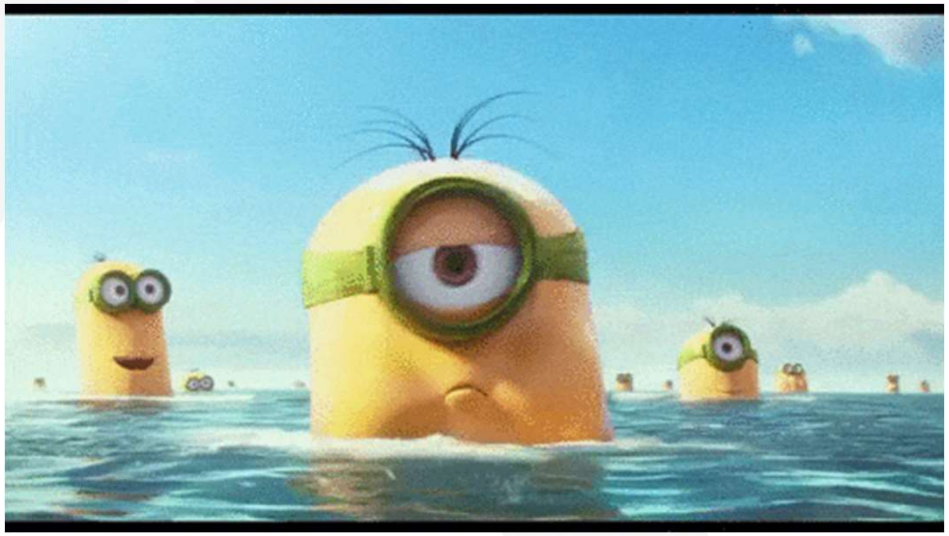
SHARE
EDUCATE ▸ NETWORK ▸ INFLUENCE

Don’t just Play in the Pool, Own it…Buffer pool that is

Adrian Burke
DB2 SWAT team SVL
agburke@us.ibm.com

Agenda

- Starting point
- Separation
- Thresholds
- Tuning
- Buffer pools and Storage



Methodology - disclaimer

- ‘It’s all relative’ as they say
  - Is 1,000 I/Os per second a lot for a buffer pool?
    - Customer #1 only has 100,000 getpages per second, but buffer pool size was restricted by real storage constraint
    - Customer #2 has 1,000,000 getpages per second
      - Customer #2’s BP is 10x larger
    - So are getpages per sync I/O, page residency, or hit ratio a more relevant metric?
  - The key is to start with *your* environment and understand the behavior of your pools relative to each other
    - Compare before/after and iterate around again
  - Know what knobs to tune, and when the opportunity for real storage exploitation arises – be ready to act!

zParms and Buffer Pool pollution

- Zparms
  - General recommendation to separate CAT/DIR from user objects
  - Db2 defaults for user objects create ***pollution***

	VPSIZE
• DEFAULT 4-KB BUFFER POOL FOR USER DATA ===> BP0	*(20000)*
• DEFAULT 8-KB BUFFER POOL FOR USER DATA ===> BP8K0	*(2000)*
• DEFAULT 16-KB BUFFER POOL FOR USER DATA ===> BP16K0	*(500)*
• DEFAULT 32-KB BUFFER POOL FOR USER DATA ===> BP32K	*(**250**)*

    - DEFAULT BUFFER POOL FOR USER LOB DATA ===> BP0
    - DEFAULT BUFFER POOL FOR USER XML DATA ===> BP16K0
    - DEFAULT BUFFER POOL FOR USER INDEXES ===> BP0
  - Pollution of the CAT/DIR buffer pools are guaranteed
  - Remember what important objects can be found there
    - Customer data pollutes CAT/DIR buffer pools and Db2 objects can negatively affect user data performance due to LOBS

- 8K = 10
  - SYSPLUXA DSNDB06
  - SYSXML DSNDB06
- 16K = 6
  - SYSCONTX DSNDB06
  - SYSROLES DSNDB06
  - SYSSTATS DSNDB06
  - SYSTSROU DSNDB06
- 32K = 6
  - SPT01 DSNDB01
  - SYSUTILX DSNDB01
  - SYSTSCTL DSNDB06

zParms and Buffer Pool pollution

- How can you tell?
  - Zparms as well as looking in the catalog
    - Query SYSTABLESPACE to understand where your user objects reside (see Appendix)
    - Often customers have more non-catalog objects in the pool than they do CAT/DIR objects and the number of active pages or leaf pages tells this story
    - Often user and vendor objects as well as auxiliary tables/indexes end up there
  - BP statistics, like seeing BP32K having the highest Sync I/O rate in the subsystem
  - So What? – Disruption!
    - Migration job elapsed times, rebinds, DDL
    - Object RECOVER times
    - Space growth of objects
    - CPU and elapsed of apps

BPOOL	PAGESET	TYPE	COUNT	PAGES_ACTIVE_LEAF
BP0	IX		1339	1034440
BP0	IX	C	253	73750
BP0	IX	D	11	11
BP0	TS		103	40180
BP0	TS	C	88	56874
BP0	TS	D	6	6
BP8K0	TS		51	11276
BP8K0	TS	C	10	64478
BP16K0	TS		22	148
BP16K0	TS	C	6	4193
BP32K	TS		35	1948171
BP32K	TS	C	4	5360
BP32K	TS	D	2	2

Typical customer example

TBSBP16K	inst	BP16K0
TBSBP32K	inst	BP32K
TBSBP8K	inst	BP8K0
TBSBPLOB	inst	BP0
TBSBPOOL	inst	BP2
TBSBPXML	inst	BP16K0

LOB pollution

- LOB objects (even SLOBs) are usually infrequently re-referenced
  - If the data was that useful put it as an inline LOB, and then it can even be compressed
  - Often simply used to document changes to base row
  - Max of 1 LOB row per page in AUX tablespace
    - So sync I/O for each row, or if it spans pages then prefetched at up to 128K at a time
- So what?
  - Starting in V9 we lost some granularity of the Read LSN, or longest running reader in the system as it went up to the buffer pool level (i.e. GBP level) for the entire data sharing group
  - Space from deleted LOB objects in the AUX table space cannot be reused if there 'might' be an application reading LOB rows
  - Thus without LOB locks space reuse is hindered when there are long running readers/UOWs running in the same buffer pool as the LOB objects
    - Watch for space growth in AUX tablespaces, to include CAT/DIR objects
      - (SPT01/ DBDXA) – PM80804 addressed some of this

Segregate LOB objects

- Mixing LOB and non-lob table spaces can also increase XES messages in data sharing environments as the existence of AUX table spaces, AUX indexes, and static sensitive scrollable cursors cause updates to the SCA structure to track the Read LSN (see PI78061)
  - READ LSN is stored in the SCA
- In data sharing environments affects claims against the READ LSN (read log sequence number)
  - 1 per buffer pool
  - Inserters must get the lock to check for space reuse
    - MBA lock mentioned in PI78061, where we removed it from application's commit scope
      - Affected L-LOCK OTHER time increasing elapsed and CL2 CPU time
- V12 will help greatly with a much more granular RLSN/CLSN

