V6R1 Journal Recovery Enhancements, Part 2

By Larry Youngren and Robert Andrews

In part 1, you learned that V6R1 comes with a number of journal enhancements that might help address concerns and frustrations that you currently have or have had in the past. In part 2, let's continue our discussion of a few more of the enhancements, delving into the history of the problems they address and discussing how V6R1 handles them.

Mind Your Meter Maid

The date was 9/12/01. Larry was scheduled to fly from Minnesota to Florida to attend an IBM technical conference. It was the day after 9/11. There was uncertainty regarding how soon air traffic would be restored so he dutifully set his alarm for 4:00 a.m. and by 5:00 was on the road, headed to the Minneapolis airport. You can probably guess the rest of the story. He never got off the ground that day, nor did he ever get to the conference.

Rules regarding airports, air travel, and unloading zones in front of airports have never been the same since — a fact that was driven home months later when Larry tried pulling up to the loading/unloading zone at another airport. Along came an officer who in no uncertain terms advised him to "move along and be quick about it."

What does this story have to do with V6R1? It turns out that for a number of releases, i5/OS has harbored a similar meter-maid — a low-level microcode task that monitors for the presence of unwritten journal entries accumulating in main memory and schedules their departure to disk on a timely basis. We call it the journal cache sweeper task. Its mission is to assure that folks who employ the journal caching feature (i.e., the i5/OS optional feature that helps improve performance in a journal intensive environment) don't have their cached journal entries linger in main memory too long. But how long is too long? That's the tough question!

Flush this main memory buffer too frequently and you thwart the performance benefits achieved by caching. Flush this main memory buffer in too lackadaisical a fashion and you risk (especially in a remote journal environment) sending recent database changes to the target machine too infrequently.

In the earliest releases in which this meter maid was introduced, the decision regarding how frequently to have her swing by and empty the journal cache was a hard-coded value established in the lab — and we deliberately erred on the conservative side, not allowing her to show up so frequently that she could be accused of being a CPU hog. We quickly got the message that not all customers appreciated our heavy-handed approach. They wanted to exercise some control over the frequency at which this meter

maid uttered the words “move along!” And so, for a few releases there were some rather obscure and poorly advertised APIs one could use to give the meter maid a kick in the pants.

Most folks thought these APIs were tough to work with and not very obvious. They yearned for a more natural way to control the frequency with which cached journal entries are flushed from main memory.

V6R1 addresses that concern by letting you customize how frequently the meter maid shows up. You do so via the CHGJRNA command, which allows you to select a value between 1 and 600 seconds, suggesting how frequently you want this sweeping behavior to occur. Simply insert your own customized value for Cache wait time.

The Two Shall Become One

Larry’s 82 year old mom faces a challenge when an electrical storm knocks out power to her residence. She calls Larry and asks, “What time is it?” Throughout her townhome are multiple gadgets that attempt to display the current time of day. There’s the clock on the nightstand, the clock built into the stove, the clock on the microwave, the clock on the VCR, and probably half a dozen more that Larry doesn’t know about. If they all tell the same time, life is good. When they don’t, Mom is troubled, and Larry’s phone rings. You can imagine the chaos at Mom’s place when the U.S. Congress changed the starting date for switching out of daylight savings time last fall — some devices seemed to have a mind of their own: Some fell back an hour two weeks early, some didn’t move at all. There was no uniformity and predicting the outcome was tough. It all led to confusion.

Confusion crops up for journal users when there are multiple commands for achieving the same objective, but the commands behave differently. Yet for the past couple of i5/OS releases, that’s precisely what our customers faced. There were two competing commands at their disposal for use when they wanted to apply a set of journal entries.

There was the traditional journal-apply command — APYJRNCHG — and there was the newer apply-extended command — APYJRNCHGX. If you used the first, you’d fail to replay certain types of object-wide journal entries. If you used the second, you’d restrict yourself to only database objects and ignore the other types of objects that deserved journal protection (e.g., data areas, data queues). Confusion reigned.

What we needed was a single command — the same way Mom’s life would be less stressful if she had a single clock! V6R1 achieves that objective.

