

Memory for MIPS: Leveraging Big Memory to Boost Db2 for z/OS CPU Efficiency

—

Robert Catterall
Senior Consulting Db2 for z/OS Specialist
IBM
rfcatter@us.ibm.com

Tridug

December 5, 2018



Agenda

The current landscape

Getting your buffer pool house in order

Being bold – but not reckless – in asking for more real storage for a Db2 subsystem

Exploiting `RELEASE(DEALLOCATE)`

Other ways to trade memory for MIPS

The current landscape

IBM Z server memory: getting BIG

Seeing more production LPARs with 100s of GB of memory

What's driving the trend towards larger LPAR memory sizes?

– Desire for balanced configurations: more memory to go with more MIPS

- Up to 141 engines on a z13, up to 170 engines on a z14 – each engine provides 1000+ MIPS of processing capacity

– Price of IBM Z memory now much lower than it used to be

My recommendation:

– z/OS LPAR: at least 20 GB of real storage per engine

- That's any kind of engine – if a z/OS LPAR has 5 general-purpose engines and 5 zIIP engines, it should have at least 200 GB of real storage, for a balanced configuration

New ways to leverage Big Memory on Z

Recent Db2 for z/OS developments provide more opportunities to use IBM Z memory advantageously

A few examples (covered in more detail later in presentation):

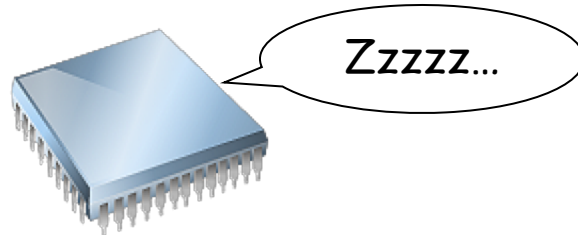
- Page-fixed buffer pools
- Use of larger real storage page frames for buffer pools
- Db2-aware “pinning” of objects in buffer pools (enhanced with Db2 12)
- Thread-related virtual storage almost entirely above 2 GB “bar” (assuming packages bound or rebound in Db2 10 or later environment)
 - More virtual storage “head room” for use of `RELEASE(DEALLOCATE)`
- High-performance DBATs
- Index fast traverse blocks (Db2 12)
- Greater use of in-memory sorting, sparse indexes (Db2 12)

There are a lot of “sleeping gigabytes” out there

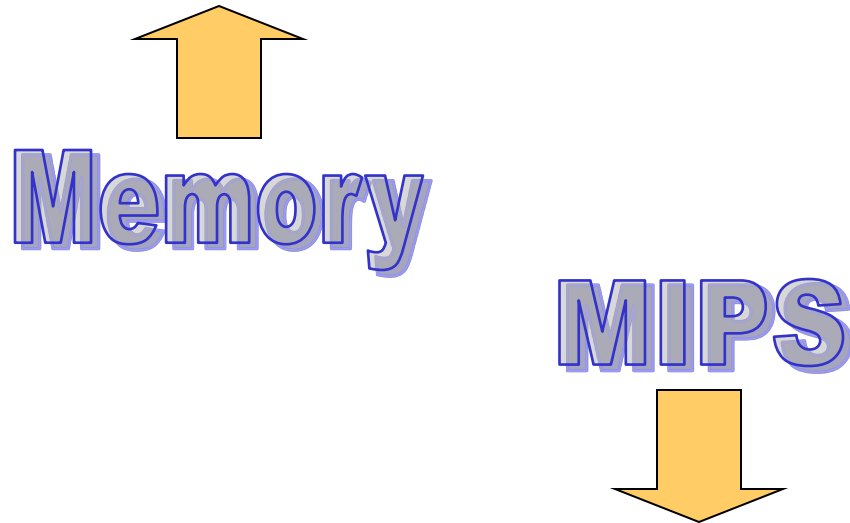
At many Db2 for z/OS sites, LOTS of spare memory capacity

Do you know your z/OS system’s demand paging rate?