Query to verify which BPs contain LOB objects

```
Select substr(c.name,1,25),b.name, b.bpool from sysibm.sysauxrels a, sysibm.systablespace b, sysibm.systables c
where a.auxtbowner='SYSIBM' and a.auxtbname=c.name and c.tsname=b.name group by c.name,b.name,b.bpool;
```

zParms and Buffer Pool pollution

- How can we un-pollute?
- V10 brought ALTER X.X BUFFERPOOL(BPx) - immediate
  - Prior to DB2 10 moving objects required a STOP and START around the ALTER
  - DB2 10 allows the bufferpool change to now be "immediate"
    - Page **size** cannot be changed in this manner
  - DB2 10 performs
    - DRAIN ALL operation on the object, which drives physical close on all DB2 members
    - Physical close by the last DB2 member drives all of the GBP castout and purge processing
    - X mode lock taken on skeleton cursor and package tables
  - Should only be submitted during quiet period
- Then enforce separation standards in ZParms and run the Catalog query regularly

Parameters

- VPSIZE – buffer pool size in buffers (for space remember VPSIZE x PGSIZE)
  - SPSIZE – simulated net increase in VPSIZE
- VPSEQT (default 80%) – sequential steal threshold: when x% of the buffers in this pool are sequential, we steal those buffers prior to random buffers (became much more effective in V11, see later slide)
  - SPSEQT – simulated VPSEQT for net increase in VPSIZE
- DWQT (default 30%) – deferred write threshold (all data sets in BP); number/pct of dirty pages in pool
- VDWQT (default 5%) – vertical deferred write threshold (per data set): 0% means 40 pages queued, 64 if GBP dep
  - Historically this threshold would have to be very low for NPIs to avoid flooding the GBP with updated pages during Utility execution ***BUT*** GBP write-around in V11 eliminates this need
- PGSTEAL (default LRU) – steals buffers based on frequency of use *OR…*
  - NONE – 'in memory' buffer pool
  - FIFO – steals buffers in the relative order of when the pages were brought in
  - MRU – used by Db2 internally for Utility executions to avoid flushing out useful pages
    - PI87228 stops REORG from throwing out 'new' pages in shadow data set
- **For Workfile BPs – *Conservatively* set VPSEQT=90-95, DWQT= 60, VDWQT=50, PGSTEAL=LRU: this is to protect against rogue DGTTs thrashing it

Deferred write thresholds

- Writing pages out due to hitting VDWQT is more efficient than DWQT because Db2 only schedules 128 pages out at a time in up to 4 I/Os
  - These pages must be in the range of 180 buffers to avoid excessive buffer pool latch contention
  - Much less likely that DWQT will find 32 pages from different data sets in a range of 180 buffers
  - VDWQT writes out pages until VDWQT met
  - DWQT will try to write out enough page to bring the % of 'dirty' pages down to DWQT – 10%, so the there may be many more less efficient deferred write operations
- Ideally VDWQT should be used over DWQT for efficiency of writes and avoiding latch contention
  - Pages written per write I/O

Deferred write thresholds…

- Starting in DB2 10 the root pages of the indexes are ‘fixed’ in the buffer pool
  - How many indexes/parts do you have in your index buffer pool?
- This would affect DWQT threshold as the pages are ‘dirty’ but cannot be stolen
  - (ex.) 10,000 buffers, DWQT of 30%
    - With 1,000 indexes you have basically made the DWQT threshold 20%
    - Watch for DWQT being hit multiple times per second and elevated LC23
      - Customer saw DWQT threshold being hit 80 times a second and LC23 at ***40,000*** a second – anything over ~10,000 should be investigated
      - Application response times were significantly impacted due to being I/O bound, elapsed times increased 2-3x
    - LC14 contention can also be elevated because although these root pages are on the LRU chain, then cannot be stolen, so the chain can be elongated by may indexes and frequently bumping into DWQT
  - **ANSWER:** Customer doubled VPSIZE, ensured headroom to avoid being within DWQT threshold

BP Thresholds to be aware of

- Immediate Write Threshold - SYNCHRONOUS WRITES > 'normal' — **97.5% dirty pages**
  - Checked at page update – SMF 100 reports sync writes, but not **this** threshold
  - After update, synchronous write
    - Pages also synchronously written if they have been in the BP for more than 2 system checkpoints
- Data Management Threshold – DM THRESHOLD — **95% dirty pages**
  - Checked at page read or update – reported in SMF 100
  - Getpage can be issued for each row sequentially scanned on the same page – potential large CPU time increase
- Prefetch Threshold - PREF DISABLED – NOBUFFER — **90% dirty pages**
  - Checked before and during prefetch – reported in SMF 100
  - 90% of buffers not available for steal, or running out of sequential buffers
    - OR VPSEQT=0 and INDEX_IO_PARALLELISM = YES
  - Disables Prefetch, hence many more Sync I/Os

TUNING

Residency time

- Generally we try to protect random pages because the application intended to touch them
- Random hit ratio/Sequential hit ratio although popular with vendor tools will vary significantly between DB2 10 and DB2 11 due to classification of getpages
  - Inherent discrepancy between getpage/buffer classification so hit ratio was not accurate and VPSEQT was not a valid tuning option
  - Prior to Db2 11 getpages that resulted in dynamic/list prefetch were considered random
    - Random hit ratio could be inflated as the getpage was random but the I/O was asychronous
    - Random Hit Ratio = (total random getpages – synchronous pages read) / random getpages
- Sequential hit ratio could be (-) negative when prefetch brings in more pages than are consumed by the application, which is a reflection of access path, not buffer pool tuning
- Residency Time = average time that a page is resident in the buffer pool
  - System RT (seconds) = VPSIZE / Total pages read per second
  - Total pages read = synchronous reads for random getpages + synchronous reads for sequential getpages + pages read via sequential prefetch + pages read via list prefetch + pages read via dynamic prefetch

Residency time…

- General Rule of Thumb of 60 seconds (arbitrary)
- The longer the page is resident in the BP the higher the likelihood of the page being re-referenced and avoiding an I/O in the future
  - Behavioural differences in DB2 10 and DB2 11 will cause this number to vary
- If the BP shows a ~30 second residency time, and we double the size to get to 60 seconds we only have linear improvement
- Ideally we want the absolute number of I/Os to decrease more than the # of buffers added, thus show savings from re-reference of pages
  - If we aim for 60, but doubling the pool ends up with an average residency of >60 seconds, we have achieved incremental performance gains
  - Next step would be to test VPSEQT changes
- There is no magic number for page residency, but it is useful in determining the ratio of size increase, to I/O decrease

TOTAL # OF INTERVALS	1440	
	BP10	BP11
LESS THAN 300 SEC	1433	0
LESS THAN 60 SEC	1286	0
LESS THAN 30 SEC	1104	0
LESS THAN 10 SEC	640	0

Buffer Pool Simulation (V11 CM)

- Ability to simulate increasing VPSIZE and increasing/decreasing VPSEQT
  - Assumes LRU mechanism in place
- Storage cost for a simulated buffer pool is ~2% for 4K pages similar to Hiperpool
  - SPSIZE * 4K (e.g. 20MB for 1GB size buffer pool)
  - Simulation is done with a separate buffer search hash table with buffer control blocks
  - CPU overhead is between 1-3% when running lab workloads
- Results seen in statistics in DSNB431I - 432I, and OMPE (IFCID 002)
- Look for law of diminishing returns
  - Track absolute number of I/Os saved per GB of buffers added to the pool
    - Use *-DIS BP(x) DETAIL over specific intervals*
  - **Remember to increase GBP as well