Sure, we could have come up with yet another apply command and claimed that it was the be-all culmination of this evolution in journal-apply technology, but doing so probably would have added to the confusion. Instead, cooler heads prevailed, and we simply elected to enhance the granddaddy command (APYJRNCHG) to be as smart as the new young whippersnapper command (APYJRNCHGX). Hence, the two have effectively become one, and whereas we often suggested that sophisticated shops use APYJRNCHGX in releases prior to V6R1, the new advice is to all begin the process of using

APYJRNCHG. In time, I suspect we'll withdraw support for APYJRNCHGX. While we were at it, we also elected to beef up how the whole apply process works and to do so in such a manner that

- logical files are no longer second-class citizens
- creation of new data queues and data areas now produces journal entries that can be replayed.

As a consequence, you'll begin to see some new flavors of journal entries show up. Their purpose is to help assure that changes to both physical files and the descriptive properties of logical files are all treated in an even-handed fashion and that all get replayed properly.

Inheritance, It's a Great Thing

To end up with some of these new embellished journal behaviors and thereby enable both IPL recovery and APYJRNCHG to strut their stuff, you need to flag the surrounding library itself as a journaled object. The name of this new property is inheritance, and it is managed with the new STRJRNLIB command.

Beginning in V6R1, when a library associates itself with a journal, all journal-eligible objects (e.g., physical files, logical files, data areas, data queues) created thereafter and therein inherit the same journal characteristics as the surrounding library. For example, if you issued the new STRJRNLIB command, designating library Lib1 as the library that you want to behave in this new fashion and journal J1 as the matching journal of interest, any CRTPF issued thereafter that placed the resulting PF into Lib1 would correspondingly enable journal protection for the PF at birth. Hence, the new object inherits journal protection from the surrounding library and routes its changes to journal J1.

Why does this behavior matter? It's especially attractive in a high availability (HA) environment in which you tend to have applications on the production machine that create new physical files or data areas as they execute. Without inheritance, the HA software provided by your business partner has to monitor the audit journal to notice that a new object has been created. It's then a foot race to see whether the third-party HA software can snag a copy of the new PF, prime it to the target machine, and enable journaling before your application begins to populate the new PF. As you can probably guess, the HA software often loses the foot race.

This new V6R1 support, called library inheritance, eliminates the foot race. Instead, i5/OS lets you ensure that every new PF has journaling initiated at birth. This assurance, in turn, reduces the likelihood that you'll end up with missing or overlooked objects or out-of-sync objects when you need to switch to the target machine.

In addition, the inheritance rules can be highly customized to influence which types of objects behave in this new fashion. You can also customize the conditions under which journaling is automatically started for an object. For example, you might want this behavior to occur when a new object of the designated type is created but not when it is restored from tape, or you might want automatic journal protection kicked off when an existing object is moved from a test library to a production library. For complete details about all the parameters, see the System i InfoCenter (ibm.com/systems/i/infocenter).

You can see these choices on the new STRJRNLIB command. For example, to direct i5/OS to use journal J1 for all objects created/moved/restored hereafter into library LIB1 you could issue the following command:

STRJRNLIB LIB(LIB1) JRN(MYJRNLIB/J1) INHRULES((*ALL *ALLOPR *INCLUDE *OBJDFT *OBJDFT))

With One Wave of Your Hand

Designating preferred behavior (inheritance) that you want to happen hereafter is good, but what if you've already got a bazillion objects residing in an existing library and want to enable journaling for every last one of them? That's a pain, right?

Well . . . it used to be a pain. That's because, prior to V6R1, if you wanted to start journaling for every physical file residing in library Lib1, you had to know the name of each PF and had to issue a separate STRJRNPF command for every last object!

It reminds Larry of the chore that he faces each autumn when the leaves and acorns begin to fall. Larry's fishing shack in northern Minnesota is surrounded by giant oaks. They provide lots of shade on hot days in July and August, but when late October rolls around, he yearns for a means to clean up the yard with one single, gigantic wave of his hand.