- Available via a z/OS monitor, this is the rate at which pages that have been sent by z/OS to auxiliary storage are paged back into system memory on-demand
- In my experience, often 0, even during busy periods
 - If the demand paging rate is 0, z/OS LPAR memory is not stressed at all – should be OK to expand use of real storage pretty aggressively
 - Very small non-zero demand paging rate (i.e., <1/second) indicates small amount of pressure on real storage – some targeted, modest additional use of memory probably OK
 - Look for opportunities to use more memory as a means of boosting the CPU efficiency of your Db2 workload



I call this memory for MIPS – that's what this presentation is all about



Getting your buffer pool house in order

What I mean...

There are things you can do to get more performance bang for your buffer pool buck without enlarging the buffer pool configuration



First, customize settings for work file buffer pools

Referring here to the buffer pools dedicated to the 4K-page and 32K-page work file table spaces

– **Highly recommended:** dedicate buffer pools to work file table spaces

Work file table spaces are different from others in a couple of ways that have implications for recommended buffer pool parameter settings

– For one thing, large majority of reads tend to be of the prefetch variety

- Why that matters: the default value of the VPSEQT buffer pool parameter (the percentage of a pool's pages that can be occupied by pages read into memory via prefetch) is 80
- Stay with that, and you could be under-utilizing 20% of the buffers in a pool dedicated to work file table spaces
- Increasing VPSEQT to 90-95% for a work file-dedicated pool should result in decreased read I/O activity, and fewer I/Os means less CPU consumption

Work file-dedicated pools – the other difference

Different motivation for externalizing changed pages to disk

- For other buffer pools, getting changed pages externalized to disk in a timely manner is important for Db2 restart performance
 - Forward log recovery phase of restart: data on disk updated to reflect changes committed, but not yet externalized, at the time of Db2 subsystem failure
 - The more changed-but-not-yet-externalized pages there are in buffer pools (other than work file pools) at time of Db2 failure, the longer restart will take
 - That being the case, you want fairly low values for deferred write thresholds (DWQT, VDWQT) for these pools (defaults of 30 and 5 are usually good)
- Work files are like scratch pads for query processing – they are not recovered when Db2 is restarted (queries would be resubmitted)
 - Therefore, only need a level of changed-page externalization activity that is sufficient to prevent thrashing (i.e., avoid shortage of stealable buffers)
 - Fewer page writes = less CPU consumption
 - 70/40 can be good for DWQT/VDWQT for work file pools – even 80/50

But don't take this too far...

A warning about too-high DWQT/VDWQT values

One organization set DWQT and VDWQT to 90 for a work file-dedicated buffer pool

The data manager threshold (aka DMTH) is reached when 95% of a pool's buffers are non-stealable (either currently in use or changed-but-not-yet-externalized)

- When DMTH is hit for a pool, there will be a GETPAGE for each row retrieved from a page cached in that pool
 - So, if 20 rows are retrieved from a page, there will be 20 GETPAGEs vs. 1
 - Result: significant increase in Db2 CPU consumption
- With DWQT and VDWQT both at 90, at this site DMTH was hit many times per hour for the buffer pool in question, unbeknownst to Db2 staff
- Check Db2 monitor display or statistics report, or output of Db2 command `-DISPLAY BUFFERPOOL(BPn) DETAIL` to see if DMTH hit
 - You always want that value to be zero

Smarter “pinning” with PGSTEAL(NONE)

When PGSTEAL(NONE) is specified for a pool:

- When an object (table space or index) assigned to the pool is first accessed, requesting application process will get what it needs and Db2 will asynchronously read all the rest of the object’s pages into the pool
- If objects assigned to a PGSTEAL(NONE) pool have more pages than the pool has buffers, Db2 will use FIFO buffer steal algorithm to accommodate new page reads from disk when pool is full
- Db2 12 will arrange an object’s pages in memory as they are arranged on disk, for more-efficient page access
 - If some buffer stealing is required for a PGSTEAL(NONE) pool in a Db2 12 system, it will be limited to the pool’s overflow area (10% of the pool), so that pages in other 90% of the pool can continue to be arranged in memory as they are on disk

CPU savings via page-fixed buffer pools

PGFIX(YES) specification for a buffer pool saves MIPS, partly by making database read and write I/Os less costly

- PGFIX(YES) buffers stay fixed in memory, so no need for Db2 to ask z/OS to fix (and subsequently release) a real storage page frame holding a buffer when data is read into, or written from, that buffer
 - For 32-page prefetch read, that's 32 page-fix/page-release operations avoided

