Remote Journal 101

By Robert Andrews

Do you have a High Availability (HA) plan involving data replication? If so, you are most likely dependent on remote journaling. Remote journaling, a core component of the IBM i operating system, allows for the contents of a journal receiver to be replicated to a remote system. A journal records the changes and activity to a wide variety of objects on your system. By recording the changes to these objects, HA software packages are able read these logs and replicate any changes made on the primary source system to backup target systems. This allows your data to be up to date on multiple target systems when changing it on only the source system. In this article we will start to look at the various facets that make up remote journal and allow for data replication.

Cascade vs. Broadcast

It is possible with remote journal to send changes made on one system to multiple systems. There are two methods to accomplish this. The first is known as broadcast. In broadcast distribution mode, the source system talks directly to each remote target system individually. The second method is cascade. In cascade distribution mode, the source sends the information to the first target system. The first target system then passes this information on to the second target system, and so on down the line.

Let's take a look at the advantages and disadvantages of each method. In broadcast distribution, the source system must send multiple copies of the data. This takes up additional bandwidth from your source system. With cascade, we delegate that bandwidth use down the line to the first target box, relieving the load on the source. However, if that first target box goes down, the data cannot cascade down to the second target system. The choice becomes between added workload and bandwidth on the source system and the need for multiple target systems to always have the most current data.

Asynchronous vs. Synchronous

To move the entries from a source to a target system, they can be sent in one of two modes. To understand how these modes vary, we need to first understand what happens when you make a database change in a non-journaled environment. When a record is changed, that image is only updated in memory. Eventually, it will be paged out to the disk for storage. However, as soon as that memory image is updated, the change is complete and control is returned to the program. Now, let's add local journaling to that picture. When a record is changed on a journaled file, we first update the image in memory. We then have to write the journal entry to disk. Remember that writing to the disk is still the slowest operation in the System i. This disk write of the journal entry to the journal receiver is what slows down applications after journaling has been started. Only after both the memory image of the record is updated and the journal entry is written to disk is control returned to the program.

So how does remote journaling affect this? The effect is different based on the method used. The first and most common method is to send the entries asynchronously. In this method, we update the record image in memory, write the journal entry on the local side, queue up the journal entry to be sent to the remote system, and return control to the program. As you can see, we only add a small amount of work

to queue up the entry to be sent. The second method of sending entries is synchronously. In this mode, we have to update the record image in memory, write the journal entry to disk on the local side, transmit the journal entry to the target system, the target side writes the journal entry to its disk, and sends an acknowledgement back to the source system. Only after this acknowledgement is received by the source system is control returned to the program.

As you can see, synchronous remote journaling requires us to wait for the communications traffic over, another disk write, and communications traffic back. Because of this, the amount of time to complete each update or transaction is increased significantly. The advantage to this method is that the target system is always up to date with the source system. The flip side to this is asynchronous communications. Here, each transaction is only slightly increased by the queuing action. However, because of this queue, the target system can fall behind the source system. Depending on the amount and size of the transactions and the speed of the communications link between the source and target systems, this delay could be a few seconds, minutes, or even hours. If you cannot wait and need 100% assurance that the entry is safe on both systems, use synchronous communications. For most customers, the decrease in application performance is too much and they can live with this slight delay from the queuing process and opt for asynchronous communications.

Journal and Library redirection

When sending a journal and its receivers to a target system, the library that contains the journal and the receivers can be overridden. This can be useful if multiple source machines are using the same target system. Let's assume that you have a standard PRDDTA library on each of the productions machines in New York and Los Angeles. Let's further assume that you have one back up system in Chicago. You first activate the remote journal for the New York PRDDTA library. It gets created fine. However, if you also wanted to back up you Los Angeles PRDDTA journal, it would collide with the New York copy already on the back up Chicago system.

However, we can use the concept of journal and library redirection to change the name of the library that holds the remote journal as well as the remote journal receivers. In the above example, we can redirect the New York PRDDTA to NYPRDDTA and the Los Angeles PRDDTA to LAPRDDTA. This way we can use the same target system to receive a common journal name from multiple source systems. And the journal and the receivers do not have to be redirected to the same library. If our New York journal was in PRDDTA and the receivers were in RCVLIB, we could redirect the journal to NYPRDDTA and the receivers to NYRCVLIB. Or we could send both the journal and the receivers to a library called NYBACKUP. The choices are limitless!

Type1 vs. Type2

After deciding how the systems are going to be configured, how they are going to talk to each other, and where the objects are going to go, the last choice deals with the type of remote journal receivers. There are two types; TYPE1 and TYPE2. TYPE1 receivers have the unique quality of knowing the names of both the source and target journal and receiver library names. Because of this, you can save and restore from either the source or target to either the source or target. However, since there is only room to store one remote journal and receiver library name, once you setup redirection for the first remote journal link, all additional links must use the same redirection information. So what we gain in the flexibility of save and restore we loose in redirection variations.

