

**4F44RCV 'Backup and Recovery for Library Units' (BRLU)
Version 4 Release 1 Modification 0 Performance Tips**

Table of Contents

Introduction.....	3
BRLU Performance tips.....	3
ISeries journal performance.....	3
Parallel CLRPFMs.....	4
Recovery performance tips.....	4
Multi-streaming.....	4
Automatic recovery.....	6
Object Holder Files.....	6

Introduction

BRLU tool is designed for high performance. In particular, parallel technologies are used by many of its features. Parallel technologies, however, is a double-edged sword: they can deliver very good performance but often require tuning and good understanding of their principles of operation.

This paper contains several tips on how to maximise BRLU performance and avoid bottlenecks.

BRLU Performance tips

ISeries journal performance

BRLU stipulates that all database tables and data areas used by the application must be journaled to iSeries journals. IBM “Striving for Optimal Journal Performance on DB2 Universal Database for iSeries (SG24-6286-00)” contains a lot of useful advice on how to make the best use of iSeries journaling. It is highly recommended to study this manual before proceeding with BRLU implementation.

The following sections are of particular importance:

- Disk write cache
- Batch Journal Caching PRPQ (5799-BJC)

Sufficient amount of disk write cache must be provided in order to make iSeries journaling perform. Very good performance can be achieved by using #2757 disk controllers with 100Mb+ of disk cache space per controller.

Many iSeries publications suggest that the best journal performance is achieved by creation of a dedicated journal ASP, thus dedicating at least one disk write cache device to journals. This suggestion, although good and sound, is nevertheless very expensive. On the one hand, there are rarely any disk arms to spare, and on the other, with multiple library units in a single iSeries partition, the requirement of dedicating a separate ASP with a write cache device to each of the library unit journals looks obviously excessive. A much more practical approach would be to only use dedicated ASPs when there are disk arms to spare in the system ASP. If this is not the case, however, good performance can be achieved by creation of journal receivers spanning the maximum possible number of disk arms in the system ASP.

The algorithm employed to decide how many disk arms to spread the journal receiver across is roughly: $\# \text{ Arms} = \text{Journal_Receiver Threshold_setting} / 64 \text{ Meg}$. Thus, journal setting of RCVSIZOPT(*MAXOPT1) combined with the receiver setting THRESHOLD(10000000) can help achieve reasonable performance even in the case of journal objects allocated in the system ASP.

Batch journal cache feature (5799-BJC or 5722-SS1 option 42) is strongly recommended when long-running batch processes are part of the application.

Parallel CLRPFMs

In the BRLU environment each CLRPFM command issued for files in protected libraries is intercepted and replaced with a native iSeries process removing all records from the target file. Some other commands (e.g. CPYF MBROPT(*REPLACE)) issued for files in protected libraries invoke modified CLRPFM process under the covers. If the target file contains a large number of records this can turn into a performance problem. NREC parameter of STRRCVMOD command can be used to address this issue. IF the value of NREC is a number each CLRPFM process is broken down into a number of streams with each stream deleting the given number of records from the range allocated to the stream.

Streams are submitted as separate jobs using either CRCVCLR job description, if it can be found in *LIBL, or QBATCH job description. CRCVCLR job descriptions for different BRLU units can be tailored differently, e.g. they can be used to direct CLRPFM streams to either one or multiple main storage pools, thus limiting CLRPFM influence on other jobs in the system. How to use job descriptions for directing submitted jobs to different main storage pools is explained in the IBM iSeries Work Management manual (SC41-5306-03) (<http://publib.boulder.ibm.com/series/v5r2/ic2924/books/c4153063.pdf>).

Recovery performance tips

Multi-streaming

RCVTOCKP command represents a single interface to the BRLU unit recovery process. This process, however, can be tailored according to the size of the unit and the number of modifications being recovered.

By default, RCTVOCKP process runs in the same job where the command is issued. In this case a single RMVJRNCHG system command is used under the covers for all of the objects being recovered. The major approach of this approach is its simplicity, but if a large number of DB modifications are to be removed the performance of this process can be rather poor. The reason is, RMVJRNCHG command is executed as a single stream and consequently, only one iSeries processor (out of potential many) can be used.

There are several ways to force RCVTOCKP process to be executed in parallel mode. IF STREAMS, FLIST, OFLIST, or ROLIST parameters are given anything but default values RCVTOCKP process submits concurrent jobs to perform the recovery. There is generally no way to control the unit of work allocated to each of the recovery streams: objects to be recovered are analysed and broken down into a number of groups, up to 190 objects in each, with recovery process for each group submitted as a separate job.

