Db2 for z/OS: Exploiting Large Real Storage Resources on IBM Z

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

Tridex Db2 Users Group

September 10, 2020

IBM

Agenda

- The lay of the land
- 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 exploit real storage to enhance Db2 performance

The lay of the land

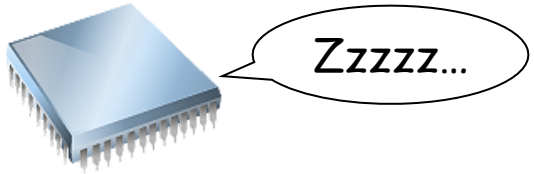
IBM Z server memory resources: getting bigger and bigger

- Not unusual now to see production LPARs with 100s of GB of memory
- Driving the trend towards larger LPAR memory sizes:
  - Desire for balanced configurations: more memory to go with more MIPS
    - Up to 170 engines on a z14, up to 190 engines on a z15 – each engine provides 1000+ MIPS of processing capacity
  - Price of IBM Z memory much lower than it used to be
    - And, like zIIP engines, Z memory does not affect cost of z/OS-related software
- 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

Newer ways to leverage Z memory for better Db2 performance

- All the items below will be 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)

Still plenty of “sleeping gigabytes” out there



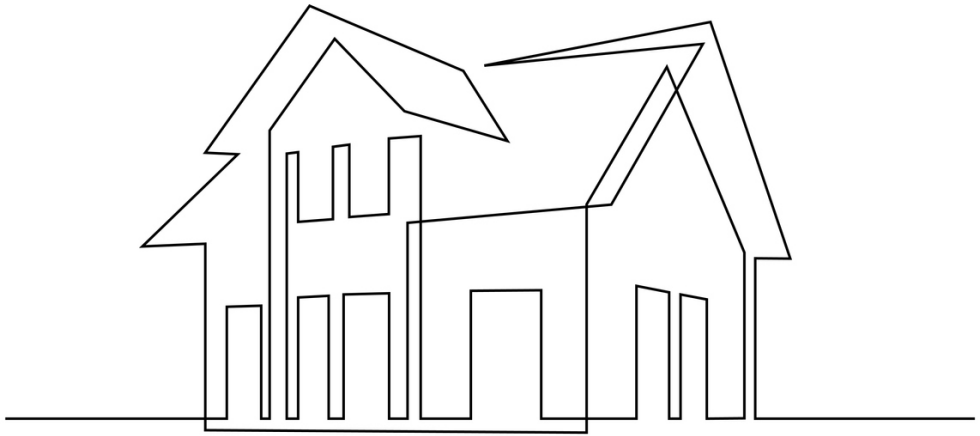
- At many Db2 for z/OS sites, lots of spare memory capacity
- Do you know your z/OS system’s demand paging rate?
  - Available via a z/OS monitor – it is the rate at which pages that have been sent by z/OS to auxiliary storage are paged back into system memory on-demand
  - In my experience, this rate is 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 – 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

Getting your buffer pool house in order

What this means...

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



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: default value of the VPSEQT buffer pool parameter (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 a lot 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 – another 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
    - 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 not recovered when Db2 restarted (no need to do that)
    - Therefore, for work file-dedicated buffer pool, only need a level of changed-page externalization 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 pool, or even 80/50 – but don't take this too far (DMTH threshold hit when 95% of pool's buffers are non-stealable)

Smarter “pinning” with PGSTEAL(NONE)

- When PGSTEAL(NONE) is specified for a pool:
  - When object assigned to pool is first accessed, requesting process will get what it needs and Db2 will asynchronously read rest of the object’s pages into pool
  - If objects assigned to PGSTEAL(NONE) pool have more pages than pool has buffers, Db2 will use FIFO buffer steal algorithm when stealing required
  - Db2 12: an object’s pages are arranged in PGSTEAL(NONE) pool as they are arranged on disk, for more-efficient page access
    - If some buffer stealing is required for PGSTEAL(NONE) pool in Db2 12 system, it will be limited to 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
    - SO, for max performance benefit, size PGSTEAL(NONE) pool in Db2 12 system so that all pages of all objects assigned to the pool will fit within 90% of the pool’s buffers (actually, overflow area of pool will have at least 50 and not more than 6400 buffers)