And, if you can't guess, TYPE2 receivers have the opposite benefits. TYPE2 remote journal receivers only store the source journal and library information. This means that you can move them from the target back to source, but not the other way around. However, since we only store the source information, you can change the library redirection for each different remote journal link. Hence, TYPE2 allows flexibility for redirection while limiting our restore locations.

Relational Database Directory Entries

To create a remote journal link, we need to tell the source system how to get to the target system. There are actually four different communication methods that can be used for remote journal: TCP/IP, SNA, OptiConnect, and DataPort Services (new at release 6.1). In this article, we are only going to discuss the most popular, which accounts for over 95% of remote journal connections, TCP/IP.

To define the connection, we need to create a relational database (RDB) directory entry on the source system that points to the remote system. To add this entry, we need to know the IP address or DNS name of the remote system as well as its RDB name. Keep in mind that if you are using remote journal to a second partition on the same physical system, using the Virtual IP capabilities of IBM i to communicate directly over the system bus will significantly increase performance.

To find the RDB name on the target system, run the WRKRDBDIRE (Work with Relational Database Directory Entries) command and look for an entry with a remote location of *LOCAL. This is the RDB name for that system. To find the IP address, you can use the CFGTCP (Configure TCP/IP) command and option 1, Work with TCP/IP interfaces. With these two items in hand, we can return to the source system.

Back on the source we can use the WRKRDBDIRE command to see if an entry already exists to the remote system. If it does not, then use the ADDRDBDIRE (Add Relational Database Directory Entry) command to create the entry to reach the target system. The entry name on the source side must match the *LOCAL entry from the target box. On IBM i, these names are always upper case. For the remote location, put in the IP address or DNS name and change the type from *SNA to *IP.

Advanced Tip: In some cases, you may want the source and target systems to have the same *LOCAL RDB name. This is crucial if a third system will be interacting with whichever of the two systems is functioning as the production system at the current time in our network. This technique would be an article in and of itself, but you can find out more details by referring to IBM's System i Knowledge Base Document 416466888.

Adding a Remote Journal

So, we have finally made all the choices of how our remote journal environment is going to be setup. Now we can actually implement all of these choices. The first step is to add a remote journal. This is done via the ADDRMTJRN command. Here we provide the RDB entry for the remote system, the local journal, any journal or library redirection, and the remote journal type. When enter is pressed, the first system will make a Distributed Data Management (DDM) connection to the target system to create the remote journal. This is the only time when we can have a journal without a receiver attached to it.

Since we are making a DDM connection, we need to make sure that the DDM server is running on the target system. We can check it via the NETSTAT OPTION(*CNN) command and look for local port 446 to

be listening. If it is not, we can start it with the STRTCPSVR SERVER(*DDM). The user profile that is making the connection from the source to the target must have the proper authorities on the target system to create the journal in the library specified.

You can control what user id and password are sent to the other system via server authentication entries. By default, the source system will only send the user id of the profile running the command. Your target system may require that a password is also sent or you may need to use a different id on the target system. To check if your target system requires a password, prompt the CHGDDMTCPA (Change DDM TCP/IP Attributes) command with F4. Look for the password required parameter. If it comes up with *SAME, you are missing the *IOSYSCFG special authority that is required to check that value.

We can accomplish either the sending of a password or a different user id via server authentication entries on the source system. These are a part of the user profile and cannot be found in any file on the system. They tie your source user profile and remote relational database name to a target user id and optional target user password. First, check if one exists for your profile and the remote relational database name that we used above with the DSPSVRAUTE (Display Server Authentication Entries) command. If it does, it will show you the user id that it will send as well as whether or not a password is stored. It will not actually show you the value of the password, just if one is stored or not.

If you need to add a server authentication entry, you can do so with the ADDSVRAUTE (Add Server Authentication Entry) command. The user profile parameter is the id on the source system. Keep in mind that this same security mechanism is used for activating the remote journal link. Therefore, if an automated profile will be starting your links, you may need to add an entry for that profile as well. The server parameter will be the remote relational database name target system in all upper case. The user id and password parameters determine the values that are sent to the remote system. In order to store a password, the QRETSVRSEC (Retain Server Security Data) system value must be set to 1. Also, if you change the password on the target system, you need to come back to source and update the server authentication entry with the new password via the CHGSVRAUTE (Change Server Authentication Entry) command.

Activating the Remote Journal

Finally, after all of the setup, we are ready to start sending entries from the source system to the target. To start moving entries, we need to change the state of the remote journal to active. We accomplish this via the CHGRMTJRN (Change Remote Journal) command. Here we provide the source and remote journal name, the state of *ACTIVE, our delivery mode (asynchronous or synchronous), and the starting journal receiver to send. Taking the defaults will result in asynchronous communications and will start with the currently attached receiver on the source system. If you need more history from the source side to be sent, you can specify which specific receiver to start with.

When you activate a remote journal, we start in what is known as catch-up mode. If this is our initial activation, we need to send the entire source receiver until this point. Catch-up mode optimizes the communications for sending larger blocks of data. This allows for any backlog data to be moved over as quickly as possible. Once we are caught up to current, we transition out of catch-up mode and optimize communications for individual entries. This allows for the quickest transfer while transactions are processing. While we are in catch-up mode, the job that activated the remote journal link will be in system inhibited mode and will only return after we transition out. Therefore, if you have a large

amount of data to move at activation time, you may want to submit the CHGRMTJRN command to batch processing.

