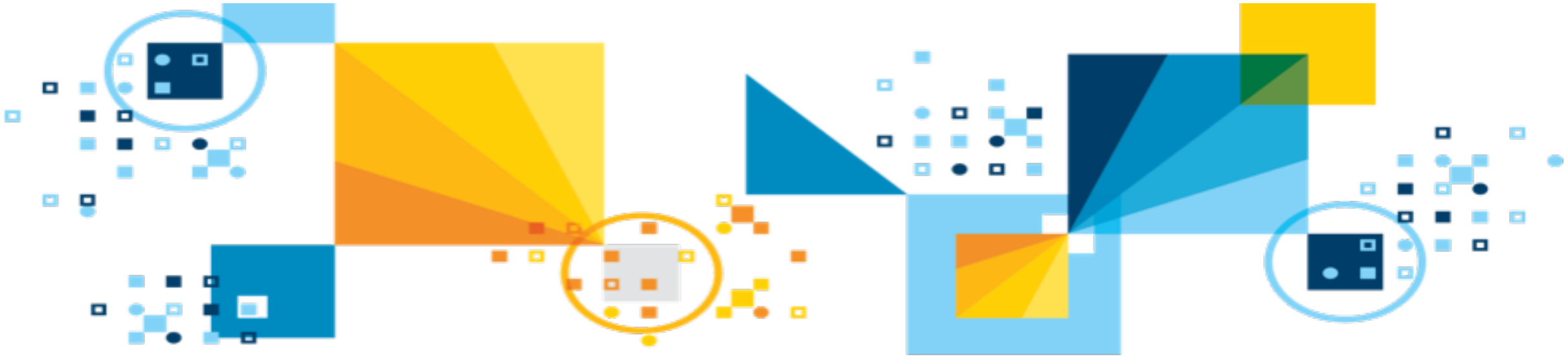
New England Db2 Users Group
June 21, 2018

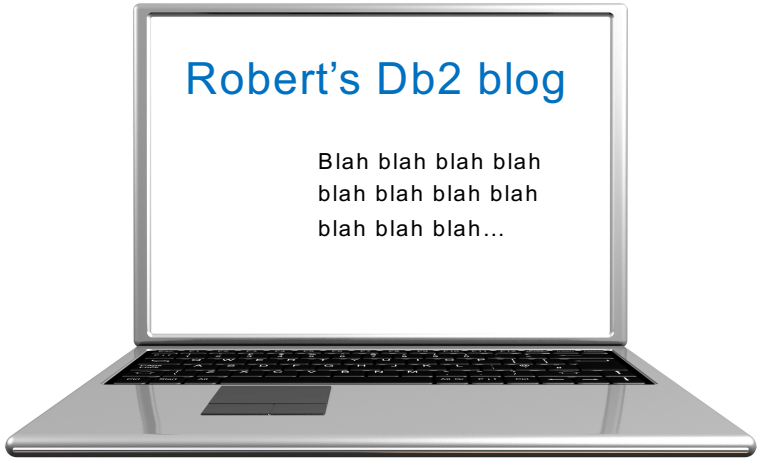
Robert's Db2 for z/OS Blog: Best of the Blog, Volume 1

Robert Catterall
IBM Senior Consulting Db2 for z/OS Specialist



Introduction

- I have a Db2 for z/OS blog, called Robert's Db2 blog (http://robertsdb2blog.blogspot.com), to which I've been contributing for the past eight years
- This blog gets quite a few page views, and a number of folks have indicated that it's a useful source of technical Db2 for z/OS-related information
- I thought, "How about creating a presentation based on entries in my Db2 blog that seem to be favorites of readers, and some that are favorites of mine?"
- Thus this session…



Contents

Monitoring Db2 for z/OS: What's in YOUR Subsystem?

(http://robertsdb2blog.blogspot.com/2011/03/monitoring-db2-for-zos-whats-in-your.html)

- The main point: you should know what's running in your Db2 for z/OS system – key tool for this purpose is reports generated by your Db2 monitor
  - Online monitoring capabilities of your monitor are important, but analyzing reports is best route to discovery (a-ha!) and in-depth knowledge of environment
  - If you haven't used your Db2 monitor to generate reports, DO IT (monitor vendor can help)
- Which reports? Two are critically important:
  - Statistics Report - Long (depending on monitor, may be called Statistics Detail Report)
    - Provides "system view" of Db2 performance
  - Accounting Report - Long (depending on monitor, may be called Accounting Summary - Long)
    - Provides "application view" of Db2 performance
- Generate these reports for the same time period (maybe a busy hour of a busy day), and use them together