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

- Some of z/OS LPAR's real storage can be managed as large page frames
  - Can have 1 MB and 2 GB page frames
  - Specified via LFAREA parameter in IEASYSxx member of PARMLIB
- Can use large page frames for PGFIX(YES) buffer pools
  - With large page frames, more CPU-efficient translation of virtual storage addresses to real storage addresses
  - CPU benefit is in addition to savings resulting from less-costly I/Os
    - Important: large frames deliver CPU savings *even for pools with low read I/O rates*
    - I'd go with PGFIX(YES) and FRAMESIZE(1M) or FRAMESIZE(2G) for any pool with GETPAGE rate above 1000/second
  - Use -DISPLAY BUFFERPOOL(*BPn*) DETAIL command to see if Db2 is using large page frames for a PGFIX(YES) buffer pool

Db2 12, 2 GB page frames and PGSTEAL(NONE) buffer pools

- In contiguous part of a PGSTEAL(NONE) buffer pool, a given real storage page frame can hold buffers belonging to one table or index
  - Necessary so that pages of object can be arranged in memory as they are on disk
- Because of that, 2 GB frames cannot be used for PGSTEAL(NONE) pool
  - Reason: if last few pages of an object assigned to the pool go into a 2 GB frame, rest of frame cannot be used – that is a lot of wasted space
- What if PGSTEAL(NONE) and FRAMESIZE(2G) specified for pool?
  - Formerly: PGSTEAL(NONE) honored, FRAMESIZE(2G) ignored (4K frames used)
  - APAR PH22469: FRAMESIZE(2G) honored, PGSTEAL(NONE) ignored (LRU used)
- Probably want to use FRAMESIZE(1M) with PGSTEAL(NONE)
  - If real storage abundant, no big deal if some space in some 1M frames not used

Reduce I/Os by redistributing buffers

- Idea: reduce size of low-I/O pools, increase size of high-I/O pools

  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 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 + list + dynamic) 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 16)?
  - My opinion: less than 1000 read I/Os per second for a pool is good, less than 100 per second is very good, less than 10 per second is great
    - 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) 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 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

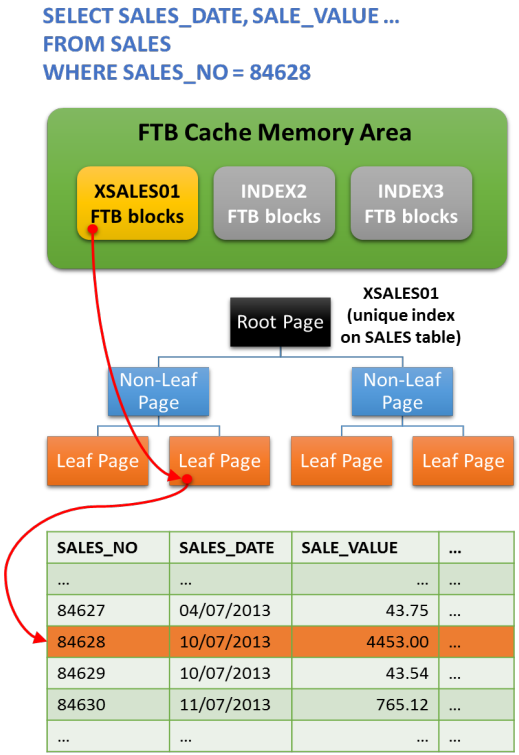
- Again, the measure of “memory stress” that I like to use is the demand paging rate
  - As you implement memory-for-MIPS changes, keep an eye on the z/OS LPAR’s demand paging rate, and don’t let this get out of hand
    - Demand paging rate of 0 is great, between 0 and 1 per second is good, 2-3 per 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 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 – can you get some below 10 per second?
  - For PGSTEAL(NONE) pool, desired read I/O rate is zero – 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, efficiently identify leaf page containing that value
- Only one GETPAGE required for leaf page identified via FTB (versus GETPAGE at every level of index otherwise)
- FTB usage completely transparent to applications
- Limited to unique indexes with key length <= 64 bytes



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
  - In a blog entry I provided an approach to GBP sizing aimed at eliminating directory entry reclaims

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

Another group buffer pool performance metric: XI read hit ratio

- That's the percentage of GBP synchronous read requests *due to local buffer cross invalidation* (XI) that resulted in "page found" in GBP

  (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 A is changed on member B
  - 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 page was changed on another Db2 member, it had to have been written to 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 GBP read usually way faster than disk read
  - For each GBP, check number of GBP sync reads per second that are due to XI
    - Reads with data returned + reads with no data returned, per second
  - If more than 10 sync reads due to XI per second for a GBP (if less than 10 per second, who cares?), check the XI read hit ratio for that GBP
    - If that XI read hit ratio value for the GBP is less than 80%, and there is sufficient CF LPAR memory to enlarge the GBP, consider doing to to increase XI read hit ratio
    - If you enlarge GBP for this reason, consider lowering GBP's directory-to-data ratio (at least some) as well, to get even more GBP data entries