In a sense, that's what V6R1 provides. It lets you issue a single STRJRNPF command but with a new option — one that suggests that you know what you're doing and truly do want ALL PFs within the designated library to have journal protection started. Such a command would look something like this:

STRJRNPF FILE(LIB1/*ALL) JRN(MYJRNLIB/J1)

Notice that you could even provide a generic name (perhaps aimed at including your production files because they start with PROD* but leaving out the work files).

Timely Responses

With all this journaling going on, talking about good housekeeping makes sense. Some of the best housekeeping happens when you simply refuse to let junk build up in the corners. Journal receivers that stick around long past their useful life can become such clutter. They're a bit like a carton of sour milk trapped in the back of the fridge long past its expiration date. They take up precious space but it's unlikely anyone is going to get any useful purpose from their presence. In fact, on numerous occasions we've encountered customers who have complained about the need to purchase more and more disk and yet felt that their business had not grown by nearly the rate of disk consumption. What did we often discover? Journal receivers that were months (in one case, years!) old. Timely response to do a little housekeeping and cleanup on such journal receivers would have been prudent.

Larry heard about one of the best examples of prompt response when he lived in a northern Illinois community and taught in a school district that included the adjacent penitentiary known as Stateville. Among his students were the warden's daughters. The penitentiary was proud of the culinary skills it

had imparted to some of the inmates and loved to show them off. Special guests were ushered into a formal dining room where the stewards were inmates. The superintendent of the school district was one of those guests and told Larry how he had positioned the linen napkin on his lap, it had slipped off, and before he had time to reach down for it an inmate was at his side with a fresh one. That's a timely response!

Journal users, beginning in V6R1, can orchestrate a similar timely response. That's because there is now an opportunity to register an exit routine to be invoked each time a journal receiver is detached. You can accomplish this task with a command similar to the following:

ADDEXITPGM EXITPNT(QIBM_QJO_CHG_JRNRCV) FORMAT(CRCV0100) PGMNBR(*LOW) PGM(MYLIB/MYPGM)

As you probably realize, journal receivers are like buckets, and journals are like funnels placed over those buckets. Hence any database row images passed along to a journal for safekeeping travel through the funnel and end up in the bucket. The longer the bucket stays in place, the fuller it becomes. Eventually it nears its capacity and it's time to swap buckets. This is a two-step process known as detaching the old journal receiver and attaching a new one.

After a journal receiver is detached, it's time for some housekeeping to kick in. Although you've been able, for years, with commands such as CHGJRN JRN(MYLIB/MYJRN) DLTRCV(*YES), to instruct the operating system to automatically delete the journal receiver as soon as it's been detached, such automated behavior has felt too abrupt for some users. It meant that the journal receiver vanished before they had the chance to react. As a consequence, not nearly as many users capitalized on this automated delete receiver behavior as we had hoped. Instead, they had certain housekeeping chores they wanted to perform before the journal receiver disappeared. Managing that concern has just become easier.

Such housekeeping chores might include saving a copy of the receiver to a save file. With the preceding new exit routine interface, you could register a simple CL program that responds to the detach operation by saving the receiver before you delete it. That way you not only assure that you have a second copy available for the nightly save, but also that old journal receivers don't linger in perpetuity.

Want to take this further? Perhaps you'd like to save the detached receiver to a save file, ship a copy of the save file to a distant machine, and then clean up the space occupied by both the save file and the journal receiver. This procedure becomes particularly important as part of a timely space-management strategy if your applications tend to slurp up multiple gigabytes of journal receiver space per hour.

There are simply times and circumstances when housekeeping can't afford to be delayed until the end of the day. If that's the need that you have, the new V6R1 support should make your chore easier.

Having a Really Bad Day

We've all had them: days which started out looking pretty darn good only to take an unexpected turn for the worse. One of Larry's came in December 1976, when he had just completed graduate school and was looking for a job. Larry had taken summer assignments over the years at a variety of places, including the telephone company in Chicago (a branch of AT&T at the time known as Illinois Bell). His work there had been interesting, but he wanted to move into a development lab, so his buddies at Illinois Bell pulled some strings and got him an interview in a Chicago suburb at the Bell Labs site.