More on monitoring: Accounting Report - Long

- The really valuable form of this valuable report: data ordered by connection type (i.e., aggregated at connection-type level)
  - Within larger accounting report, you'll have sub-reports for each connection type:
    - One showing all DDF-related work (connection type: DRDA)
    - One showing all CICS-Db2 work (connection type: CICS)
    - One showing all batch work using call attach facility (connection type: DB2CALL)
    - Etc.
  - Very useful exercise: for each Db2 workload component (i.e., each connection type), determine aggregate CPU cost of SQL statement execution
    - ((avg. class 2 general-purpose CPU time) + (avg. class 2 zIIP CPU time)) * (# of occurrences)
    - Why this calculation: because "average" is "average per occurrence" ("occurrence" = trace record)
    - Track this over time – at many sites, DDF-related work (DRDA connection type) is the fastest growing (and increasingly the very largest) component of a Db2 system's overall workload
    - Think mainframes just do "legacy" work? Db2 DDF workload growth says otherwise

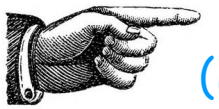
Statistics Report - Long, and management of Db2 monitor reports

- The Statistics Report - Long contains a lot of very valuable information, to which I'll refer in subsequent slides in this presentation
  - Of particular utility: numbers showing activity pertaining to buffer pools, data set open/close, package and DBD caches in the EDM pool, the dynamic statement cache, and the DDF
- Report management (a great approach I've seen):
  - Generate Statistics Report - Long and Accounting Report - Long (latter with data ordered by connection type) on a daily basis (same hour of day, or maybe pair of reports – same "online" hour and same "batch" hour), and "print" them to GDGs with (for example) 60 data sets
  - That way, you have a rolling collection of reports for the most recent 60 days – easy to select and browse any of them
    - Fantastic resource for tracking workload and gauging effect of system and application changes

Db2 for z/OS: Trading Memory for MIPS (Part 1)

(http://robertsdb2blog.blogspot.com/2012/03/db2-for-zos-trading-memory-for-mips.html)

- (there are “part 2” and “part 3” entries, as well)
- The main point: IBM Z servers these days often have LOTS of real storage (hundreds of GB, or maybe several TB) – LEVERAGE IT!
  - Increasingly, organizations view mainframe memory much as they view zIIP engines: as a hardware resource that can boost application performance without increasing software costs
  - I like to see 20 GB or more of memory per engine (zIIP or general-purpose) for z/OS LPAR
    - So, if LPAR has 5 general-purpose and 5 zIIP engines, should have at least 200 GB of memory
    - This ratio of memory to compute power delivers what I feel is a balanced configuration
- No z/OS subsystem takes advantage of memory like Db2
  - Greatest single leverage point: Db2 buffer pools
    - Your objective: drive down the total read I/O rate for a pool (more on this to come)
    - Also leverage fixed-in-memory buffers and large real storage page frames (more on this to come)
    - Also consider using buffer pools to “pin” objects in memory…



(next slide)

More memory-for-MIPs – “pinning” pools

- If your z/OS system has lots of memory, why not use a Db2 buffer pool (or pools) to cache certain objects (tables and/or indexes) in memory in their entirety?
  - Best way to do that: define buffer pool with PGSTEAL(NONE)
    - Db2 will asynchronously read every page of an object assigned to a PGSTEAL(NONE) buffer pool into the pool when object is first accessed after pool has been allocated
    - Db2 12 enhancement: pages in PGSTEAL(NONE) pool will be arranged in memory just as they are arranged on disk – enhances efficiency of page access
    - Best objects for PGSTEAL(NONE) pools: table spaces and/or indexes that a) are accessed really frequently and b) don’t have a humongous number of pages
    - Almost certainly want to use large page frames for PGSTEAL(NONE) pool (more on this to come)
    - **Note:** if Db2 needs to steal buffers in a PGSTEAL(NONE) pool – if there are not enough buffers in the pool to hold all pages of all objects assigned to the pool – it will do that, using FIFO algorithm (“NONE” means buffer stealing is not expected for the pool – doesn’t mean it cannot happen)