Exploiting RELEASE(DEALLOCATE)

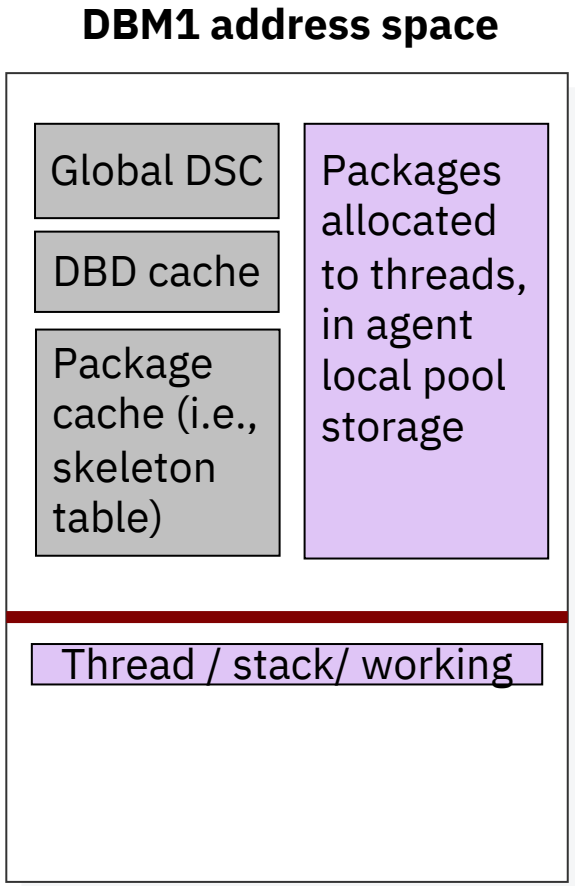
What's good about RELEASE(DEALLOCATE)?

- 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 vs. 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

RELEASE(DEALLOCATE) and memory

- RELEASE(DEALLOCATE) + persistent threads means more use of memory versus RELEASE(COMMIT), but almost all thread-related virtual storage usage is above 2 GB bar
- So, if real storage isn't under too much pressure (if LPAR's demand paging rate is 0 or < 1 per second), use of RELEASE(DEALLOCATE) packages with persistent threads is a good use of more memory to boost CPU efficiency



RELEASE(DEALLOCATE) and DBATs (i.e., DDF threads)

- 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
- -MODIFY DDF PKGREL(COMMIT) can be used to “turn off” high-performance DBATs (DDF work still gets done, using “regular” DBATs)
  - Might do that to reduce contention with some database administration activities
  - -MODIFY DDF PKGREL(BNDOPT) turns high-perf 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 automatically SET CURRENT PACKAGE PATH to alternate collection name via Db2 profile tables)

RELEASE(DEALLOCATE): best candidates

- Packages that are 1) frequently executed, 2) executed via persistent threads, and 3) have low in-Db2 CPU time (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

RELEASE(DEALLOCATE): 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?
  - Starting with Db2 11 NFM: those operations can “break in” on persistent threads used to execute RELEASE(DEALLOCATE) packages
  - Db2 12 function level V12R1M505 introduced rebind phase-in for non-disruptive package rebind
  - 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 DBD cache and package cache, you want REALLY high ratio of "requests" to "not found" - like thousands to one ("not found" requires read from directory)
- If that ratio is not really high for one of these caches, make the cache larger if LPAR has the memory for that (demand paging rate 0 or < 1 per second)

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

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:

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

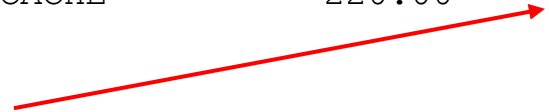
Sparse index access not used due to insufficient memory – not good (add memory to system, or reduce MXDTCACH, or reduce buffer pool configuration size)

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)

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
- Similar story for CACHERAC – used for routine authorization

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
    - Max SRTPOOL value is 128 MB – largest I've seen on Db2 for z/OS system is 48 MB
    - Db2 12: sort tree has MANY more nodes, so larger SRTPOOL 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 – ideal for that is 0 or < 1 per second

Thanks for your time

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