PGFIX(YES) is not just about cheaper I/Os

Part of memory resource of an IBM Z server can be managed in 1 MB page frames

– LFAREA parameter in IEASYSxx member of PARMLIB

1 MB page frames can be used to back a PGFIX(YES) buffer pool

– Larger page frame means more CPU-efficient translation of virtual storage addresses to real storage addresses

– This CPU benefit is on top of the previously mentioned MIPS savings resulting from less-costly I/Os into and out of page-fixed buffer pools

- Important: large frames deliver CPU savings even for pools with low read I/O rates

– Use -DISPLAY BUFFERPOOL(*BPn*) DETAIL command to see if Db2 is using large page frames for a PGFIX(YES) buffer pool

– For a pool that was not page-fixed in a Db2 10 environment, explicitly specify FRAMESIZE(1M) in a Db2 11 or later system

2 GB page frames for buffer pools

Requires Db2 11 or later, plus:

- z/OS 2.1 or later
- zEC12 or later server
- LFAREA specification (in IEASYSxx member of PARMLIB) with non-zero value for 2G option

When it will be used:

- Buffer pool must be defined with PGFIX(YES) and FRAMESIZE(2G)
- Size of buffer pool must be at least 2 GB (or very close to it, if < 2 GB)
 - Amount backed by 2 GB frames will be integer portion of (pool size) / 2 GB
 - Remainder (if any) will be backed by 1 MB and/or 4 KB page frames
- 2 GB frames may not improve performance much vs. 1 MB frames, unless pool is really big
 - One organization reported seeing significant CPU efficiency benefits from the use of 2 GB (versus 1 MB) page frames when pool size was 20 GB or larger
 - That said, doesn't hurt to use 2 GB (versus 1 MB) page frames for pool that is smaller than 20 GB

Reduce I/Os by redistributing buffers

Referring here to reducing size of low-I/O pools and increasing size of high-I/O pools by a like amount

- Goal: about same read I/O rate for reduced-size pools (don't take too many buffers from these pools), and less read I/O activity for enlarged pools – all with no net increase in overall size of buffer pool configuration
 - Again, fewer I/Os = reduced CPU consumption

Step 1: determine read I/O rate for each buffer pool, using information in...

- Db2 monitor (statistics report, or online display of buffer pool activity)
- or-
- Output of Db2 command `-DISPLAY BUFFERPOOL(ACTIVE) DETAIL`
 - Issue command once, then wait an hour and issue it again
 - Output of second issuance of command will capture 1 hour of activity – divide activity counters by 3600 to get per-second figures

Calculating a pool's read I/O rate

What you want is per-second data

Total read I/O rate is sum of:

all synchronous reads (random and sequential) per second

and

all prefetch reads (sequential prefetch + list prefetch + dynamic prefetch) per second

Example: BP1 and BP2 both have 40,000 buffers, and total read I/O rate is 20/second for BP1 and 2000/second for BP2

– In that case, I'd consider decreasing size of BP1 by 20,000 buffers and increasing size of BP2 by 20,000 buffers

Being bold – but not reckless – in asking
for more real storage for a Db2 subsystem

Would bigger be better?

If buffer pool house is in order, should you make house bigger (i.e., increase the size of the overall buffer pool configuration)?

Depends – what's the current total read I/O rate for each of your buffer pools (see slide 18)?

- My opinion: less than 1000 read I/Os per second for a pool is good, less than 100 per second is great
 - Some organizations drive for a read I/O rate below 10 per second for their buffer pools
 - Fewer I/Os means less CPU consumed in driving I/Os
 - For CPU savings, check average in-Db2 CPU time in Db2 monitor accounting report or online display
- Note: for PGSTEAL(NONE) buffer pool, target read I/O rate is zero (or very close to it)

If you're going to make a buffer pool bigger...