Memory-for-MIPS – beyond buffer pools

- Bigger buffer pools great for Db2 performance, but don't stop there
- Some other beneficial uses of Big Memory in a Db2 for z/OS system:
  - Enlarge the package cache (aka the "skeleton pool") in the EDM pool if the ratio of PT requests to "PT not found" is less than several thousand to one (PT = "package table")
    - DBD cache in EDM pool usually large enough, but check that ratio of requests to "not found," too
  - Increase size of dynamic statement cache in EDM pool if hit ratio for that cache < 90%
  - Increase DSMAX in ZPARM if more than a few data sets closed due to hitting "max open" threshold
  - Consider increasing size of sort pool (SRTPOOL in ZPARM) to get more in-memory sorts (especially in a Db2 12 environment – sort tree can have more nodes than before)
  - Increase value of ZPARM parameter MXDTCACH (area in memory for building sparse indexes) if Statistics Report - Long shows more than few instances of "SPARSE IX BUILT WF" (shows how many times sparse index built in work file on disk because in-memory area too small)
  - Use RELEASE(DEALLOCATE) bind option for more packages that are a) frequently executed and b) executed using persistent threads (more on this to come)

Db2 for z/OS: Get Your DDF-Related Dispatching Priorities Right

(http://robertsdb2blog.blogspot.com/2013/08/db2-for-zos-get-your-ddf-related.html)

- The main point: with DDF access to Db2 growing (at many sites, it is the largest component of the overall Db2 workload), it becomes more and more important to properly prioritize DDF and DDF-using applications in a z/OS system
- A sign that it may not be right: high in-Db2 not-accounted-for time for DDF workload
  - Where to find this: in a Db2 monitor Accounting Report - Long, with data ordered by connection type, look in the sub-report for the DRDA connection type
  - How it's calculated: subtract in-Db2 CPU time (general-purpose and zIIP) and total class 3 wait time ("known" wait times) from in-Db2 elapsed time
  - In-Db2 not-accounted-for time is generally a reflection of wait-for-dispatch time
  - In-Db2 not-accounted-for time should be less than 10% for a higher-priority transactional workload, such as CICS-Db2, or IMS-Db2, or the DDF workload
  - If in-Db2 not-accounted-for time high for DDF and for CICS-Db2 (or IMS-Db2): CPU-constrained
  - If low for CICS-Db2 and/or IMS-Db2 and high for DDF, DDF-related priorities may not be right

DDF-related priorities: system tasks

- DDF itself should have same priority as Db2 MSTR and DBM1 address spaces
  - That priority should be below SYSSTC (IRLM is the only Db2 address space that should be assigned to SYSSTC service class) but above CICS AORs or IMS MPRs
    - Rationale: “system” tasks should run at a higher priority than application tasks
- Why some sites give DDF lower priority than Db2 MSTR, DBM1: “We don’t want SQL statements coming through DDF to run at the priority of MSTR and DBM1”
  - They won’t! DDF priority only applies to DDF “system” tasks – those tasks use very little CPU
    - Can verify by looking at DDF address space CPU time in Db2 monitor Statistics Report - Long
    - TCB and non-preemptable SRB time = DDF system tasks, preemptable SRB time = application tasks
    - DDF system CPU time often less than 1% of total DDF CPU time

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	PREEMPT IIP SRB
SYSTEM SVCS	23.203	13:18.792	10.465	55.128
DATABASE SVCS	58.468	1:26.688	17.226	13:49.714
IRLM	0.006	0.000	6:00.680	0.000
DDF	5.742	9:36.359	2.596	12:23.363

DDF-related priorities: application and stored procedure tasks

- If SQL coming through DDF does not inherit DDF's priority, at what priority does it run?
  - Answer: SQL runs at priority of service class to which DDF-using application has been assigned in the LPAR's WLM policy (WLM = z/OS workload manager)
    - If DDF-using application not assigned to service class, gets SYSOTHER by default and runs with DISCRECTIONARY priority – probably not what you want (that's a low priority)
    - If you set up service classes for DDF-using applications long ago, are the associated priorities appropriate given the current business importance of the applications? If the priorities are lower than those of CICS-Db2 and/or IMS-Db2 transactions, should they be?
- And, if you use external Db2 stored procedures, stored procedure address spaces should have same priority as Db2 MSTR, DBM1, and DIST (DIST = DDF)
  - This priority applies to the stored procedure address space's "main task" – NOT to stored procedures themselves (stored procedure runs at priority of calling application)
  - If stored procedure address space has too-low priority and system is busy, called procedures may not be scheduled for execution in a timely manner – negative performance impact

Db2 for z/OS: How Big is Big?

