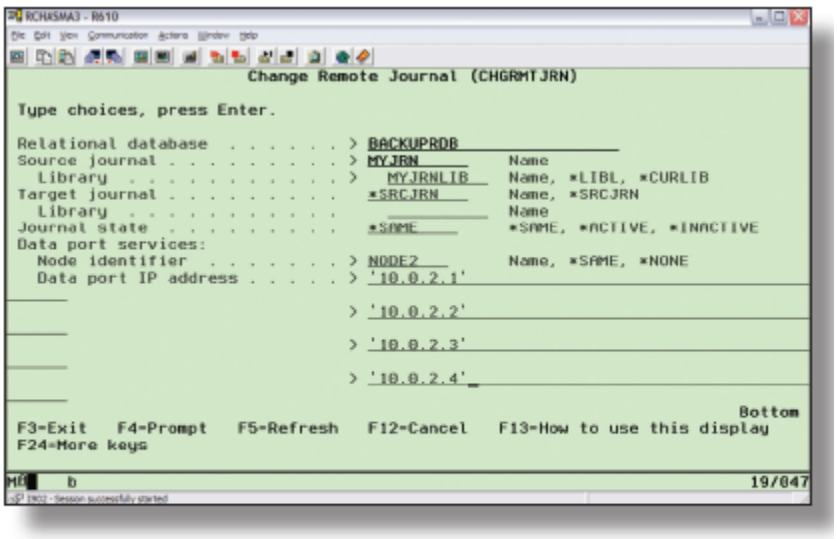

FIGURE 4
Change Remote Journal screen

To activate remote journal using this new support, the command looks like this:

CHGRMTJRN RDB(BACKUPRDB) SRCJRN(MYJRNLIB/MYJRN) DTAPORTSRV(NODE2 ('10.0.2.1' '10.0.2.2' '10.0.2.3' '10.0.2.4'))

Notice the new DTAPORTSRV keyword for V6R1. It's here that we specify the target node serving as a destination for our remote journal traffic as well as all four IP addresses of the second node.

Here are a couple of caveats:

- If we IPL the machine, the connections are dropped, and we must issue the CHGRMTJRN anew.
- If we IPL either system, the cluster nodes go inactive, and we must restart the nodes.

Also note that some of the Data Port options require additional software: 5761SS1 option 41 HA Switchable Resources and/or the 5761HAS System i High Availability Solutions Manager LPP. Both of these are additional, chargeable features. Alternatively, you can use the Example Tools Library (product option 7 for i5/OS — 5761SS1) to accomplish the same thing.

A Feast for Journal Users

Whether it's more even-handed journal treatment, better monitoring for potential traffic jams, timely detection and enhanced prevention for garbling, or merely increased bandwidth for remote journal needs, V6R1 has handy new features that will make some journal users salivate. Look for Part 2 of this article in an upcoming issue, when we'll continue our examination of V6R1 journal enhancements and show you how to exercise greater control over background tasks performing housekeeping chores on behalf of your journal environment, simplify data recovery, ease the burden of assuring that no file gets overlooked, and peek into the hidden journaling happening behind the scenes.

After more than 30 years of experience leading the design efforts for System i journal support at IBM, Larry Youngren recently retired from IBM and now lectures and consults on high availability issues. You can e-mail Larry at journal_guru@yahoo.com.

Robert Andrews is an advisory software engineer at IBM and focuses on database and journaling technologies on the System i. You can e-mail Robert at robert.andrews@us.ibm.com.