...add enough buffers to make a difference

- Adding another 1000 buffers to a pool that already has 80,000 buffers is not likely to move the needle very much
 - Increase size of 80,000-buffer pool by 20,000 or 40,000 buffers, and I'd say, "Now you're talking"
 - If a pool is quite small – say, 10,000 4K buffers – and has a high read I/O rate, I might want to increase its size by a factor of two (or more)

Some organizations definitely “get it,” in terms of taking advantage of Big Memory to reduce Db2 CPU consumption via larger buffer pool configurations

- The largest buffer pool configuration I've seen on one Db2 for z/OS subsystem is 879 GB (associated z/OS LPAR has 1104 GB of memory)
- Largest single pool I've seen: 253 GB

Don't over-use server memory

Again, the measure of “memory stress” that I like to use is the demand paging rate

– As you implement memory-for-MIPS changes, keep an eye on the z/OS LPAR's demand paging rate, and don't let this get out of hand

- Demand paging rate of 0 is great, between 0 and 1 per second is good, 2-3 per second is about as high as I'd want it to get

Keep monitoring read I/O rate for buffer pools

Triage approach: focus on pools with highest read I/O rates

Good if you can get read I/O rate below 1000 per second for each buffer pool

- If that objective is accomplished (or if you're seeing diminishing returns with respect to enlarging a high-I/O buffer pool), turn focus to pools with read I/O rates between 100 and 1000 per second
 - Nice to get these below 100 per second
- For “pinning” pool, desired read I/O rate is zero (or very close to it) – enlarge if necessary

Keep in mind: it's not just about making existing pools larger

- At some point, may want to create new BPy, and reassign objects to that pool from BPx
 - Can be particularly effective for separating “history” vs. “current” tables, access patterns for which tend to be different

Db2 12 index fast traverse blocks (FTBs)

FTBs kept in FTB storage pool (by default, 20% of total size of buffer pool configuration)

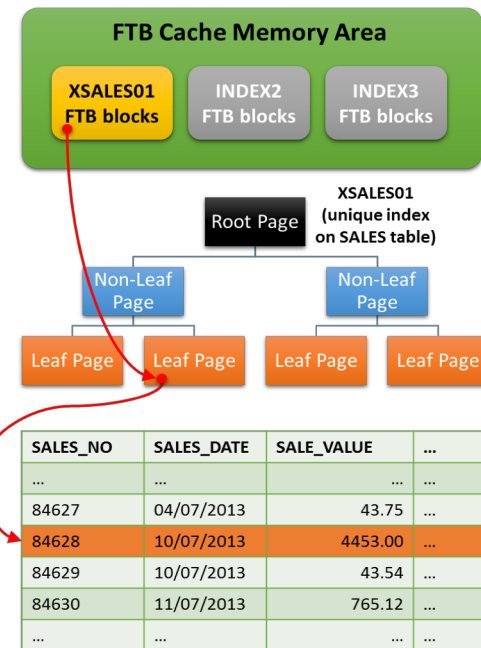
Purpose: based on key value, identify leaf page containing that value

Only a single GETPAGE required for the leaf page (plus a GETPAGE for the data page)

FTB usage is completely transparent to applications

Limited to unique indexes with key length ≤ 64 bytes

```
SELECT SALES_DATE, SALE_VALUE ...  
FROM SALES  
WHERE SALES_NO = 84628
```



If data sharing, don't forget group buffer pools

Sometimes people will make BPx larger across members of a data sharing group, and will forget to enlarge GBPx accordingly

- If aggregate size of local BPs gets too large relative to the size of the corresponding GBP, you could end up with a lot of directory entry reclaims and that's not good for performance
 - Can check on directory entry reclaim activity using output of Db2 command `-DISPLAY GROUPBUFFERPOOL(GBPn) GDETAIL`

For 4K GBP with the default 5:1 ratio of directory entries to data pages, directory entry reclaims likely to be 0 if size of GBP is 37.5% of combined size of associated local BPs

- Example: 3-way group, BP1 at 40K buffers (160 MB) on each member
 - Good GBP1 size is $37.5\% \times (3 \times 160 \text{ MB}) = 180 \text{ MB}$
- In a blog entry I provided a more general approach to GBP sizing

<http://robertsdb2blog.blogspot.com/2013/07/db2-for-zos-data-sharing-evolution-of.html>

Group buffer pools and “XI read hit ratio”

Percentage of GBP synchronous read requests due to local buffer cross invalidation (XI) that resulted in “page found”

