Let Me Make This Clear: Explaining Often-Misunderstood Db2 for z/OS Concepts and Facilities

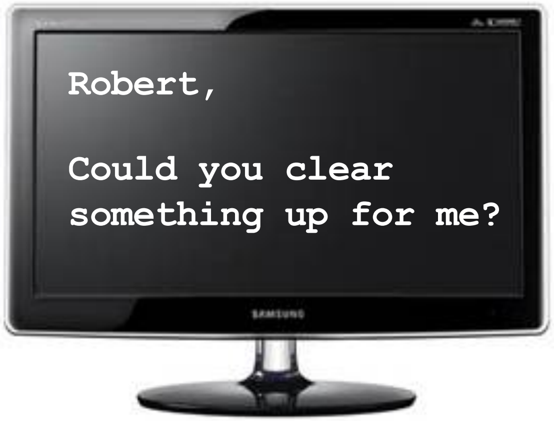
Wisconsin Db2 Users Group

September 11, 2019

Robert Catterall, IBM
rfcatter@us.ibm.com

Introduction

- In the course of my work, I get a lot of questions from a lot of Db2 for z/OS people



Robert,

Could you clear something up for me?

- Some of these questions suggest widespread misunderstanding of certain Db2 features and functions

- In this presentation, I'll highlight some of these misunderstandings and provide (I hope) some clarity
  - I'll highlight misunderstandings in *dark red italics*

Agenda

- zIIP offload and native SQL procedures
- Buffer pool monitoring
- Page-fixed buffer pools and large page frames
- The PGFIX and PGSTEAL buffer pool specifications
- Prefetch reads
- Partition-by-growth and smaller tables
- Transparent archiving: base and archive tables
- Db2 address space prioritization
- Re-using DDF threads
- High-performance DBATs
- “Hard” and “soft” data set close
- Db2 data set extents

zIIP offload and native SQL procedures

- *A lot of people are under the impression that native SQL procedure execution is always zIIP-eligible*
- In fact, native SQL procedure execution is <u>only</u> zIIP-eligible when CALL comes through Db2 distributed data facility, aka DDF

Handles requests from applications that are network-connected to Db2 system

  - Why? Because zIIP eligibility depends on process running under preemptible SRB versus TCB or non-preemptible SRB
  - Native SQL procedure runs under task of process that called it – when caller is network-connected to Db2, that task is a <u>preemptible SRB in the Db2 DDF address space</u>

That makes native SQL procedure zIIP-eligible

More on native SQL procedure zIIP eligibility

- When a native SQL procedure is called by a process that runs under a TCB (e.g., a batch job), it will run under that TCB and so will not be zIIP-eligible
- Suppose a DDF-using application calls an external Db2 stored procedure, and that stored procedure calls a native SQL procedure
  - Q: Will native SQL procedure’s execution be zIIP-eligible?
  - A: NO
    - An external stored procedure always runs under its own TCB in a stored procedure address space, so nested native SQL procedure will run under that TCB and will not be zIIP-eligible

Note: external stored procedure called through DDF will get a little zIIP offload, for send/receive processing

Still on the topic of native SQL procedures and zIIPs

- *Some people think that native SQL procedures, when zIIP-eligible, are 100% zIIP-offload-able*
- Nope – when a native SQL procedure is zIIP-eligible (i.e., when the CALL comes through the Db2 DDF), it will be up to 60% zIIP-offload-able
  - Why? Because SQL statements that execute under preemptible SRBs in Db2 DDF address space get up to 60% zIIP offload, and a native SQL procedure is just SQL
  - Implication: moving SQL issued directly from DDF-using applications to native SQL procedures will not boost zIIP offload – the SQL is up to 60% zIIP-eligible either way
    - You can boost zIIP offload when you change external stored procedures called by DDF-using applications to native SQL procedures

zIIP offload-o-meter

Buffer pool monitoring

- *Lots of people think that "hit ratio" is the key metric when it comes to buffer pool monitoring and tuning*
- My opinion: buffer pool hit ratio has little value 👎
- I focus on a buffer pool's total read I/O rate
  - That's total synchronous reads plus total prefetch reads (sequential, list, dynamic) for a buffer pool, per second
    - Can get these numbers from Db2 monitor statistics long report or online display of buffer pool activity, or from Db2 command -DISPLAY BUFFERPOOL DETAIL
  - What's your objective for a buffer pool?
    - Total read I/O rate < 1000/second is good, < 100/second is very good (some sites go for < 10/second with "super-sized" pools)
    - **Note:** for buffer pool used to "pin" objects in memory, objective is total read I/O rate of zero