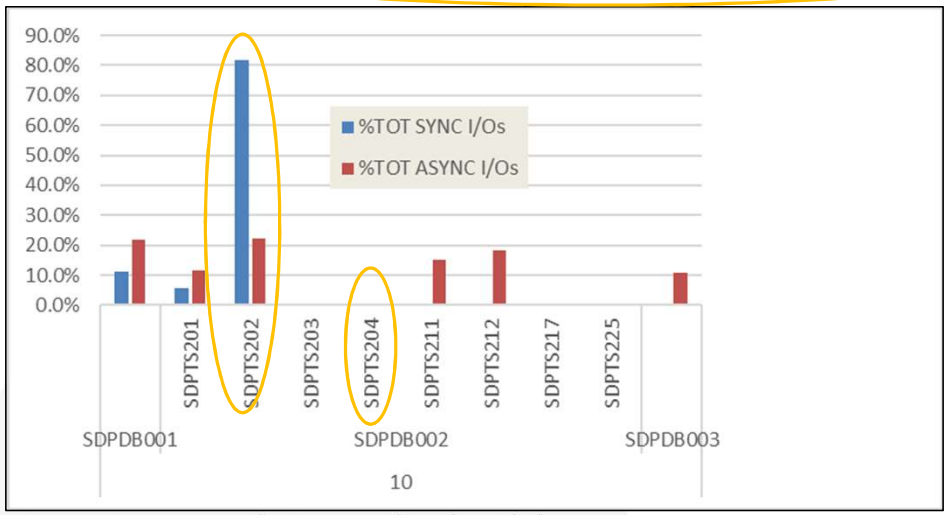
Cannot account for cross-invalidated pages

```
DSNB432I  -CEA1 SIMULATED BUFFER POOL ACTIVITY -
          AVOIDABLE READ I/O -
           SYNC  READ I/O (R)  =365071
           SYNC  READ I/O (S)  =5983
           ASYNC READ I/O      =21911
           SYNC  GBP READS (R) =89742
           SYNC  GBP READS (S) =184
           ASYNC GBP READS     =279
      PAGES MOVED INTO SIMULATED BUFFER POOL
=13610872
           TOTAL AVOIDABLE SYNC I/O DELAY =158014 MS
```

Object level statistics

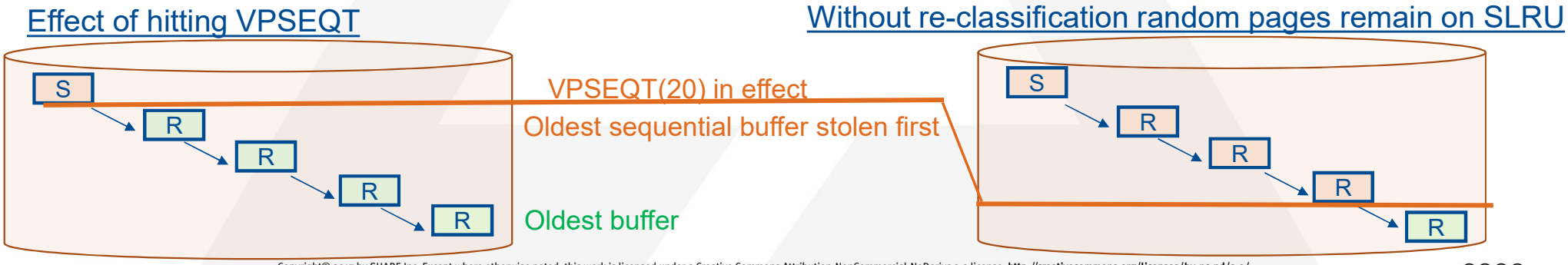
- IFCID 199 (Stats Class 8) can be used to delve into what objects are impacting the buffer pool stats
  - Negligible overhead and ~2K per record of 50 objects w/ >1 I/O per second
- Use OMPE CSV generator to pull these stats for several intervals
  - Does one table or partition monopolize the I/O?
    - Do the getpages match with the % of I/O?
  - What objects are candidates for PGSTEAL (NONE)?
  - Is it sync or async?
    - Goal is to protect random pages

Row Labels	SUM GETPAGES	%TOT GETPAGES
10	103,366,499	100.00%
SAMNR001	7,645	0.01%
SAMPB001	136,144	0.13%
SDPDB001	24,836,021	24.03%
SDPDB002	73,787,051	71.38%
SDPTS201	9,962,147	9.64%
SDPTS202	21,652,946	20.95%
SDPTS203	92,935	0.09%
SDPTS204	3,495,321	3.38%



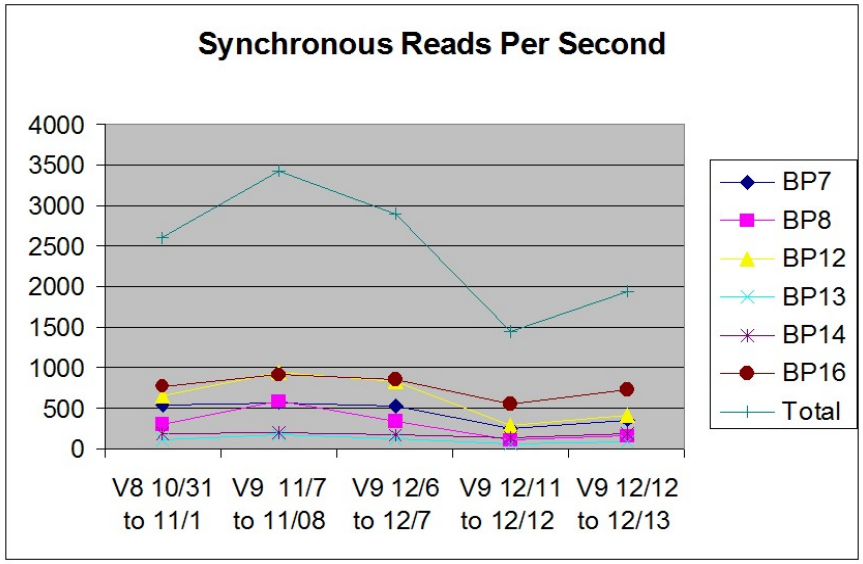
VPSEQT - impact

- **Customer** monitors DSNB411I for sync I/Os tracked it across V8 → V9 release
  - 30% increase in sync I/O and large increase dynamic prefetch reads
  - The customer's BP tuning strategy was to protect random pages by keeping VPSEQT quite low
- In V8 and V11 Db2 would reclassify sequential buffers which were later re-ferenced randomly, hence they would move from the sequential (SLRU) chain to the random (LRU) chain
  - This behavior was removed in V9 and V10
  - Even though these pages were re-referenced by applications, they remained subject to the VPSEQT steal threshold and were stolen prior to the truly oldest buffer in the pool
  - Hence applications which read in many pages then update a subset might have them stolen and need to retrieve them from disc again