One new feature in release 6.1 of IBM i is the ability to have the remote journal communications link add in a validity checksum. This uses a special algorithm to verify that the packets received by the target system over the line are the same as when they left the source system. While the algorithm is tuned for optimal software and hardware performance, this will add additional CPU overhead to both the source and target systems. However, if you are uncertain about the health of your communications lines, you can activate this feature on the CHGRMTJRN command.

If you need to end the remote journal connection, you can also do that from the CHGRMTJRN command by setting the state to *INACTIVE. Keep in mind that when you shut down the source system, the remote journal link will be ended. This means that after a system IPLs, the remote journal links will be inactive. You will need to the run the CHGRMTJRN to set it back to *ACTIVE. While this is something you could automate in a process such as QSTRUP, you need to remember the security requirements listed above and the profile that will actually be executing the command.

For those of you that have firewalls between the source and target systems, there are two main ports that need to be open to the target system. As mentioned above, we use DDM to create and start remote journal links. This requires that port 446 is open. After the DDM handshake is done for security, we transition to a proprietary communications protocol running on port 3777. Both catch-up mode and regular run-time modes communicate over this port.

Performance

Finally, let's review a couple of performance considerations. Everything that is in the local journal will be sent across to the remote journal. If the remote journal link is falling behind, there are two major things we can do: decrease the amount of data sent or increase the size of the communications line. Keep in mind that there may be some normal fluctuations in the amount of transactions on your system. For example, you may have a batch process that kicks off at midnight and finishes at 2 A.M. However, it may take until 4 A.M. for that large rush of transactions to be sent across to the target. But if there are no other tractions being done until business hours start, is this really a problem?

If we suspect there is an issue, let's look at the first problem point: too much data to send. There are several ways to reduce the amount of data inside a journal receiver. First, we may need to look at how our journaling environment is setup. Maybe you have several libraries all journaled to the same journal but you only need some of them for your remote journal purpose. In that case, split the one journal up into two and only send the one to the remote system.

We can reduce the number of journal entries by only journaling the after images instead of both before and after images or by omitting the open and close entries for files. We can also reduce the size of each journal entry by reducing the information captured in the header of each entry. Often when we update a row, we only change one or two fields out of the entire record. By default, journaling records the entire record. This can consume a lot of extra space since most of the record has not changed. To help with this, we can use a concept known as minimized entry-specific data. It is important to note that depending on how we implement this feature, the entries may no longer be human readable, but the HA products have no problem working with these compressed entries.

There are also internal entries that can be stored in a special part of the journal. To enable the movement of the internal entries to a space that is not sent to a remote journal, we want to use the RMVINTENT (Remove Internal Entries) option on the journal. These journal diet plans are covered in detail in the Journal TechNotes section listed at the end of this article. Please see those articles for more information on the ramifications and implementation methods of these various items.

If we have the journal receivers as small as possible, the other place we can turn to is the communications line. When planning the size, we need to account for the size of the journal receivers to be sent, the added overhead of TCP and IP headers, as well as any other traffic on that same line. Keep in mind that data sent over TCP/IP is sent in packets, usually around 1400 bytes large. Each packet has a 20 byte IP header and a second 20 byte TCP header. These headers alone add almost an additional 3% traffic load.

Because of this and other line considerations, we usually estimate to get an actual data throughput of about 75% of what the line is rated for. For example, if you have a 10 megabit per second connection, in theory, that line can move 4.5 gigabytes of data per hour. However, applying our 75% rule, we expect an actual data throughput of closer to 3.375 gigabytes per hour. If you think that you may have a problem, a methodology for tracking your journal and line usage is explained in IBM's System i Knowledge Base Document 396656252.

Conclusion

I hope that I have given you the basic information you need to make an informed choice about how to setup your remote journal environment. Remote journal is the cornerstone technology to building a data-level replicated High Availability (HA) system. However, keep in mind that remote journal simply gets the journal receivers from the source system to the target system. It is now time to pass off these remote journals to your HA software to complete your environment.

References and Additional Reading

- IBM i Information Center – Journal Management
  - http://publib.boulder.ibm.com/infocenter/systems/scope/i5os/index.jsp?topic=/rzaki/rzakiremotekickoff.htm
- Redbook - Striving for Optimal Journal Performance on DB2 Universal Database for iSeries
  - http://www.redbooks.ibm.com/abstracts/sg246286.html
- Journal TechNotes
  - http://www.redbooks.ibm.com/redbooks.nsf/Tips?SearchView&Query=journal
- System i Knowledge Base
  - http://www-912.ibm.com/s_dir/slkbase.NSF/wdb2?OpenView&Expand=9.1#9.1

Robert Andrews is an advisory software engineer with IBM Global Technology Services. He has been with IBM for eight years and is currently working in the IBM i Rochester Support Center focusing on DB2 and journaling. Robert can be reached at robert.andrews@us.ibm.com.