Page-fixed buffer pools and large page frames

- *Some people think that page-fixing is only good for high-I/O buffer pools*
- Good for such pools (cheaper I/Os), but also good for high-activity pools – *even those with little I/O activity* – if it leads to use of large page frames (1M or 2G)
  - A pool with > 1000 GETPAGEs/second is “high activity”
  - Availability of large page frames depends on LFAREA specification (in IEASYSxx member of SYS1.PARMLIB)
  - Use of large frames for buffer pool requires PGFIX(YES) specification (and FRAMESIZE(1M or 2G) specification, starting with Db2 11 for z/OS)
  - Why large page frames are good for high-activity pools: they make translation of virtual storage to real storage addresses more CPU-efficient

PGFIX(YES) and large page frames

- *My PGFIX(YES) buffer pools are not being backed by large page frames – why?*
- First, how do you know if a PGFIX(YES) buffer pool is backed by large page frames?
  - Issue Db2 command -DISPLAY BUFFERPOOL(ACTIVE) DETAIL, and check output for each pool

```
DSNB402I  *DBP1 BUFFER POOL SIZE = 250000 BUFFERS
DSNB406I  *DBP1 PGFIX ATTRIBUTE -
                CURRENT = YES
                PENDING = YES
DSNB546I  *DBP1 PREFERRED FRAME SIZE 1M
          0 BUFFERS USING 1M FRAME SIZE ALLOCATED
DSNB546I  *DBP1 PREFERRED FRAME SIZE 1M
          250000 BUFFERS USING 4K FRAME SIZE ALLOCATED
```

PGFIX(YES) in effect for this pool

1 MB is preferred page frame size for PGFIX(YES) buffer pool

NONE of this pool's buffers are backed by preferred 1 MB page frames

- Why are 1 MB page frames, though preferred, not used for this pool?

To use large page frames, must have them

- If large page frames not being used for PGFIX(YES) buffer pool, may be that LPAR has no large frames, or has some but not enough for the pool
- In setting LFAREA value for LPAR, don’t go overboard
  - If wanting to use 2 GB page frames (Db2 11 or later), specify LFAREA “G” value that will accommodate Db2 buffer pools (if any) defined with FRAMESIZE(2G)
    - When using FRAMESIZE(2G) for buffer pool, may want VPSIZE (number of buffers) for 4K pool to be multiple of 524,288
  - Specify LFAREA “M” value sufficient for buffer pools that will use 1 MB frames, plus a bit of a pad to cover other uses of 1 MB frames – pad of about 5% should be OK
    - If you use WebSphere Application Server for z/OS, keep in mind that large page frames can be used for Java heap, and plan accordingly

PGFIX and PGSTEAL buffer pool specifications

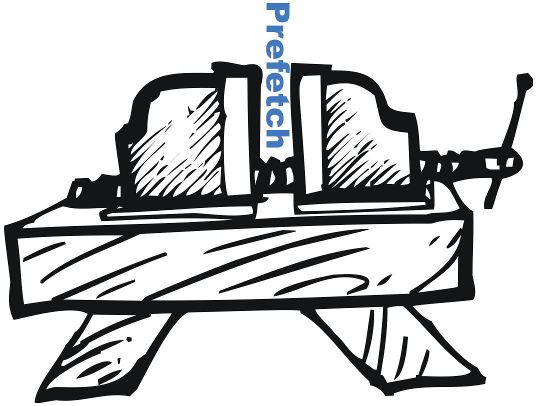
- *Some folks think these specifications are interrelated*
- In fact, PGSTEAL and PGFIX are independent of each other – setting one does not affect behavior pertaining to the other
- Difference: PGSTEAL relates to Db2 management of its buffer resources (*virtual storage*), while PGFIX has to do with z/OS management of *real storage*
  - Db2's management of buffers in pool, as indicated by PGSTEAL setting (LRU, FIFO, or NONE), is NOT affected by a pool's PGFIX attribute (YES or NO)
  - Similarly, z/OS's management of real storage used by buffer pool, as indicated by pool's PGFIX setting, is NOT affected by way in which Db2 manages the pool's buffers (PGSTEAL)