There is one exception to the above rule. FLIST parameter may be used to specify up to 10 database tables to be recovered in separate streams. This facility can be used when the number of modifications for each of those tables is known to be much higher than for any of the others.

All data areas are always recovered by one stream.

STREAMS parameter can be used to specify the lower limit of the number of recovery streams to be submitted. If the number of objects in the unit libraries is too high to be broken in the given number of streams with each stream processing no more than 190 objects the number of streams is adjusted accordingly. The same happens if FLIST parameter is used to specify tables for individual recovery.

Journal management component of the iSeries makes very aggressive use of main storage. Therefore, submitting multiple recovery streams to the same main storage pool may lead to excessive paging (thrashing) because of multiple recovery streams trying to process different entry ranges from the same journals. Ideally, each of the streams should be executed in its own main storage pool. This is especially important for the streams removing large numbers of entries.

RCVTOCKP submits the recovery streams in the following order: first, streams defined by FLIST parameter, then all other streams. Streams are assigned sequence numbers – in the order they are submitted for execution. RCVTOCKP attempts to submit each stream using the following algorithm to determine the name of the job description to be used:

- CRCVRJDnnn, where nnn is the sequence number of the process, if the job description with this name can be found in *LIBL
- CRCVRJD, if a job description with the above name cannot be found in *LIBL but CRCVRJD job description can
- QBATCH in any other case

Jobs are submitted to *JOBQ job queue, unless the default value of the JOBQ parameter of SBMJOB system command has been changed.

Using job descriptions CRCVRJDnnn one can direct multiple recovery jobs to different main storage pools. It may be done only for the first few recovery streams (particularly for those generated from FLIST parameter); the rest of the streams can be submitted to the same pool.

It is recommended to create a separate subsystem to be used in case of recovery. This subsystem must contain the required number of main storage pool definitions and also multiple number of job queue or routing entry definitions – for directing jobs submitted using CRCVRJDnnn job descriptions to their related main storage pools.

Parallel execution of RCVTOCKP process can increase its speed up to 10 times.

Automatic recovery

Another facility that can help improve recovery runtimes is Option 3 Automatic Recovery feature. Option 3 was primarily designed to implement BRLU standing data when the objects to start journaling for, locks to be verified, number of parallel recovery streams, and object distribution between these streams are defined only once for each unit. After that there is no need for the operator to take any of the above decisions; he can just refer recovery to the previously defined set of parameters.

Option 3, however, also includes the work management component responsible for generation of recovery job descriptions and manipulation of i5/OS work management objects, such as job queues and subsystems. Enable Automatic Recovery (ENARCYUNT) command optionally creates these work management objects to be used by the recovery process. The list of such work management objects includes CRCVSBSunt subsystem, CRCVSBSunt job queue, and CRCVRJDxx job descriptions; the subsystem description and the job queue are saved in CRCVQS library, and the job descriptions – in the control library for the unit. The subsystem has 10 main storage pools defined, *SHRPOOL1-*SHRPOOL10. The number of job descriptions created is $\max(n+1,10)$ where n is the value of the “additional file streams” (FLIST) parameter of RCYTOCKP command. At execution time the first $\max(n+1,10)$ recovery streams are, therefore, run in separate main storage pools. The tool, however, exercises no control over the actual main storage allocation. The recommended strategy is to use QPFRADJ=3 setting thus allowing the operating system to automatically allocate main storage to shared pools.

It is highly recommended to use Automatic Recovery Facility to facilitate main storage management for high performance.

Object Holder Files

If the value of LIBLIST parameter of RCYTOCKP command equals ‘*OBJECTS’ BRLU only recovers individual objects (ROLIST). Starting from V4R1M0 this list can contain so called object holder files (*OBJHLD type). Object holders are actually “DSPOBJ DETAIL(*BASIC) OUTPUT*OUTFILE)”-compatible physical files containing actual objects (files and data areas) to be recovered.

It sometimes happens that although a large number of objects from a certain library must be recovered, a large number of objects (as many or even more) from the same library don't require recovery. Of course, ending journaling for the latter would allow to drop them from the list of objects to be recovered, but it may not always be convenient, especially in High Availability configurations where journaling for objects is taken over by the HA software.

In such cases object holder files can be used to reduce the number of actual objects to be recovered and therefore to improve performance characteristics of the recovery process.