(http://robertsdb2blog.blogspot.com/2013/10/db2-for-zos-how-big-is-big.html)

- The main point: Db2 for z/OS systems and workloads are getting bigger all the time, and I have to keep adjusting my definitions regarding what constitutes “big”
- Case in point: buffer pool configurations
  - Largest buffer pool configuration I’ve seen (aggregate size of all pools allocated for a Db2 subsystem): **162 GB** (largest I’d seen was 46 GB when I posted the blog entry in 2013)
    - That 162 GB buffer pool configuration is for Db2 subsystem in LPAR that has 290 GB of memory
    - All pools in that configuration are defined with PGFIX(YES), and LPAR’s demand paging rate is 0
  - Largest single Db2 buffer pool I’ve seen: **52 GB**
    - I didn’t mention largest single pool in the 2013 blog entry, but note that this one pool is larger than the largest total buffer pool configuration size I’d seen back in 2013 (46 GB)
- What do I see as a smaller, medium-sized, or large buffer pool configuration?
  - In 2013 I wrote: small is < 5 GB, medium is between 5 and 20 GB, large is > 20 GB
  - Today I’d say: small is < 10 GB, medium is between 10 and 50 GB, large is > 50 GB

How big is big – DDF workloads

- In my 2013 blog entry, I mentioned that the largest DDF transaction rate I'd seen for a single Db2 subsystem (versus a data sharing group) was 786 per second
- I recently reviewed a Db2 monitor Accounting Report - Long (with data ordered by connection type) that showed a DDF transaction rate of **3057 per second** for a single subsystem
  - Easy to check DDF transaction rate at your site: in a Db2 monitor Accounting Report - Long, with data ordered by connection type, check number of commits in the sub-report for the DRDA connection type, and divide that by number of seconds in report interval

Some Db2 numbers are getting smaller

- For example, buffer pool read I/O rates
  - In the 2013 blog entry, I wrote that the highest total read I/O rate (all synchronous plus all prefetch read I/Os, per second) I'd seen for a single buffer pool was **9000 per second**
  - With more organizations configuring z/OS LPARs with LOTS of memory, and exploiting large real storage resources with ever-larger buffer pool configurations, read I/O rates are dropping
  - At some sites, aim is to get total read I/O rate below 100 per second for each and every buffer pool – success is achieved through buffer pool enlargement
- Also trending down: number of members in a Db2 data sharing group
  - Factors:
    - Ever-greater processing capacity of mainframe servers – don't need many, even for huge workload
    - The Db2 10-introduced change in management of thread-related virtual storage – resulted in 5-10X increase in number of threads that can be concurrently active on one Db2 subsystem
    - Also introduced with Db2 10: ability to remove a member from a data sharing group (prior to that, could permanently quiesce a member, but it would still show up in some message output)

Db2 for z/OS Work: the Task's the Thing

(http://robertsdb2blog.blogspot.com/2013/11/db2-for-zos-work-tasks-thing.html)

- The main point: type of task under which Db2 for z/OS work executes is important
- One reason that's important: zIIP eligibility
- Consider the DDF situation
  - A SQL statement that comes through DDF runs under a preemptable SRB in the DDF address space, and for that reason it is up to 60% zIIP-eligible (55-60% zIIP offload is good)
  - SQL that runs under a TCB is not zIIP-eligible, and an external stored procedure (one written in language such as COBOL) always runs under a TCB in a stored procedure address space
    - Therefore, an external stored procedure will NOT be zIIP-eligible, even if CALL comes through DDF (heavy use of external stored procedures is #1 reason DDF workload gets reduced zIIP offload)
  - A native SQL procedure (one written in SQL PL) always runs under the task of its caller, and if CALL comes through DDF, task is a preemptable SRB in the DDF address space, and for that reason a native SQL procedure is up to 60% eligible when called through DDF
    - "Nested" stored procedures: when native SQL procedure is called by external stored procedure, it will run under external stored procedure's TCB, and so will NOT be zIIP-eligible

The task's the thing: more on zIIP-eligibility

- When Db2 parallelizes a query, "pieces" of split query run under preemptable SRBs
  - Before Db2 12, execution of the pieces of a parallelized query is 80% zIIP-eligible
  - With Db2 12, that goes to 100% zIIP-eligible
    - That jump in zIIP eligibility for parallelized queries could lead to more organizations looking to expand exploitation of query parallelization, especially for batch and reporting queries
    - Starting with Db2 10, you can use the BIND QUERY command and the SYSQUERY and SYSQUERYOPTS tables in the Db2 catalog to 1) identify specific queries (static or dynamic) that are candidates for parallelization, and 2) set the limit on degree of parallelization for those queries
- More zIIP eligibility: prefetch reads and database writes
  - That activity has been 100% zIIP-eligible since Db2 10 for z/OS
  - Result: vast majority of Db2 database services address space (DBM1) CPU time shifted to zIIPs