More on PGSTEAL

- Db2 will always steal a buffer in a pool whenever it has to (it has to if all buffers are occupied and a new page has to be read into memory)
  - PGSTEAL(NONE): “Buffer stealing should not be necessary for this pool, because it has more buffers than there are pages belonging to objects assigned to the pool”
    - “If you have to steal buffers, use FIFO algorithm (first in/first out)”
  - PGSTEAL(FIFO): “Some buffer stealing likely required”
    - “Don’t waste CPU cycles keeping track of which buffer has gone longest time without being accessed – when stealing is required, take buffer holding page that has been in memory longest time”
  - PGSTEAL(LRU): “A lot of buffer stealing likely required for this pool”
    - “When you steal a buffer, take the one holding the page that’s gone the longest time without being accessed”

Buffer pool prefetch reads

- *Some people believe: prefetch reads not important*
- Prefetch reads may not be as important as synchronous reads, but they do matter
  - That is why the most important performance metric for a buffer pool is (as previously noted) the total read I/O rate: all synchronous reads plus all prefetch reads, per second
  - Db2 person, thinking that prefetch reads don't matter much, may take a pool's VPSEQT setting WAY down – from default of 80 to maybe 20 – in effort to reduce synchronous reads
    - VPSEQT: % of buffers in pool that can hold pages brought into memory via prefetch (VPSEQT does not limit buffers available for synchronous reads – ALL buffers available for those)



What happens when a very low VPSEQT value puts too much of a squeeze on prefetch?

When buffer pool VPSEQT is too low

- Might drive up prefetch reads – performance impact
  - Increased “wait for other read” time (Db2 accounting report)
- Might drive up synchronous reads (check *sequential synchronous reads* in Db2 statistics report or output of -DISPLAY BUFFERPOOL DETAIL)
  - Maybe prefetched pages getting “flushed” from memory before being accessed by application process
  - Perhaps prefetch sometimes disabled – check Db2 monitor statistics long report or -DISPLAY BUFFERPOOL DETAIL
    - **PREFETCH DISABLED - NO BUFFER**
    - **PREFETCH DISABLED - NO READ ENGINE**
- Not saying, “Don’t ever lower VPSEQT” – I’m saying:
  - Have reason for making change (simulate effect via SPSEQT option of -ALTER BUFFERPOOL)
  - A relatively modest change could yield the benefits you seek

Partition-by-growth (PBG) and smaller tables

- *Some people think that PBG table spaces are only appropriate for large tables*
- Not so
  - People under this impression may be influenced by the word "partition," which traditionally (before universal table spaces) was associated with large tables
  - A PBG table space's DSSIZE (smallest value is 1 GB) is the space-used value that triggers allocation of an additional partition for the table space
  - Small table won't grow to the DSSIZE value, so PBG will be 1-partition table space
  - DSSIZE value doesn't determine disk space utilization – that's determined by amount of data in table, PRIQTY, and SECQTY

Transparent archiving: base and archive tables

- Db2-managed archiving (also known as "transparent archiving") was introduced with Db2 11 for z/OS
  - Optimize access performance for "popular" rows (i.e., frequently-accessed rows; often, recently inserted rows) by concentrating them in a base table
  - "Older and colder" rows are physically stored in a separate archive table, but applications see a single logical table
- *Some people think that an archive table has to be identical to its associated base table, in every way*
- In fact, what is required is that a base table and its archive table are <u>logically</u> identical
  - Same column names, in same order, with same data type
- Can a base table and its archive table be different with regard to <u>physical</u> design?

Archive table “physical design freedom”

- Consider medium-sized base table, huge archive table
  - If base table is in a traditional segmented table space, could associated archive table be in a partitioned table space?
  - **Sure!** And, it could be a partition-by-range or a partition-by-growth table space – doesn’t matter!
    - Table spaces have to do with physical database design – all you need is logical equivalence
  - If base table has 8 indexes, must same indexes be defined on the associated archive table?
  - **No!** Maybe it is understood that queries targeting older rows (stored in an archive table) will not perform as well as queries targeting newer rows (in a base table)
    - In that case, reduce number of indexes defined on archive table
    - Save disk space, reduce INSERT/DELETE overhead, make utility execution more CPU-efficient

And, this “physical design freedom” applies as well to a base and a history table, when using Db2 system-time temporal functionality!