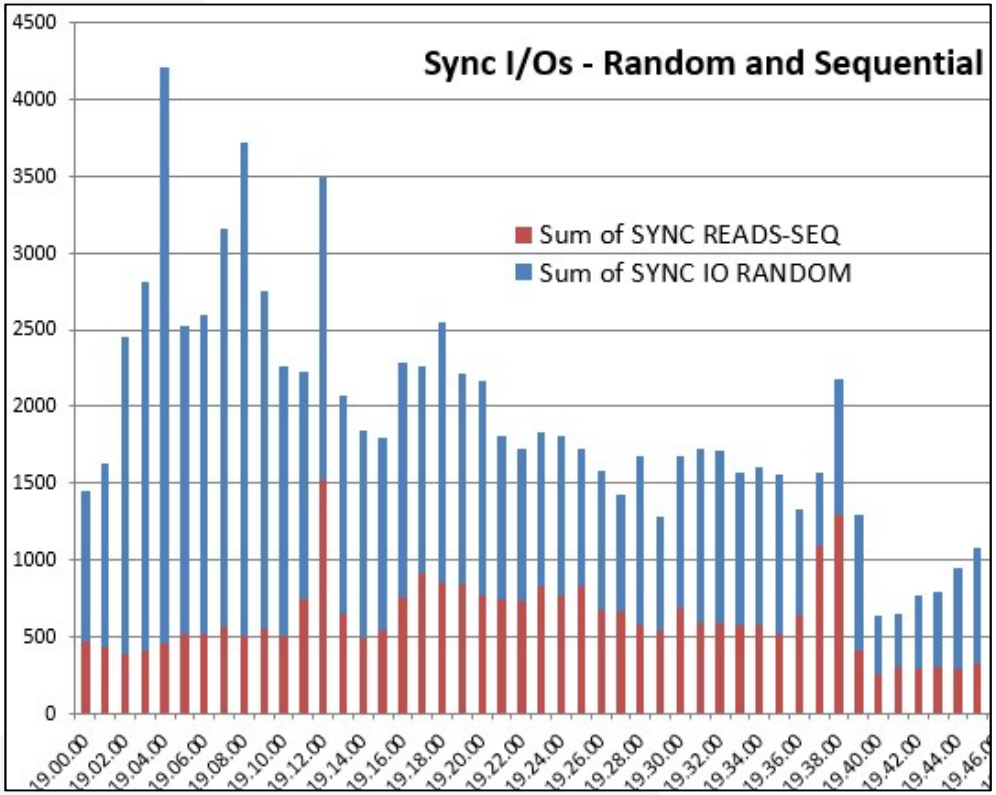
VPSEQT - impact…

- **Customer's** steps to identify the root cause and tuning:
  - V8 10/31: Baseline (month end so slightly elevated)
  - V9 11/7: Move to V9 and identify sync I/O increase
  - V9 12/6: Tune Sync I/O Sequential (VPSEQT=10 → VPSQET=80 [default])
    - Needed to increase the amount of sequential pages in the buffer pool to avoid stealing pages before they were re-referenced due to dynamic prefetch increase
  - V9 12/11: Apply PM55357 (similar to V11 CM)
    - Reverted to V8 code to reclassify sequential buffers to random when synchronously touched by another process
  - V9 12/12: Tune VPSEQT back to 10%
    - Protect random pages from being stolen by prefetch operations (especially sequential prefetch from table space scans)
    - Noticeable increase in sync reads due to heavier workloads and sync read sequential due to VPSEQT



Sync Read sequential I/O

- Sync I/O for sequentially accessed pages is the equivalent of ***1 I/O for the price of 2***
  - 1) The pages were brought into the buffer pool via prefetch, but before the application could consume the page that buffer was stolen by another application process
  - 2) Prefetch was disabled for NO READ ENGINES / NO BUFFERS
- Hence the application needed another sync I/O to finally consume that page
- Solution is to increase VPSIZE *or* VPSEQT, and monitor
- **ROT**: VPSIZE x **VPSEQ**T >= 320MB (450MB in V12)
  - Ensures maximum prefetch concurrency
  - Avoid PREF.QUANT.REDUCED in stats



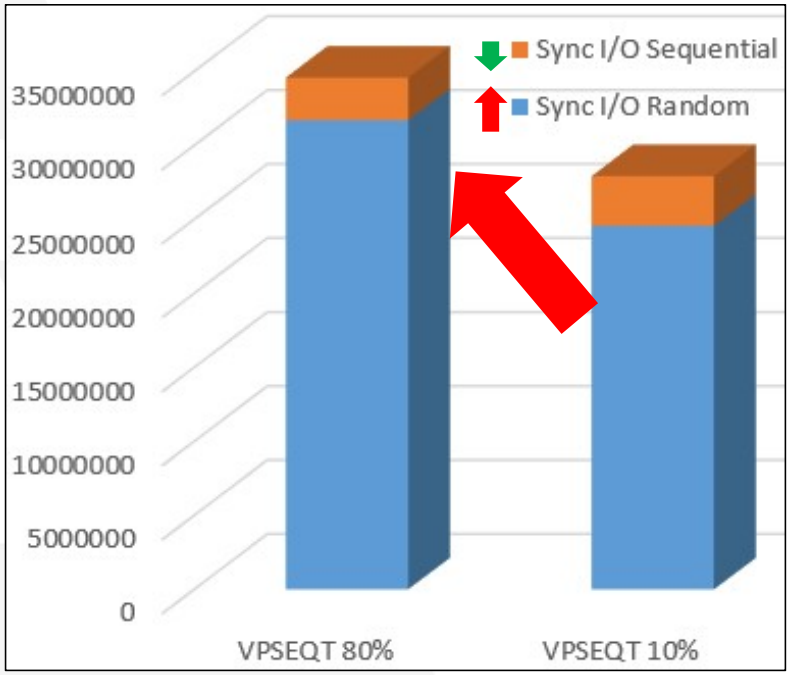
VPSEQT - tuning

- Customer monitors DSNB411I for sync I/Os and was
- Customer moved VPSEQT from **10%** to **80%** in *hopes* of:
  - **Tuning out sync sequential I/Os**
  - Aligning with IBM defaults
  - Lowering how frequently VPSEQT was hit
  - Influencing the reclassification of buffers
    - How often pages were prefetched into the pool, then accessed randomly by another application

```
DSNB411I
SYNC READ I/O (R) =24572028
SYNC READ I/O (S) =3357642
VPSEQT HIT        =2876053
RECLASSIFY        =181788332
```

- ***BUT.. this resulted in more I/Os → … Why?***

Net result was increase of >6M Sync I/Os



VPSEQT - tuning…

- Changing VPSEQT increased number of potential sequential buffers from 470k → 3.7M
  - This resulted in 12M more re-classifications of pages (from sequential to random)
- VPSEQT was too high to preserve the random buffers, hence the re-referenced random buffers were being stolen for sequential pages, were prefetched back in again, along with many unneeded sequential pages
- **Summary:** Use SPSEQT (BP simulation in V11 CM) to estimate effect of changing VPSEQT *or* at least measure before and after of both types of sync I/O, and watch for the tradeoff between eliminating sync sequential I/O and adding sync random I/O

DSNB411I	*Before*
SYNC READ I/O (R)	=24572028
SYNC READ I/O (S)	=3357642
VPSEQT HIT	=2876053
RECLASSIFY	=181788332

DSNB411I	*After*
SYNC READ I/O (R)	=31700555
SYNC READ I/O (S)	=2883360
VPSEQT HIT	=3522324
RECLASSIFY	=193560229

Asynchronous I/O (V10+)