(from Db2 monitor Statistics Report - Long)

CPU TIMES	(A) TCB TIME	(B) PREEMPT SRB	(C) NONPREEMPT SRB	(D) CP CPU TIME	PREEMPT IIP SRB
DB SERVICES ADDRESS SPACE	58.468010	1:26.687937	17.225778	2:42.381725	13:49.714110

A + B + C = D

Db2 10 (and beyond) for z/OS: Being Smart About More RELEASE(DEALLOCATE)

(http://robertsdb2blog.blogspot.com/2014/01/db2-10-and-beyond-for-zos-being-smart.html)

- The main point: the RELEASE(DEALLOCATE) option of BIND (and REBIND) PACKAGE can be a performance-booster, but don't over-do it
- First, how can RELEASE(DEALLOCATE) boost performance?
  - When the package is executed, associated table space locks and package sections allocated to a thread to remain allocated to the thread until the thread is deallocated
    - Almost all table space locks are intent-type (IS, IX) – not a problem if held for longer period of time
  - If the package is executed frequently, *and if the thread is persistent (i.e., if the thread persists through commits)*, CPU time that would otherwise be spent on repeatedly releasing and reacquiring the same table space locks and package sections is saved
  - Examples of persistent threads:
    - CICS-Db2 protected entry threads; Db2 threads for IMS wait-for-input (WFI), pseudo-WFI, and Fast Path Regions; high-performance DBATs (database access threads – i.e., DDF threads); batch threads

How Db2 10 changed the RELEASE(DEALLOCATE) picture

1. When packages bound or rebound in Db2 10 or later environment, two important changes related to virtual storage management (applies to plans, too):
   a) Virtual storage used for packages allocated to threads no longer comes from EDM pool – now comes from what is called agent local pool storage
      - Limited size of EDM pool formerly put a crimp on RELEASE(DEALLOCATE) usage – not an issue with agent local pool virtual storage, as there is tons of that
   b) Almost all virtual storage used for packages allocated to threads is above the 2 GB “bar” in the Db2 database services address space (DBM1)

   – Result: lots more virtual storage “headroom” for RELEASE(DEALLOCATE) usage
2. RELEASE(DEALLOCATE) honored for packages allocated to DBATs
   – When a package bound with RELEASE(DEALLOCATE) is allocated to a DBAT, thread becomes a high-performance DBAT (if it wasn’t already such)
   – High-performance DBAT can be reused 200 times, then will be terminated (to free up resources)

Db2 11 removed an operational challenge posed by RELEASE(DEALLOCATE) + persistent threads

- Pre-Db2 11 problem: a package cannot be bound (referring to BIND REPLACE), rebound, or invalidated while it is allocated to a thread
  - RELEASE(DEALLOCATE) + persistent thread = package continuously allocated to thread
  - Problem could be especially acute for CICS-Db2 environments – could be hard to find a "window" for BINDs/REBINDs, ALTERs, and pending DDL-materializing online REORGs
- Db2 11 (new-function mode) will automatically detect when a BIND, REBIND, ALTER or pending DDL-materializing online REORG is blocked by RELEASE(DEALLOCATE) package allocated to persistent thread
  - If thread "in Db2," package's behavior changed to RELEASE(COMMIT) at next commit
  - If thread not in-Db2 (awaiting re-use), Db2 will recycle thread to separate package from thread
    - Exception: doesn't apply to high-performance DBAT that's not in-Db2
    - Recommendation: issue the command -MODIFY DDF PKGREL(COMMIT) to temporarily "turn off" high-performance DBATs, if needed (turn back on with -MODIFY DDF PKGREL(BNDOPT))

Don't over-use RELEASE(DEALLOCATE)

- Transactional applications: best for packages, executed via persistent threads, that are executed frequently and have low average in-Db2 CPU time (like, < 10 ms)
  - As in-Db2 CPU time increases, time spent on releasing and reacquiring table space locks and package sections gets proportionately smaller
  - For simple transactions, RELEASE(DEALLOCATE) can reduce in-Db2 CPU by 10%, 20%, or more
  - Package-level info (accounting class 7 and 8) in Db2 monitor Accounting Report - Long can be useful in identifying good candidates for RELEASE(DEALLOCATE)
- DDF transactions: in addition to stored procedure packages, consider binding IBM Data Server Driver/Db2 Connect packages – in collection other than NULLID – with RELEASE(DEALLOCATE)
  - Then, point application to alternate collection via client-side data source property, or by automatic setting of CURRENT PACKAGE PATH special register via Db2 profile tables
  - Packages in NULLID collection should not be bound with RELEASE(DEALLOCATE)
    - That would make all DBATs high-performance DBATS – not an optimal situation