This was in the days long before laptops, e-mail, and electronic résumés. Larry's interview had been hastily arranged and hence he'd had no opportunity to send his paper résumé ahead via snail mail. Instead, the Bell Labs HR office had told him over the phone to bring along a typed copy of his résumé and hand it over at the first interview of the day. That sounded simple enough, so Larry put a fresh ribbon in the typewriter, used the finest cotton fiber paper he could afford, and carefully typed his one and only copy of a résumé.

Larry donned a white shirt, conservative tie, sport coat, and newly shined wingtips and drove to the Bell Labs location. He was ever so careful to keep the one-page résumé crisp and unfolded. Larry arrived a few minutes early, announced his presence to the receptionist, and spotted a nearby washroom. Knowing how each interviewer had a tendency to offer the interviewee a cup of java, he elected to be proactive and assure he entered the fray with nothing on his mind but the interview. Seeing no convenient shelf to park his résumé, Larry improvised and tucked his neatly typed résumé under his chin as he went about his business. You already can sense what's about to ensue, can't you?

He was doing fine until he moved to the wash basin and an old college chum who now worked at Bell Labs sauntered in, recognized Larry, and spoke his name. Instinctively Larry's head turned and, well, you can guess the rest of the story. Larry ended up with one soggy résumé and sensed that his pretty-good day had suddenly turned sour.

What does all this have to do with V6R1? In the System i world, there can be days that seem to be going well only to have a hardware or software glitch suddenly bring the machine down. Depending on which database files you had open at the time, and especially on how many SQL indexes or keyed logical files are affected, you can end up with a rather long recovery/IPL. Such long-duration IPLs are not much more pleasant than a soggy résumé.

The good news is that built in to i5/OS is a critical piece of support — think of it as a safety net — aimed at making your IPL duration not nearly so painful. It's System Managed Access Protection (SMAPP), a type of subtle journaling specifically for access paths.

This SMAPP support has evolved over the years and now has screens for letting you take a peek inside the machine and find out what's going on. In particular, you can display a screen that estimates how long an abnormal IPL would spend rebuilding keyed access paths if the machine were to go down abruptly. There are also companion screens that help you understand which indexes are currently protected and which are not. It's this set of screens that has been beefed up for V6R1. Prior to V6R1

there were only two; now there are three. Getting to know these screens and how to find them can help you extract the data you need to make better SMAPP decisions.

To gain access to these screens and thereby sample the current status of your SMAPP protection and/or to revise such protection, you can use the Edit Recovery for Access Paths (EDTRCYAP) command. From the EDTRCYAP main screen, you can use function keys to navigate to the display screen of interest. The three SMAPP status screens of particular interest are:

- the one that reveals the list of access paths that are not eligible to be protected: F13
- the one that reveals the list of currently protected access paths: F14
- the one that reveals the list of currently "exposed" access paths that qualify for protection but are not currently being protected (often because they're too small): F15 (this one is new for V6R1)

What might insights deep into the behavior of the automated SMAPP support allow us to learn?

If you discover that lots of very small duration access paths are protected, it probably means that your SMAPP setting is too low and that you're wasting precious CPU cycles for very little payback. Discovering that some substantial sized access paths are flagged as ineligible for SMAPP protection probably suggests that you have far more risk of long-duration recovery outages than you realize and that you should try to coax these access paths back onto the straight-and-narrow. Finding that lots of eligible access paths are on the unprotected screen might make you rethink your SMAPP settings — perhaps you're still locked into a choice made a decade ago and need to modernize.

V6R1 Journal Protection Has You Covered

What have we learned in this two part series? Whether you want to keep an eye on your remote journal traffic, take a peek at what SMAPP is doing, change your mind while a file is open, automate more of your journal garbage collection, assure that no garbled transmissions are slipping under the radar, nudge journal bundles onto the communication wire more rapidly, speed up IPL duration, increase the likelihood that no new PFs start out life without journal protection, or just shout out, "Are we there yet?", the new goodies tucked into V6R1 journal protection have you covered. The new features are in keeping with a major theme of V6R1: It's a release that takes high availability seriously!

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.