- Index I/O Parallelism for updates
  - If there are more than 2 indexes on a table (clustering index does not count) or 2 if the table is defined with APPEND, HASH, or MEMBER CLUSTER
    - DB2 detects an I/O delay we use sequential prefetch engine to do the I/O for each index leaf page in parallel
  - You will see S.PRF.PAGES READ/S.PRF.READ = 1.00 in the statistics report for index buffer pools
    - Use IFCID 357-358 to trace it
    - zParm INDEX_IO_PARALLELISM =YES (default)
  - If VPPSEQT or VPSEQT = 0**
    - Disabled at BP level
    - PREF.DISABLED-NO BUFFER Will be non-0

SEQUENTIAL PREFETCH REQUEST	22308.00
SEQUENTIAL PREFETCH READS	0.00
PREF.DISABLED-NO BUFFER	22308.00

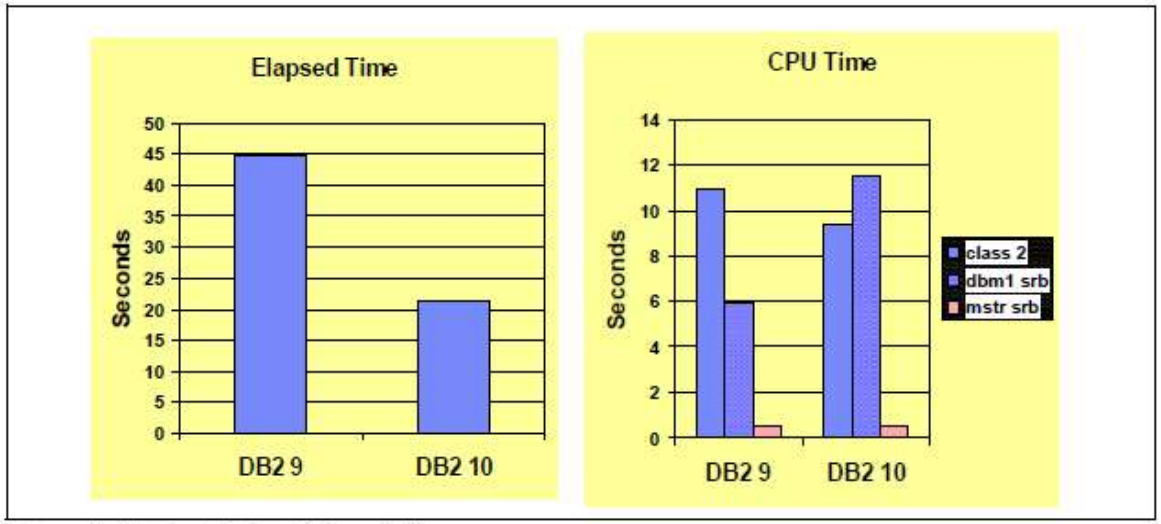


Figure 2-14 Insert index I/O parallelism

Asynchronous I/O (V12)…

- More prefetch engines are available for use
  - Moved from 600 to 900 engines per DB2 subsystem
    - Hidden ZPARM SPRMRDU controls the number
    - Still uses ESQA and some below the bar storage so don't go crazy
- Remove unnecessary prefetch scheduling in V12
  - Tracks Dyanamic Prefetch failures
    - If last 3 prefetch requests did not result in prefetch I/O
      - Disable dynamic prefetch on that thread/object
    - First Synchronous sequential I/O detected
      - Dynamic prefetch is re-enabled
- Saves
  - zIIP cycles
  - Unnecessary other READ I/O class 3 delays
  - Prefetch disabled NO READ ENGINE
  - LC24 contention caused by multiple prefetch requests against the same page
    - ~30k requests/sec = ~ 10k latch/second

Customer has a 70:1 ratio of requests vs. scheduled prefetch

Dsnb414i	Dsnb414i
Dynamic Prefetch Requests	Dynamic Prefetch I/O
36,973,390	550,642

Ex. LC24 contention at 250k per second caused DBM1 zIIP spike

DSAS - SRB TIME	DSAS - PREEMPT SRB	DSAS - PREEMPT IIP SRB
0.010091	0.008489	0.133704
1.400117	1.398301	8.189153

PCLOSEN and PCLOSET

SHARE
EDUCATE ▸ NETWORK ▸ INFLUENCE

- PCLOSEN and PCLOSET are Zparms governing the frequency with which GBP dependent objects are closed off, i.e. pseudo-closed due to a lack of intersystem read/write interest
  - The default starting in DB2 10 → PCLOSEN=5 (number of system checkpoints) / PCLOSET=10 (time in minutes)
- Page sets affected are those defined with CLOSE=YES, mainly in a data sharing environment
  - Once a PCLOSE* interval has passed the data set will be pseudo closed, removing it from group buffer pool dependency
    - After the second interval the object will be physically closed on that member
- CLOSE=YES (allow more pseudo close) vs. CLOSE=NO (less pseudo closed objects) DDL
  - CLOSE NO allows you to keep objects opened in the local buffer pool and avoid CPU and elapsed time overhead of physical opening/closing (much diminished after V11), less SYSLGRNX updates from objects opening and closing
  - CLOSE YES means less likelihood of infrequent data set reads keeping an object GBP dependent *forever* (-ACCESS DB MODE NGBPDEP)
    - Shorter recovery during subsystem crash, less objects put in LPL/GRECP status

PCLOSEN and PCLOSET

- DB2 10 added a mechanism to avoid local buffer pool scans when objects go in and out of GBP dependency by simply invalidating the pages of those objects
  - This saves DBM1 SRB time, and application elapsed time of updaters
  - But depending on the amount of pseudo closes you have it can increase cross-invalidation and synch I/O for some applications that bounce in and out of GBP dependency
    - V11 APAR PI59168 addresses some XI conditions
  - *Increasing size of local BP/GBP will not help I/Os from cross-invalidation*

DB2 9

GROUP BP7	AVERAGE	TOTAL	GROUP BP8	AVERAGE	TOTAL	GROUP BP12	AVERAGE
GBP-DEPEND GETPAGES	343.4K	16481954	GBP-DEPEND GETPAGES	3300.33	158416	GBP-DEPEND GETPAGES	102.9K
READ(XI)-DATA RETUR	30.67	1472				READ(XI)-DATA RETUR	36.23
READ(XI)-NO DATA RT	2.50	120				READ(XI)-NO DATA RT	0.04
READ(NF)-DATA RETUR	190.04	9122	READ(NF)-DATA RETUR	0.00	0	READ(NF)-DATA RETUR	10.52
READ(NF)-NO DATA RT	12379.02	594193	READ(NF)-NO DATA RT	1.79	86	READ(NF)-NO DATA RT	1227.54

DB2 10

GROUP BP7	AVERAGE	TOTAL	GROUP BP8	AVERAGE	TOTAL	GROUP BP12	AVERAGE
GBP-DEPEND GETPAGES	320.4K	15380145		3306.02	158689	GBP-DEPEND GETPAGES	91613.63
READ(XI)-DATA RETUR	54.67	2624				READ(XI)-DATA RETUR	24.73
READ(XI)-NO DATA RT	16597.00	7968				READ(XI)-NO DATA RT	24369.08
READ(NF)-DATA RETUR	140.60	6			0	READ(NF)-DATA RETUR	0.29
READ(NF)-NO DATA RT	16620.69	7977			71	READ(NF)-NO DATA RT	3086.73