More on smart use of RELEASE(DEALLOCATE)

- Consider RELEASE(DEALLOCATE) for packages executed by batch jobs that issue lots of commits
  - Enhances effectiveness of sequential detection (which drives dynamic prefetch) and index look-aside (areas in virtual storage used with those two performance-enhancing features are wiped clean when batch job commits, unless package bound with RELEASE(DEALLOCATE))
- For transactional and batch applications, not a good idea to use RELEASE(DEALLOCATE) for packages that get exclusive table space locks
  - Could acquire such locks via SQL statement LOCK TABLE, or through lock escalation

A Tiger Changes His Stripes: I LIKE Java on z/OS

(http://robertsdb2blog.blogspot.com/2014/04/a-tiger-changes-his-stripes-i-like-java.html)

- The main point: Java + z/OS = great combination
  - Db2 angle: lots of Java applications access Db2 for z/OS data, many run in z/OS systems
  - Also, Db2 stored procedures can be written in Java (true since Db2 Version 5)
- My former thinking on Java in a z/OS system: inadvisable (maybe unfortunate)
  - Rationale: Java on z/OS consumes too many MIPS and too much memory, doesn't perform well
- Now I like z/OS as an environment for Java applications – here's why:
  - Huge performance improvement for Java on z/OS, especially over the past 10 years
    - Multiple factors: IBM Z hardware and z/OS enhancements, Java itself (example: Java gets better and better at JIT-ing code, referring to just-in-time compilation – more-efficient code, generated at runtime)
  - Java on z/OS is zIIP-eligible, reducing cost of computing
  - Java likes a lot of memory, but so does Db2 – and many z/OS systems have much more real storage that was typical a few years ago (partly because Z memory is a lot cheaper than before)

More on Java on z/OS

- How do people run Java programs on z/OS systems?
  - One option: use WebSphere Application Server for z/OS
  - Another option: the IBM JZOS Toolkit – a set of tools that helps you to develop Java applications that run as z/OS batch jobs (or started tasks)

    https://www.ibm.com/support/knowledgecenter/en/SSYKE2_8.0.0/com.ibm.java.zsecurity.80.doc/zsecurity-component/jzos.html
- For both WebSphere Application Server for z/OS and the JZOS toolkit, when target Db2 subsystem is on same z/OS LPAR, you can use type 2 or type 4 JDBC driver
  - Type 2: fewer instructions to get to Db2 and back when executing SQL statements
  - Type 4: more instructions to get to Db2 and back (go through LPAR's TCP/IP stack and into Db2 via DDF), but SQL statement execution is up to 60% zIIP-eligible
- Java stored procedures: starting with Db2 11, a single, multi-threaded, 64-bit JVM serves all Java stored procedures running in one stored procedure address space
  - Significantly greater scalability, and more efficient use of CPU and memory resources versus pre-Db2 11 situation (a single-threaded, 31-bit JVM for each Java stored procedure)

Db2 for z/OS: Avoiding zIIP Engine Contention Issues

(http://robertsdb2blog.blogspot.com/2014/09/db2-for-zos-avoiding-ziip-engine.html)

- The main point: if you don't have enough zIIP capacity for the zIIP-eligible work in your system, you could get more zIIP spill-over than you'd like
- "zIIP spill-over" = zIIP-eligible work executed on general-purpose engines
  - Happens when zIIP-eligible piece of work is ready for dispatch, and all zIIP engines are busy
- Negative impacts of zIIP spill-over:
  - Economic: whole point of zIIP engines is reduced cost of computing on IBM Z, and that benefit is reduced to the extent that zIIP spill-over increases general-purpose engine utilization
  - Performance: zIIP engine contention can affect elapsed times in two ways:
    - Prefetch slowdown: prefetch reads are 100% zIIP eligible – if prefetch read task is ready for dispatch and zIIP engines are busy, it gets re-directed to general-purpose engine after a delay of about 3 ms
    - Might see increased in-Db2 "not-accounted-for" time in Db2 monitor accounting report, especially for zIIP-using workload like DDF (in-Db2 not-accounted-for time generally reflects wait-for-dispatch time)