Archive table physical design idea

- Partition archive table on row-change timestamp column – GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
  - Limit key value for each partition could be, for example, one week greater than limit key value of preceding partition
- If base table does not already have row-change timestamp column, add that to base table (to maintain logical equivalence between base and archive tables)
  - No need to partition base table on row-change timestamp (physical difference OK)
- When a row is deleted from base table with MOVE_TO_ARCHIVE global variable set to 'Y', it will go into the archive table partition *that holds rows moved from base to archive this week*

Db2 address space prioritization

- *At many sites, one or more Db2 address spaces are given a too-low priority in z/OS LPAR's WLM policy*
  - Result: degraded throughput for Db2-accessing applications
- First, IRLM should be assigned to super-high-priority SYSSTC service class
  - When IRLM has work to do, it needs a processor right away; otherwise, lock acquisition and release is delayed
  - IRLM uses very little CPU, so no worries about it getting in the way of other address spaces if it has a higher priority than those other address spaces
- Assign other Db2 address spaces to SYSSTC?
  - I say, "No" – remember what "Syndrome" said in "The Incredibles?"

"When everyone is super, no one will be"

Prioritizing Db2 address spaces other than IRLM

- DIST and stored procedure address spaces should have same priority as MSTR and DBM1 (below SYSSTC, above CICS AORs or IMS MPRs)
  - *Some people give these Db2 address spaces a priority below CICS AORs, fearing that the Db2 address spaces will block CICS access to processors*
    - In fact, if Db2 tasks wait behind CICS tasks for processor time, CICS-Db2 transaction performance will suffer (CICS monitor will show higher "wait for Db2" times)
  - *Some folks give DIST address space (DDF) lower priority than other Db2 address spaces, fearing that higher priority will be too high for SQL coming through DDF*
    - Priority of DIST address space applies only to DDF system tasks – these use very little CPU
    - Priority of SQL statements coming through DDF depends on service class(es) to which DDF applications are mapped – if not mapped to service class, default priority is discretionary

↖ Very low

More on prioritizing Db2 address spaces

- *Stored procedure address spaces may have a lower priority vs. other Db2 address spaces, because people don’t want stored procedure priority to be too high*
  - In fact, priority at which a Db2 stored procedure executes <u>is inherited from process that called the stored procedure</u>
  - If stored procedure address space has too-low priority, result can be delays in scheduling called stored procedure for execution (negatively impacts callers of stored procedures)
  - Note: native SQL procedures, like external stored procedures, run at the priority of the calling process, but they execute in DBM1 under the caller’s task

Reusing pooled DBATs (DBATs = DDF threads)

- *How can I monitor reuse of pooled DBATs?*
- Look at Db2 monitor statistics long report (or online display), in the GLOBAL DDF ACTIVITY section:

```
GLOBAL DDF ACTIVITY               QUANTITY
---------------------------       --------
DBATS CREATED                       241.98
DISCON (POOL) DBATS REUSED         1399.0K
```

Db2 needed a DBAT 1,399,242 times during the reporting interval, and needed to create a DBAT 242 times – that's a 99.98% rate of thread reuse

- If rate of pooled DBAT reuse is less than you'd like it to be, consider increasing POOLINAC in ZPARM
  - That's the amount of time a DBAT can sit there in the pool without being reused (default is 120 seconds) – the DBAT will be terminated at that point
  - Perhaps your pooled DBATs are often terminated just before they would have been reused – in that case, modest increase in POOLINAC might significantly boost pooled DBAT re-use

High-performance DBATs – PKGREL(BNDOPT)

- *Some folks have RELEASE(DEALLOCATE) packages executed by way of DBATs, and they don't see any high-performance DBAT activity – why?*
- First of all, what do I mean by "don't see any high-performance DBAT activity?"
  - Check a Db2 monitor statistics long report (or online display), in GLOBAL DDF ACTIVITY section:

```
GLOBAL DDF ACTIVITY            QUANTITY
---------------------------    --------
HWM ACTIVE DBATS-BND DEALLC        0.00
```

- Question: Why no high-performance DBATs?
  - Answer: DDF is not enabled for high-performance DBATs

Enabling DDF for high-performance DBATs

- Is enabled for high-performance DBATs?
  - Issue Db2 command -DISPLAY DDF DETAIL, check output:

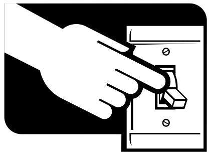
```
DSNL106I PKGREL = COMMIT
```

← DDF not enabled for high-performance DBATs

  - To change situation, issue Db2 command -MODIFY DDF PKGREL(BNDOPT)
  - Might be necessary sometimes to “turn off” high-performance DBATs to accomplish some database administration tasks:
    - BIND REPLACE or REBIND of, or action requiring invalidation* of, RELEASE(DEALLOCATE) package allocated to high-performance DBAT