(sync reads due to XI, data returned) / (total sync reads due to XI)

- Data available in Db2 monitor statistics long report or online display of GBP activity, or via Db2 command `-DISPLAY GROUPBUFFERPOOL MDETAIL`

Buffer invalidations happen when directory entry reclaims occur, and when a page cached locally by Db2 member X is changed on member Y

- If there are no directory entry reclaims, buffer invalidations must be due to pages being changed on other members of the data sharing group
- If a page was changed on another Db2 member, it had to have been written to the GBP – when you go to the GBP looking for that page, you’re hoping it’s still there

More on the GBP XI read hit ratio

The more data entries there are, the longer pages written to a GBP will stay there, and the higher the XI read hit ratio will go

- I've often seen XI read hit ratios above 80%, even above 90%
- GBP XI read hits are good, because a GBP read usually way faster than disk read

If ALLOWAUTOALT(YES) is specified for a GBP in the CFRM policy, check the GBP's ratio of directory entries to data entries

- Default ratio is 5:1
- Ratio can get very large with ALLOWAUTOALT(YES) in effect
- If you see a really high ratio of directory entries to data entries for a GBP (e.g., 40:1 or more), one effect may be a low XI read hit ratio
- If that's the case, consider enlarging GBP (if CF memory will allow) and changing the ratio of directory to data entries to something closer to 5:1
- Low XI read hit ratio no big deal if few reads due to XI (e.g., < 5/sec)

Exploiting RELEASE(DEALLOCATE)

What's good about RELEASE(DEALLOCATE)?

Package bind option – Db2 will retain certain resources allocated to a thread (table space locks, package sections) until thread deallocation vs. releasing at commit

RELEASE(DEALLOCATE) can save MIPS versus RELEASE(COMMIT) when:

- Thread in question persists through commits (examples are batch threads and CICS protected entry threads)
- and-
- Thread is used repeatedly for the execution of the same package
 - In that case, RELEASE(DEALLOCATE) avoids CPU cost of repeated release and re-acquisition of same table space locks and package sections at each commit

What's the memory-for-MIPS angle?

RELEASE(DEALLOCATE) increases memory utilization because it increases amount of virtual storage used by threads

Before Db2 10, a significant portion of thread-related virtual storage was acquired below the 2 GB “bar” in DBM1

- Talking about the part of the EDM pool used for PT (package table)
 - Copies of package sections allocated to threads
- Use of RELEASE(DEALLOCATE) made more of this space non-stealable, which could potentially lead to program failures due to lack of space in EDM pool

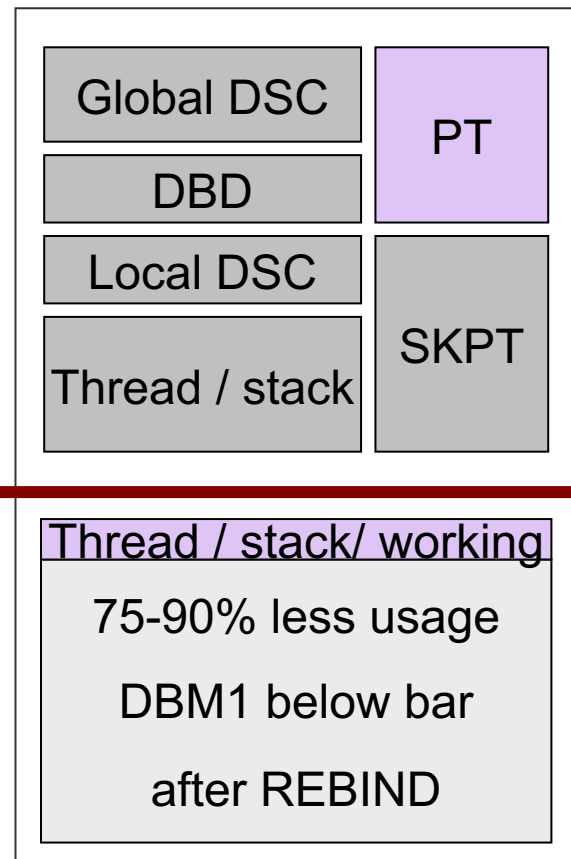
Db2 thread storage

75-90% less usage of below-the-bar DBM1 virtual storage if packages bound or rebound in Db2 10 or later system