XI No Data RT means the page was cross invalidated in the local pool, but was not found in the GBP

PCLOSEN/PCLOSET and Synch I/O

- The customer saw a 20% increase in Synch I/O after migration
- They had moved from PCLOSET=30 → PCLOSET=10 so every 10 minutes objects without inter R/W interest would pseudo close (compare OPEN/CLOSE stats with READ(XI) stats)
- When the objects moved out of GBP dependency the local buffers would be cross invalidated
  - Next execution of the application would require entire index be read back in

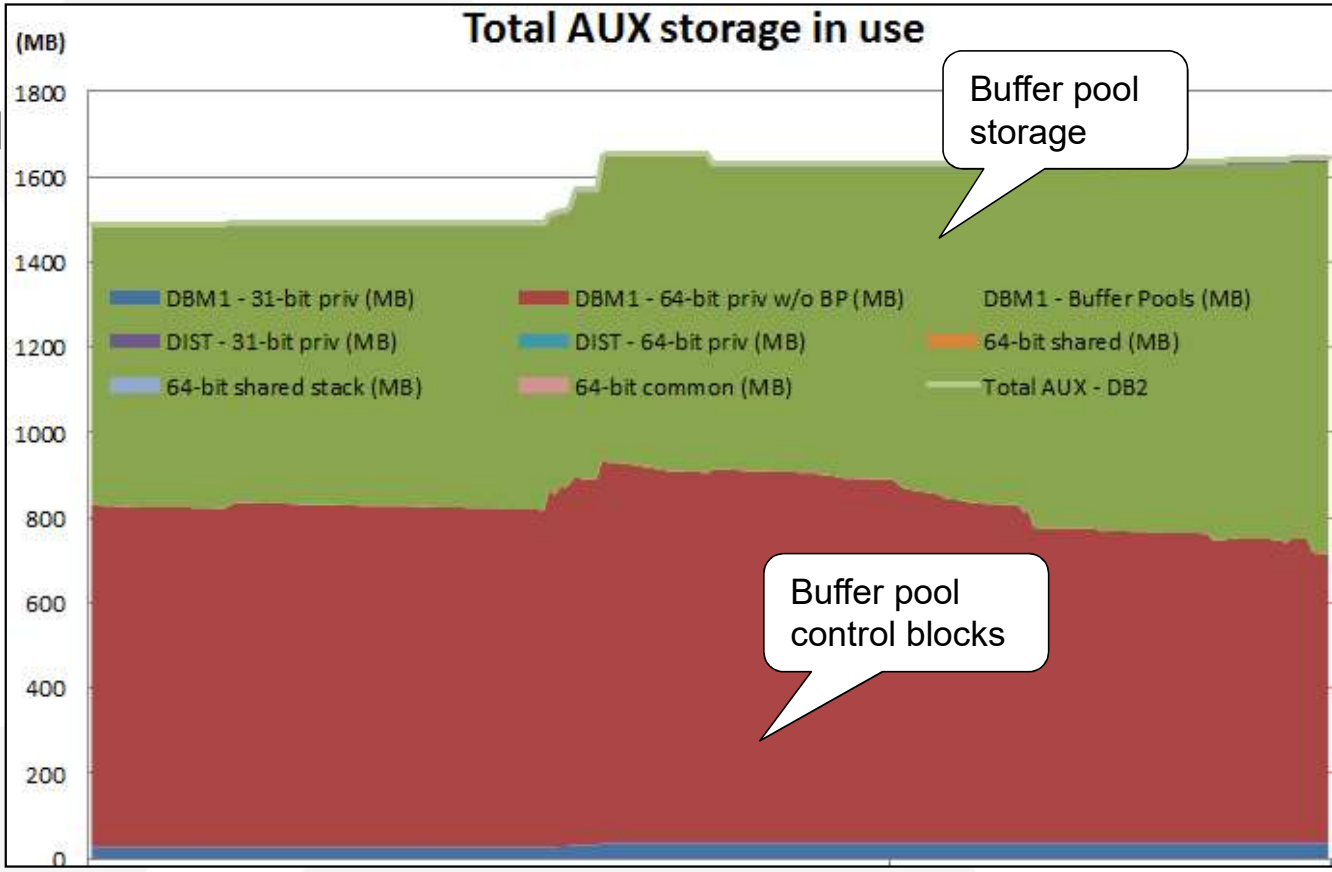
OPEN/CLOSE ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT	
DSETS CONVERTED R/W -> R/O	9010.00	0.67	0.03	0.00	<==V9
DSETS CONVERTED R/W -> R/O	24721.00	1.72	0.07	0.00	<==V10

  - **ROT: R/W → R/O = 10-15 a minute
  - The solution in ***this*** situation was to set PCLOSEN=32767 to disable it, and PCLOSET=45 minutes so that the objects did not through pseudo close until this key application ran again (every 30 minutes)
- Disabling PCLOSEN is not a recommendation, but be aware that its adjustment can affect # of I/Os and application performance

BUFFER POOLS AND STORAGE

Bufferpools and AUX storage

- What shows up in DB2 metrics regarding BP paged out to AUX?
  - Can I see evidence of paging in…
    - Sync I/O time?
    - BP Hit Ratio?
    - BP residency time?
      - **No!**
    - IFCID 225
    - Paging report
    - Class 2 not-accounted
    - z/OS uncaptured time
      - **Yes!**



Bufferpools and AUX storage

- Avoid PAGEINS for buffer pools – shows possible paging, but not the true extent
  - PAGEINS for READ/WRITE – DB2 came back to the oldest page on the LRU chain and in order to steal it for a new page we need to fix it in real before we read in the new page to take its place
    - This might occur at BP allocation, but should never be larger than the BP SIZE

If anything other than Pageins is > 0, increase the size of the virtual pool.

