

May 6-10, 2007
San Jose Convention Center
San Jose, California, USA

Session: B05 & B06

Buffer Pool Tuning for the new
Large Memory World
Parts 1 & 2

IDUG® 2007
North America

Joel Goldstein
Responsive Systems

May 8, 2007 9:20 a.m. – 11:40 a.m.

Platform: z/OS

Abstract:

The basics and effective techniques for pool tuning have not changed since the late 80's. While the advent of 64bit systems and Gigabytes of memory has expanded the payback potential for tuning - it is often less than many expect. The performance of the new disk controllers has enhanced overall performance, and pushed buffer pool performance limits and gains. Some common myths and mis-perceptions still persist, and seem to be re-fueled whenever there are significant changes to technology.

This presentation will take you from the basics of pool operation and performance, setting and impact of thresholds, and show you how to find the biggest tuning opportunities.

Many ROT's depend on whose thumb you are using. While there are necessary starting points for tuning exercises, as soon as we present an ROT, we have to address some exceptions to that rule.

Often the impacts of tuning changes vary depending on the version of DB2, version of z/OS, memory availability, and DASD technology.

Bullet Points – Presentation contents

- How pools work
- How and where do we get performance data
- Meaningful data, and when data is **NOT** meaningful
- Predicting the effect of tuning changes on pool performance
- 64bit Memory, and performance opportunities

The goal of this presentation is to give the attendees an understanding of how pools operate, the meaning and intent of pool thresholds, and an approach for tuning buffer pools. The pool performance analysis process often uncovers application performance issues that can provide huge cost reductions/savings if corrected.

Buffer Usage and Access

- Purpose of a pool
 - Keep frequently accessed data in memory
 - A page is read into a buffer
 - A page contains one or more rows
- A Getpage is caused by a data request from an application
- SetWrite to update a *row* of data
 - May be multiple per page, must be written

Every data system uses pools or caches to keep frequently referenced data in memory to avoid physical IOs.

Historical Changes

- 4 Pools
 - 3 - 4K Pools, 1 - 32K Pool
- 60 Pools
 - 50 - 4K Pools, 10 - 32K Pools
 - Now we have real tuning opportunities
 - Just because there are 50 - 4K pools, doesn't mean you should use that many
- 80 Pools
 - 50 -4K Pools, 10 -8K Pools, 10 -16K Pools 10 -32K Pools

We've come a long way over the last couple of decades. In perspective, the ability for these advances is keyed directly to the huge improvements in processor speed, and real memory availability.

What is a large pool?

- Two decades ago 10,000 buffers was a large pool
 - An application with 10 Million rows of data was huge
- Five years ago 50,000 buffers was a very large pool
- Today 100,000 is a large pool, and 500,000 buffers is a very large pool
 - 64bit memory allows us to access gigabytes of storage for buffer pools and other resources

Terabytes are coming....

The latest processors can have 512 Gigabytes of real memory. The movie 2001: A Space Odyssey was in 1968 – almost four decades ago.

When we have enough memory, HAL will be here...

What about System Activity/Volume?

- Getpage activity per hour
 - A lot more meaningful than number of SQL
 - An single SQL request may cause 2 or 3 Getpages, or 10 Million Getpages...
- Current large systems are > 250 Million Getpage requests per hour
- Some systems are > 6,000 IO/Sec
 - Huge tuning opportunity – *as we'll see in a later slide*

There are many measurements or sums we might look at to reflect the volume of work processed by any given system. However, metrics that contains other metrics that are variable, are not reliable. As example, SQL statements. Since most systems have hundreds to thousands of *different* SQL statements executed per day, and they may vary from a few to many thousands of getpage requests, this is not a useful metric.

Does bigger always = Better Performance?

- No, Yes, It depends.. ***Not necessarily***
- If more buffers reduce the IO rate/second
 - The amount of *bang* for your memory buck
 - Sometimes the hard number is significant, sometimes the percentage may be more meaningful
 - Is saving 5 IO/sec significant?
 - How much memory is used to get this?
 - If your rate was 10 IO/Sec, that's 50%
 - If your rate was 200 IO/Sec, that's *not* significant

Questions about results that may vary, depending on different inputs, don't have simple yes or no answers. There are too many variables to have a simple answer to this question. In many cases more memory will give better performance. Likewise, there are many situations where large pools will not give better performance. Another consideration should always be the tradeoff between more memory and IO reductions.

Tuning Fallacies

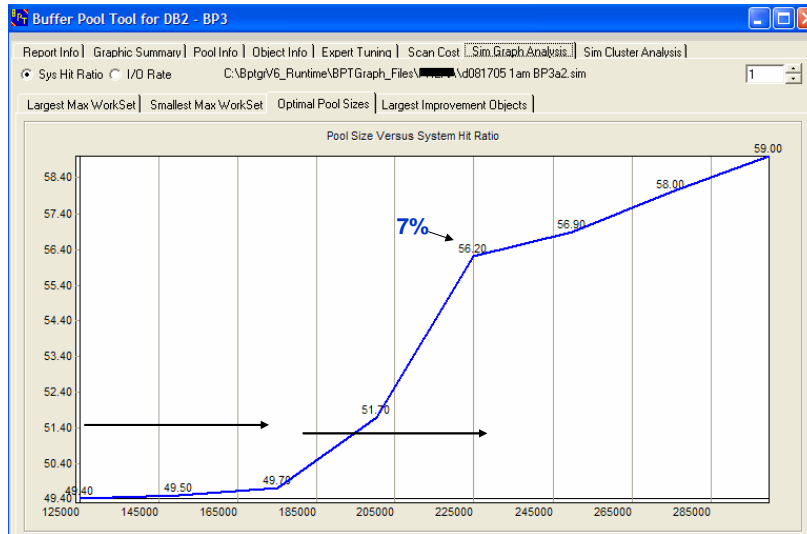
- Bigger is always better
- Just make it bigger, performance will get better
- You only need two or three pools to get good performance - almost always false, but in rare situations, this is true
 - Almost all random access, and *small working sets*
- Break up everything and use dozens of pools
 - Why not, we have 50 4K pools available...