– Almost all thread-related storage above 2 GB bar

Also, space for packages allocated to threads no longer in EDM pool – comes instead out of agent local pool

Result: much more virtual storage “head room” for use of RELEASE(DEALLOCATE)



More on RELEASE(DEALLOCATE)

Prior to Db2 10: RELEASE(DEALLOCATE) not honored when package executed via DBAT (i.e., a DDF thread)

- Instead, treated as though bound with RELEASE(COMMIT)
- Why: DBATs can stick around a LONG time, and there was concern that combination of RELEASE(DEALLOCATE) and DBATs would block DDL, binds, etc.

Starting with Db2 10: when package bound with RELEASE(DEALLOCATE) is executed via a “regular” DBAT, that DBAT becomes a high-performance DBAT

- RELEASE(DEALLOCATE) honored
- High-performance DBAT dedicated to connection through which it was instantiated, vs. going back into DBAT pool when transaction ends
- If thread reused for same package, you get CPU benefit of RELEASE(DEALLOCATE)

More on high-performance DBATs

High-performance DBAT will be terminated after being reused 200 times (to free up resources)

Can "turn off" high-performance DBAT functionality by issuing command `-MODIFY DDF PKGREL(COMMIT)`

- Might do that to reduce contention with certain database administration activities (more on this to come)
- Issue `-MODIFY DDF PKGREL(BNDOPT)` to turn high-performance DBAT functionality back on

What if most of your SQL executed through DDF is dynamic?

- Consider binding IBM Data Server Driver (or Db2 Connect) packages into default NULLID collection with `RELEASE(COMMIT)`, and into another collection with `RELEASE(DEALLOCATE)`
 - Higher-volume client-server transactions can use alternate collection to gain high-performance DBAT performance benefits (can specify alternate collection as data source property on client side, or automatically set `CURRENT PACKAGE PATH` to alternate collection name via Db2 profile tables)

RELEASE(DEALLOCATE) recommendations

Best candidates:

- Packages for transactions that are 1) frequently executed, 2) executed via persistent threads, and 3) have low in-Db2 CPU time (like maybe less than 10 ms per transaction)
- Packages associated with batch jobs that issue lots of commits
 - Batch bonus: greater benefit from dynamic prefetch, index lookaside

Operational considerations:

- Ensure that RELEASE(DEALLOCATE) packages do not acquire exclusive table space locks (check for lock escalation, LOCK TABLE)
- What about RELEASE(DEALLOCATE) packages blocking some DDL, bind/rebind actions, and pending DDL-materializing online REORGs?
 - Db2 11 NFM (and Db2 12): those operations can “break in” on persistent threads used to execute RELEASE(DEALLOCATE) packages
 - DDF: still good idea to use -MODIFY DDF command to turn high-performance DBATs off/on as needed

Other ways to trade memory for MIPS

Dynamic statement caching

Global statement cache has been above 2 GB bar in DBM1 address space since Db2 V8

- Size determined by ZPARM parameter EDMSTMTC

Larger statement cache = more cache “hits”

- CPU savings achieved through resulting avoidance of full PREPAREs

I regularly see dynamic statement cache hit ratios of 90% or higher – aim for that on your system

The EDM pool: package and DBD caches

From a Db2 monitor statistics detail report (or online display):

EDM POOL	QUANTITY	/SECOND
-----	-----	-----
PAGES IN DBD POOL (ABOVE)	150.0K	N/A
FREE PAGES	49076.25	N/A
FAILS DUE TO DBD POOL FULL	0.00	0.00
PAGES IN SKEL POOL (ABOVE)	200.0K	N/A
FREE PAGES	75860.15	N/A
FAILS DUE TO SKEL POOL FULL	0.00	0.00
DBD REQUESTS	11264.9K	3182.17
DBD NOT FOUND	12.00	0.00
DBD HIT RATIO (%)	100.00	N/A
PT REQUESTS	39556.5K	11.2K
PT NOT FOUND	107.00	0.03
PT HIT RATIO (%)	100.00	N/A

Nice to have at least 10% free pages

Definitely want zeroes here

For both the DBD cache and the package cache, you want to have a REALLY high ratio of "requests" to "not found" - like thousands to one ("not found" requires read from directory)

Data sets: are you hitting the DSMAX limit?