Calculating zIIP spill-over ratio

- Input data comes from Db2 monitor accounting report: specifically, from an Accounting Report - Long, *with data ordered by connection type*
  - In the report, find the "DRDA" sub-report (this shows all activity for applications accessing Db2 via DDF during the reporting interval – and I suggest a busy hour of the "online day")
    - In the DRDA sub-report, find these values (snippet is from an OMEGAMON for Db2 report – other monitors might have similar fields under the "average appl(class 1)" heading):

```
MEASURED/ELIG TIMES   APPL (CL1)
-------------------   ----------
CP CPU TIME             0.001935
 ELIGIBLE FOR SECP      0.000002   A

SE CPU TIME             0.000781   B
```

CPU time for zIIP-eligible work processed on general-purpose engines

CPU time for zIIP-eligible work processed on zIIP engines

zIIP spill-over ratio: A / (A+B)

zIIP spill-over

- My recommendation: keep zIIP spill-over below 5% (below 1% is great)
- How “hot” can you run zIIP engines without getting higher-than-desired zIIP spill-over?
  - Answer depends on number of zIIP engines available to z/OS LPAR – the more you have, the higher the level of utilization at which they can be run without causing contention issues
  - For example, if an LPAR has a single zIIP engine, zIIP contention could become an issue at a utilization rate not much higher than 30%
  - Other end of the spectrum: I’ve seen z/OS monitor data showing average zIIP engine utilization of 78% (very high), and corresponding Db2 monitor data showing zIIP spill-over of 1.7% (not bad)
    - The difference? The system with busy zIIPs and not-bad zIIP spill-over had 9 zIIP engines
  - **Note:** activating SMT2 for zIIPs can increase effective capacity by 25-40% (transactional workload)
    - SMT2 is short for simultaneous multithreading, with the “2” meaning that 2 processes can be dispatched simultaneously to one ”core” (one physical zIIP CPU)
    - SMT2 a feature of z13 and z14 servers, can be activated for zIIP and IFL engines (latter for Linux on Z)

I Don't Worry About Db2 for z/OS Buffer Pool Hit Ratios

(http://robertsdb2blog.blogspot.com/2015/09/i-dont-worry-about-db2-for-zos-buffer.html)

- The main point: the hit ratio, long seen as a key (often the key) metric with regard to Db2 buffer pool monitoring and tuning, is, in my opinion, of relatively little value
- Why I don't advise focusing on a buffer pool's hit ratio:
  - It can lead to a false sense that a buffer pool's activity, from a performance perspective, is as good as it reasonably can be
    - DBA: "The hit ratio for this pool is north of 99% – I'd say that my work here is done" ← Not necessarily!
  - It can lead to a discounting of prefetch read activity – something that should not be discounted
    - Some calculations of buffer pool hit ratio consider only synchronous read activity
  - Hit ratio values can be weird-looking – some calculations can yield a negative value
- My opinion: a much better metric on which to focus for buffer pool monitoring and tuning purposes is a pool's total read I/O rate

Calculating a buffer pool's total read I/O rate

- Simple: it's all read I/Os, per second:
  - All synchronous reads per second plus all prefetch reads per second (latter is sequential prefetch reads + list prefetch reads + dynamic prefetch reads, per second)
- Two sources of numbers: Db2 monitor Statistics Report - Long (or online display of buffer pool activity), or output of Db2 command -DISPLAY BUFFERPOOL
  - Db2 monitor:
    - Depending on monitor, per-minute numbers may be provided – divide by 60 to get per-second
  - Db2 command:
    - Issue command -DISPLAY BUFFERPOOL(ACTIVE) DETAIL, wait an hour, issue command again
    - Output of second issuance of command shows activity for the hour since first issuance of command (can verify with timestamp value in DSNB409I part of output)
    - Divide counters in output of second issuance of command by 3600 to get per-second figure
    - Total sync reads are sum of two counters: SYNC READ I/O (R) and SYNC READ I/O (S)
  - Monitor or command: make sure you count prefetch reads, not prefetch requests

Buffer pool total read I/O rate – what is objective?

- Short answer: fewer
  - Every read I/O you eliminate saves CPU time (application elapsed time should improve, too)
- At very least, aim to get total read I/O rate for each buffer pool below 1000/second
  - In these days of really big z/OS LPAR memory sizes, some Db2-using organizations try to get total read I/O rate for each buffer pool below 100/second
  - When enlarging Db2 buffer pools (and leveraging server memory in other ways), keep an eye on the z/OS LPAR's demand paging rate (available in a z/OS monitor CPU summary report)
    - Demand paging rate of zero is great, less than 1 per second is very good, 2-3 per second is not too bad – beyond that is more than I'd like to see
- Note that for "pinning" pool (one used to cache objects in memory in their entirety – should be defined with PGSTEAL(NONE)), desired total read I/O rate is zero
  - If total read I/O rate for pinning pool > 0, is # of buffers in pool < # of pages belonging to objects?
  - In Db2 data sharing environment, could see a few read I/Os for pinning pool, even if it has "enough" buffers – has to do with page invalidation caused by data-change activity