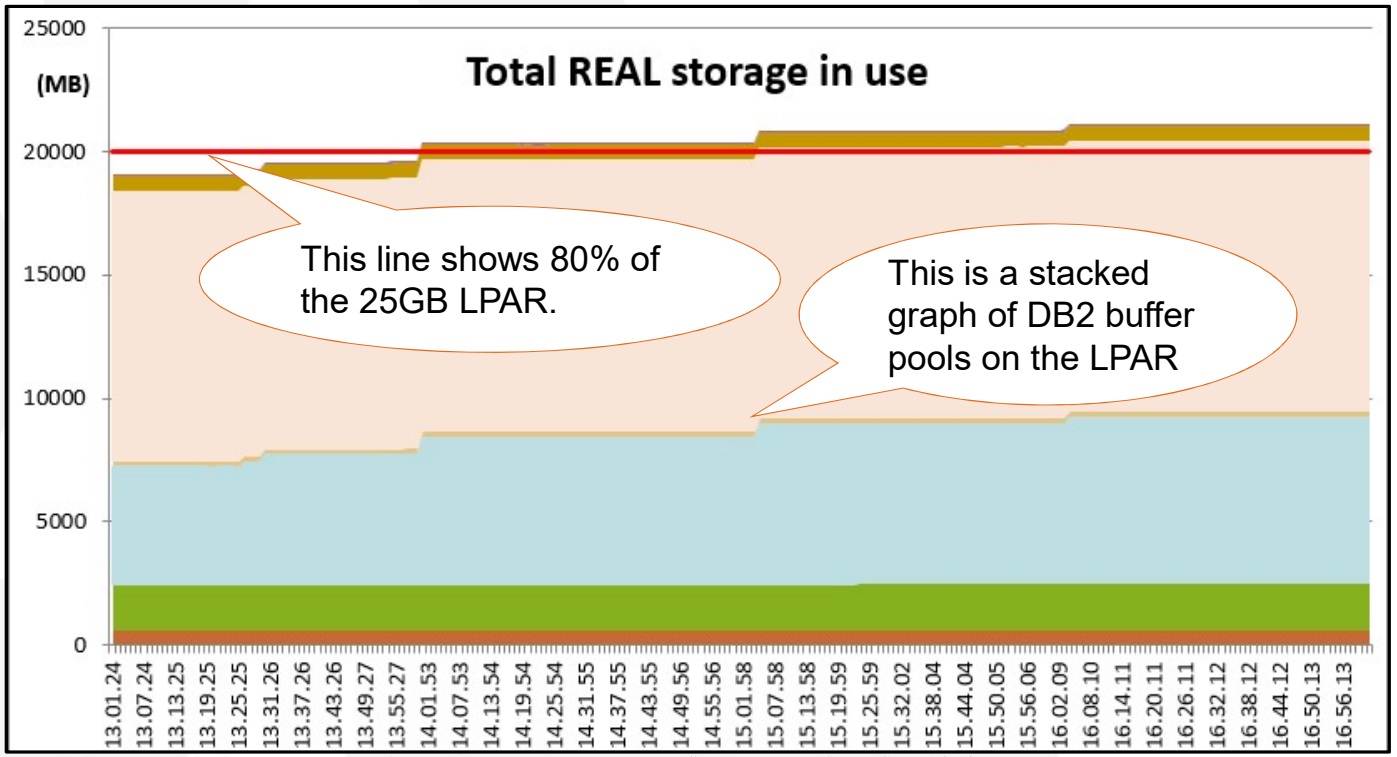
BPNAME	DEFERED WRITE THRESHHOLD	PREFETCH DISABLED NO BUFFS	PAGINS FOR WRITE IO	PAGEINS FOR READ IO	CRITICAL THRESHOLD REACHED
BP4	0	14	0	19110	0
BP4	0	103	0	0	
BP4	0	1	0	0	
BP4	0	1	0	0	
BP4	4	31	0	0	0

BPSIZE=1000

- MUST reserve space for DUMPs using MAXSPACE parameter (default of 500MB is inadequate)
  - (DBM1 – Buffer pools + total VPSIZE/6400 *1MB) + Shared memory + DIST + MSTR + IRLM + COMMON + ECSA
    - Buffer pool control blocks are VPSIZE/6400
  - PI85053 – suppress buffer pool control blocks in DUMP

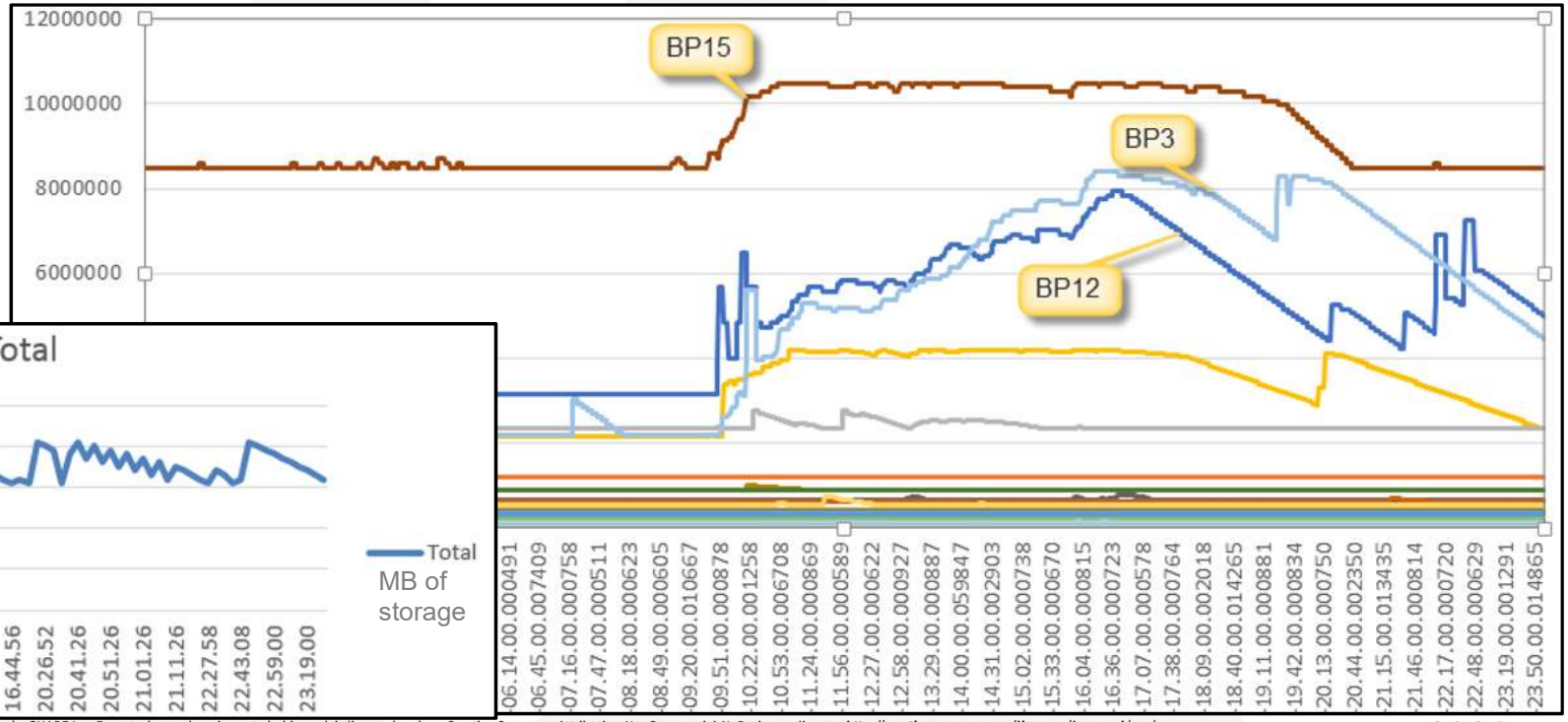
AUTOSIZE Buffer Pools…Caution

- AUTOSIZE(YES) in V10 allows WLM to alter the BPs automatically +/- 25% each time DB2 is recycled, in order to avoid sync I/O delay
  - Customer had this on, and PGFIX(YES) for all buffer pools
    - Pools didn't shrink until z/OS 2.1
  - V11 allows for a MINSIZE and MAXSIZE to be set
- Over time the BPs grew to consume 80% of the LPAR causing address spaces to be non-dispatchable = system slowdown!!