* ALTER of, or pending DDL-materializing online REORG of, object on which package is dependent

  - To temporarily turn off high-performance DBATs, issue command -MODIFY DDF PKGREL(COMMIT)
    - Frees up even RELEASE(DEALLOCATE) packages allocated to high-perf DBATs that are not in-Db2



  - Turn them back on: -MODIFY DDF PKGREL(BNDOPT)

“Hard” and “soft” closing of Db2 data sets

- *There is some confusion out there as to what these terms mean, and about the effect of CLOSE YES and CLOSE NO for a table space or index*
- “Hard close” is sometimes used to describe physical closing of Db2 table space or index data set (could be associated with partition of a table space or index)
  - Aside from shutdown of Db2 subsystem, what would cause Db2 data set to be physically closed?
    - In any Db2 environment, when number of data sets open and allocated to Db2 subsystem reaches limit specified via DSMAX in ZPARM, Db2 will physically close some of the open data sets
    - *In a Db2 data sharing system,* data set physical close actions can also be triggered by “soft close” activity

Next slide

“Soft close” of data sets explained

- “Soft close” is another term for *pseudo-close*
  - Pseudo-close occurs when Db2 data set open for read/write goes for pseudo-close interval of time with no update activity
  - Pseudo-close interval is determined by 2 ZPARM parameters: PCLOSEN (a number of Db2 checkpoints) and PCLOSET (a number of minutes) – the default value for both is 10
    - Whichever occurs first since last update of data set – PCLOSEN checkpoints or PCLOSET minutes – triggers pseudo-close
  - At data set pseudo-close, state is changed from read/write to read-only – recorded in SYSLGRNX table in Db2 directory
    - Next data-change action affecting pseudo-closed data set will change state back to read/write – also recorded in SYSLGRNX
  - Primary purpose of pseudo-close is to speed up table space and/or index recovery operations
    - For log-apply processing, RECOVER utility uses SYSLGRNX information to skip over portions of log covering time periods during which the object being recovered was in a read-only state

What about CLOSE YES and CLOSE NO?

- CLOSE YES/NO specification for table space/index affects “hard close” processing, but not “soft close”
  - When DSMAX reached, Db2 closes some data sets, *starting with those defined with CLOSE YES* (CLOSE YES data sets that have gone longest without being referenced closed first)
    - Thus, CLOSE YES/NO lets you influence which data sets will be physically closed when DSMAX is reached
    - That said, my preference is to set DSMAX high enough so that it is rarely, if ever, reached (high possible value of DSMAX is 200,000 – value up to about 70,000 generally no problem)

CLOSE YES/NO and Db2 data sharing

- Suppose a data set in which there is inter-Db2 read/write interest gets pseudo-closed on member DB2A of a data sharing group
  - If that data set goes another pseudo-close interval with no access at all from DB2A, it will be physically closed on DB2A if object defined with CLOSE YES
  - Physical close on DB2A could result in member DB2B getting an exclusive page set P-lock on the data set – that would reduce data sharing overhead for member B
  - But, if CLOSE YES widely used and there is a lot of pseudo-close activity, data sets could be frequently going into and out of group buffer pool dependency, increasing overhead
  - CLOSE YES could be good for objects that regularly end up being accessed from only one member

Extents

- *Some people are concerned about Db2 data sets going into extents*
- I’m not – here’s why:
  - In most production Db2 for z/OS systems, vast majority of GETPAGEs satisfied from buffer pool
  - Typically, the large majority of GETPAGEs that are <u>not</u> satisfied from a buffer pool result in a read from disk controller cache, not spinning disk
  - Even when reads from spinning disk occur, architecture of disk subsystems typically used with IBM Z servers is such that extents are of little consequence, performance-wise

What about hitting extent limit?

- Highly unlikely
  - Limit is 7257 if z/OS at 1.7 or later, data set is SMS-managed, and Extent Constraint Removal option is set to YES in the SMS data class
  - Otherwise, it’s 251
- Recommendation: let Db2 manage secondary disk space allocation
  - How: MGEXTSZ YES in ZPARM, SECQTY not specified for CREATE TABLESPACE/INDEX, or SECQTY set to -1
  - Benefit: virtually assured of avoiding extent limit, and likely to avoid hitting max # of volumes across which data set can be spread (59)

Thanks for your time.

Robert Catterall, IBM
rfcatter@us.ibm.com