A DSMAX value set years ago may be too small today

– Result can be a high rate of physical closure of Db2 data sets

- Not good – data set physical open/close operations are relatively expensive

– If you see (in a Db2 monitor statistics report) a lot of data set close activity due to the DSMAX threshold being reached, increase DSMAX

- Can be up to 200,000, but practical limit probably less due to below-the-bar virtual storage consumption – up to 70,000 probably OK in most cases
- Don't go overboard – good value is just large enough to make rate of data set close actions due to threshold reached either zero or very small

OPEN/CLOSE ACTIVITY	QUANTITY	/SECOND
-----	-----	-----
OPEN DATASETS - HWM	13698.00	N/A
OPEN DATASETS	13313.75	N/A
DSETS CLOSED-THRESH.REACHED	0.00	0.00

- Zero is good
- A few per hour is OK
- Several per minute is not good

RID list processing

RIDs, or row IDs, are row location indicators found in index entries

Db2 processes RID lists for things such as:

- List prefetch
- Index ANDing and index ORing

CPU savings can be achieved if RID pool (sized via ZPARM parameter MAXRBLK) is large enough to enable RID processing for a query to complete in memory

- Db2 10: default size of RID pool went to 400 MB, from 8 MB before
- If you see more than a little overflow of RID list processing to work file database, consider increasing size of RID pool
 - RID pool has been above the 2 GB bar in DBM1 since Db2 V8

MXDTCACH

ZPARM specifies amount of memory that can be used for sparse index for a given process (default is 20 MB)

In Db2 monitor accounting and statistics reports/displays:

Sparse index access not used due to insufficient memory - not good (add memory to system, or reduce MXDTCACH, or reduce buffer pool configuration size)

MISCELLANEOUS	AVERAGE	TOTAL
-----	-----	-----
SPARSE IX DISABLED	0.00	0
SPARSE IX BUILT WF	0.36	8

Sparse index built in work file - not necessarily bad, but if you have memory to spare, consider making MXDTCACH larger (e.g., 30 MB or 40 MB) to allow more sparse indexes to be built entirely in memory (boosts CPU efficiency of query execution)

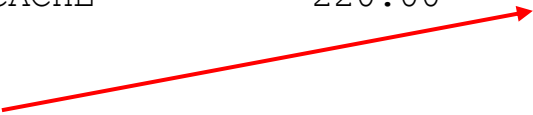
Note: likely to see greater use of sparse index access in Db2 12 environment

CACHEPAC

ZPARM specifies amount of memory to be used for caching of package authorization information

In Db2 monitor statistics reports/displays:

AUTHORIZATION MANAGEMENT	QUANTITY	/SECOND
-----	-----	-----
PKG-AUTH UNSUCC-CACHE	220.00	0.06



- Indicates number of times that package authorization could not be verified using information in cache - requires check of `SYSPACKAUTH` in catalog
- Ideally, this should be single digits per minute or less - if more than that, increase value of `CACHEPAC` in `ZPARM`

Sort pool

This is space in DBM1 (above the 2 GB bar since Db2 V8) that is used for SQL-related sorts (as opposed to utility sorts)

The larger the pool, the more CPU-efficient SQL sorts will be

- Size determined by ZPARM parameter SRTPOOL
- Note that this is the maximum size of the sort work area that Db2 will allocate *for each concurrent sort user*
 - Default size of sort pool went to 10 MB with Db2 10, from 2 MB before
 - Maximum SRTPOOL value is 128 MB – largest I've seen on a Db2 for z/OS system is 48 MB
 - **Db2 12: sort tree has MANY more nodes, so larger SRTPOOL is more likely to increase level of in-memory sorting**
 - As with any other memory-for-MIPS action, if you make SRTPOOL larger, keep an eye on the z/OS LPAR's demand paging rate – you want that to be in the low single digits or less per second

Thanks for your time

Robert Catterall

Senior Consulting Db2 for z/OS Specialist

IBM

—

rfcatter@us.ibm.com