Db2 for z/OS: Using PGFIX(YES) Buffer Pools? Don't Forget About Large Page Frames

(http://robertsdb2blog.blogspot.com/2016/09/db2-for-zos-using-pgfixyes-buffer-pools.html)

- The main point: for a buffer pool with a relatively low read I/O rate (e.g., less than 100 per second), a PGFIX(YES) specification does little to boost performance if the pool is not backed by large real storage page frames
- Background: when PGFIX(YES) is specified for a buffer pool, that pool's buffers cannot be paged out to auxiliary storage by z/OS
  - Why that's good for performance for pool with a lot of I/O activity: when PGFIX(NO) is in effect for a pool, a page frame occupied by a buffer belonging to the pool has to be fixed in memory, and subsequently un-fixed, every time a page is read into, or written out from, the buffer
    - Ensures that buffer won't be stolen out from under Db2 by z/OS in middle of a read or write operation
    - Applies to sync and async I/Os, group buffer pool reads/writes (data sharing) as well as disk reads/writes
    - With PGFIX(YES) in effect for pool, no need for these page-fix and page un-fix operations, as pages holding buffers are fixed in memory from the get go – that means cheaper I/Os with PGFIX(YES)
    - Cheaper I/Os not very important if pool has low I/O rate

PGFIX(YES) and large real storage page frames

- Can PGFIX(YES) deliver a performance boost for low-I/O buffer pools?
  - YES – by enabling use of large real storage page frames for a pool
- Large frames (1 MB or 2 GB) can deliver CPU savings when used for high-activity buffer pools (e.g., > 1000 GETPAGEs per second), even if they have low I/O rates
  - How: by making virtual storage to real storage address translation more CPU-efficient (due to fewer "misses" in the translation lookaside buffer, aka the TLB)
  - The more active a buffer pool, the greater the CPU efficiency benefit of large page frames
- Check output of Db2 command -DISPLAY BUFFERPOOL:

```
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
```

Large frames can only be used for a PGFIX(YES) pool

If large frames preferred but not used for a pool, either the system has no large frames, or there are not enough large frames to back all PGFIX(YES) pools

Making large frames available in your system

- LFAREA, a parameter in the IEASYSnn member of SYS1.PARMLIB, specifies amount of LPAR's real storage that will be managed in 1 MB and/or 2 GB frames
  - To have Db2 buffer pool backed by 2 GB page frames, must specify FRAMESIZE(2G) for pool
  - To have Db2 buffer pool backed by 1 MB frames, must specify FRAMESIZE(1M) for pool, unless the pool was in use in a Db2 10 environment and had a PGFIX(YES) specification
- How big should LFAREA values be?
  - LFAREA 2 GB specification: large enough to fully back all buffer pools defined with FRAMESIZE(2G)
  - LFAREA 1 MB specification: add up the size of all buffer pools defined with FRAMESIZE(1M), then bump that up by 5% (to provide for other uses of 1 MB frames)
  - **Don't overdo it –** plenty of processes in a z/OS system can ONLY use 4 KB frames
  - **And**, if you run WebSphere Application Server under z/OS, keep in mind that 1 MB or 2 GB frames can be used for the Java heap – consider that when determining LFAREA values

A little more on large frames for Db2 buffer pools

- If you want to back a 4K buffer pool with 2 GB frames, consider making VPSIZE (number of buffers allocated for pool) a multiple of 524,288 (number of 4K buffers that would exactly fill a 2 GB page frame)
  - If VPSIZE X buffer size is a little larger than, say, four 2 GB frames, then pool will be backed by four 2 GB frames, then as many 1 MB frames as can be filled, then 4 KB frames for the rest
- 2 GB frames may not provide much CPU efficiency benefit vs. 1 MB frames for a pool that is < 20 GB in size, but won't hurt performance if pool is smaller than that
- Some people ask, "can I go with PGFIX(YES) and large frames for all of my buffer pools – even the ones with relatively low I/O and relatively low GETPAGE rates?
  - Sure, if the z/OS LPAR has enough memory
  - Remember: want demand paging rate of zero or a very small non-zero value per second – that indicates sufficient memory for all processes in the system (Db2 and other stuff)

Thanks for your time.

Robert Catterall
rfcatter@us.ibm.com