AUTOSIZE Buffer Pools…Caution

- AUTOSIZE(YES) does not have long term memory, it is always tuning
- The buffer pool could be altered as often as every 5 minutes
  - This changes the performance profile of applications as well as any SORTs occurring on the LPAR



AUTOSIZE Buffer Pools…Caution

- Recommend:
  - Use MINSIZE and MAXSIZE to control the total swing of the buffer pool in size
    - ** This means WLM can alter the BP to min and max without Db2 bouncing, no more +/- 25% limit
  - Choose a period of several days to monitor the changes over 1 pool – NOT at first allocation
  - Monitor several key business critical applications which run frequently and compare their performance to the intervals when the buffer pool was altered
  - Did the relative number of sync I/Os decrease when the size went up, and increase when the BP shrank, or can you even tell?
    - -DIS BPOOL(?) DETAIL(INTERVAL)
  - **Ensure you keep enough white space for large SORTs or DUMPs**
  - Based on these findings determine the sweet spot based on the law of diminishing returns
  - **Increases in the GBP sizes in the CFRM policy will be necessary after this exercise**
    - If using XES AUTOALTER then reset the baseline of INITSIZE and SIZE
    - Otherwise Cross-invalidation due to directory entry reclaim will increase
      - **These sync I/Os seen by the application will degrade any improvement done through tuning

DB2 12 and REAL storage

- DB2 12 will continue to trade real storage in return for CPU/zIIP savings
  - Thread footprint will likely increase 20%+ due to continuous delivery enhancements
  - 7-byte rid will mean MAXRBLK may need to be increased by 60%
  - FTB or Fast Traversal Blocks – saw up to 45% getpage reduction
    - Goal is to cache all non-leaf pages in memory
    - Min of 10MB up to 200GB, default of 20% of allocated BPs
      - Control it with entries into SYSIBM.SYSINDEXCONTROL and ZPARM to disable is INDEX_MEMORY_CONTROL
      - -DISPLAY STATS(INDEXMEMORYUSAGE)
    - Index requirements
      - Must be a unique index / Max of 64-byte index entries
  - Contiguous Buffer Pools - saw up to 8% CPU savings
    - Pre-req PGSTEAL(NONE)
      - Uses max of 10% or 6400 buffers for overflow area → increase VPSIZE by 6400 buffers
    - Eliminates LRU chains and LC14, LC24 contention
- PGFIX and Large Frame usage is a no brainer with ample REAL storage!

References

- Subsystem and Transaction Monitoring and Tuning with DB2 11 for z/OS
  - http://www.redbooks.ibm.com/redpieces/abstracts/sg248182.html?Open
- Db2 9: Buffer Pool Monitoring and Tuning
  - http://www.redbooks.ibm.com/abstracts/redp4604.html?Open
- Performance for PGSTEAL(NONE) buffer pool reduce I/Os after online REORGs
  - https://www-01.ibm.com/support/entdocview.wss?uid=swg1PI87228
- PI85053 – suppress buffer pool control blocks for SVC dump
  - https://www-304.ibm.com/support/entdocview.wss?uid=swg1PI85053&myns=swgimgmt&mynp=OCSSEPEK&mync=E&cm_sp=swgimgmt-_-OCSSEPEK-_-E

APPENDIX

Buffer pool pollution query (CAT/DIR)

```
-- catalog and directory objects in pools
SELECT R.BP_SORT
      , R.BPOOL
      , R.PAGESET
      , R.TYPE
      , COUNT(*) AS COUNT
      , DECIMAL(SUM(R.PAGES_ACTIVE_LEAF), 15, 0)
       AS PAGES_ACTIVE_LEAF
   FROM
   ( SELECT CASE
          WHEN SUBSTR(S.BPOOL CONCAT '   ', 4, 1)=' '
           THEN INT(SUBSTR(S.BPOOL, 3, 1))
          WHEN POSSTR(S.BPOOL,'K') = 0 THEN
           INT(SUBSTR(S.BPOOL CONCAT '   ', 3, 2))
          WHEN SUBSTR(S.BPOOL CONCAT '   ', 1, 5)
             = 'BP32K' THEN
           140+INT(SUBSTR(S.BPOOL CONCAT '   ', 6, 1)
            CONCAT '0')/10
          WHEN SUBSTR(S.BPOOL CONCAT '   ', 1, 4)
             = 'BP8K' THEN
           100+INT(SUBSTR(S.BPOOL, 5, 1))
          WHEN SUBSTR(S.BPOOL CONCAT '   ', 1, 5)
             = 'BP16K' THEN
           120+INT(SUBSTR(S.BPOOL, 6, 1))
          END AS BP_SORT
        , S.BPOOL AS BPOOL
        , 'TS' AS PAGESET
        , CASE WHEN D.NAME='DSNDB01' THEN 'D'
               WHEN D.NAME='DSNDB06' THEN 'C'
               ELSE D.TYPE
               END AS TYPE
        , ABS(S.NACTIVEF) AS PAGES_ACTIVE_LEAF
   FROM SYSIBM.SYSTABLESPACE S
     , SYSIBM.SYSDATABASE D
   WHERE D.NAME = S.DBNAME
   UNION ALL
   SELECT CASE
          WHEN SUBSTR(I.BPOOL CONCAT '   ', 4, 1)=' '
           THEN INT(SUBSTR(I.BPOOL, 3, 1))
          WHEN POSSTR(I.BPOOL,'K') = 0 THEN
           INT(SUBSTR(I.BPOOL CONCAT '   ', 3, 2))
          WHEN SUBSTR(I.BPOOL CONCAT '   ', 1, 5)
             = 'BP32K' THEN
           140+INT(SUBSTR(I.BPOOL CONCAT '   ', 6, 1)
            CONCAT '0')/10
          WHEN SUBSTR(I.BPOOL CONCAT '   ', 1, 4)
             = 'BP8K' THEN
           100+INT(SUBSTR(I.BPOOL, 5, 1))
          WHEN SUBSTR(I.BPOOL CONCAT '   ', 1, 5)
             = 'BP16K' THEN
           120+INT(SUBSTR(I.BPOOL, 6, 1))
          END AS BP_SORT
        , I.BPOOL AS BPOOL
        , 'IX' AS PAGESET
        , CASE WHEN D.NAME='DSNDB01' THEN 'D'
               WHEN D.NAME='DSNDB06' THEN 'C'
               ELSE D.TYPE
               END AS TYPE
        , ABS(I.NLEAF) AS PAGES_ACTIVE_LEAF
   FROM SYSIBM.SYSINDEXES I
     , SYSIBM.SYSDATABASE D
   WHERE D.NAME = I.DBNAME
) AS R
GROUP BY R.BP_SORT, R.BPOOL, R.PAGESET, R.TYPE
ORDER BY R.BP_SORT, R.BPOOL, R.PAGESET, R.TYPE
WITH UR;
```

Buffer pool pollution query (LOB)

```
--LOB and where they are
SELECT COUNT(*) AS LOB_COUNT, LOG AS LOGGED,
          CASE GBPCACHE
            WHEN 'A' THEN 'A'
            WHEN 'N' THEN 'N'
            WHEN 'S' THEN 'S'
            WHEN ' ' THEN 'C'
          END AS GBPCACHE, BPOOL
      FROM SYSIBM.SYSTABLESPACE S ,
      SYSIBM.SYSTABLEPART P
      WHERE S.DBNAME = P.DBNAME
      AND S.NAME = P.TSNAME
      AND TYPE = 'O'
      GROUP BY LOG, GBPCACHE, BPOOL
      ORDER BY LOG, GBPCACHE, BPOOL
      WITH UR;
```