Tuning Fallacies

- Buffer Pool Hit Ratio is a good performance predictor
 - We've used hit ratios for more than three decades
 - I am the author/creator of the system hit ratio formula
 - *I declare it dead !!*
- A Miss Ratio is a good performance measurement or predictor
 - This is simply (1 - hit ratio)
 - *Used to confuse....*

A miss ration is the reciprocal of a hit ratio, and is a useless metric. Since most of the world still thinks about a hit ration, any use a mess ratio seems to obfuscate any tuning issue, and is merely an attempt to be different rather than useful.

What does a Hit Ratio really tell you?



The first 50% does not get much payback

10

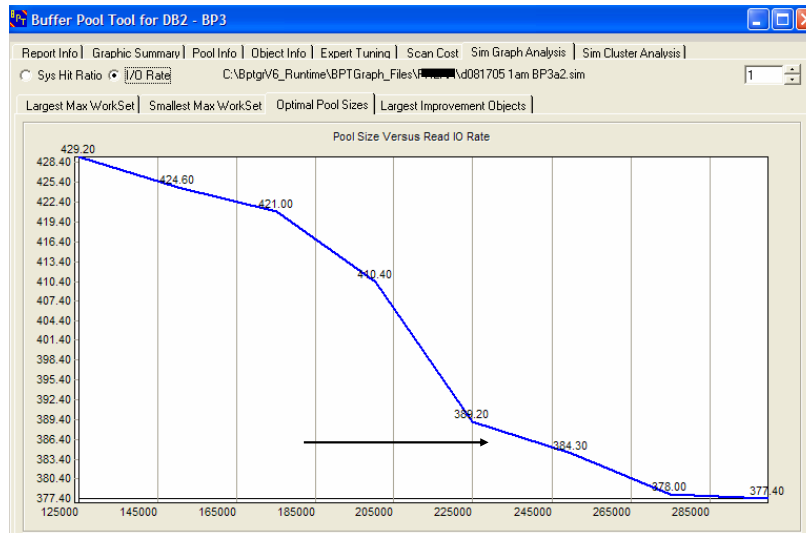
GoFurther

Ok, it shows you that performance is better. But how much better is it? How much CPU and elapsed times have been saved from I/O avoidance?

Increasing the pool by 50% does not give much payback, the next 50,000 shows a large improvement, and then the improvement curve flattens.

Again, it looks nice, but you can't take any of the numbers to the bank.

The I/O rate is a *measurable* Metric



Why does the next 50% help so much?

11 A critical WKSET was reached

The I/O rate is convertible into CPU costs, and elapsed time savings.

This is not just a suggestion to make the pool larger, it shows you the real benefit, and where to stop.

It shows you that the first 50,000 additional buffers don't provide much payback, but the next 50,000 give a huge payback.

The large payback from the second increment of 50,000 buffers is because we passed a critical working set threshold for a heavily accessed object. As stated earlier, the wkset size of an object has nothing to do with the number of pages shown in the catalog. It is the number of pages in the pool at a specific point in time.

The I/O rate is a meaningful Metric

INTV DATE	INTV TIME	GET PAGE	SYNC IO	SEQ PREFETCH	LIST PREFETCH	DYN PREFETCH	IORATE /SEC	HIT RATIO
2007-02-21	00.00.00	1,187,068	43,331	625,280	156,611	202	459	30.47% ****
2007-02-21	00.30.00	10,913,350	342,348	3,226,519	289,002	1,069,556	2,737	54.85%
2007-02-21	01.00.00	671,955	51,854	373,032	49,730	893	264	29.24%
2007-02-21	01.30.00	743,700	67,202	435,841	2,948	95	281	31.95%
2007-02-21	09.30.00	4,105,758	124,423	2,961,328	12,364	69,313	1,760	22.85%
2007-02-21	10.00.00	4,232,240	100,715	3,088,771	9,277	64,952	1,813	22.88%
2007-02-21	10.30.00	3,127,959	70,646	2,452,744	18,310	58,303	1,444	16.88% ****
2007-02-21	11.00.00	3,890,741	112,328	2,520,929	9,911	51,934	1,497	30.73%
2007-02-21	11.30.00	4,011,848	117,173	2,394,283	11,760	74,255	1,443	35.26%
2007-02-21	12.00.00	4,580,930	109,663	2,339,369	639,288	68,607	1,754	31.09%
2007-02-21	12.30.00	6,150,020	188,803	3,202,333	11,852	64,002	1,926	43.63%

If the hit ratio was meaningful, it would not show a large increase when there is a large increase to the IO rate % GP increase vs. IO

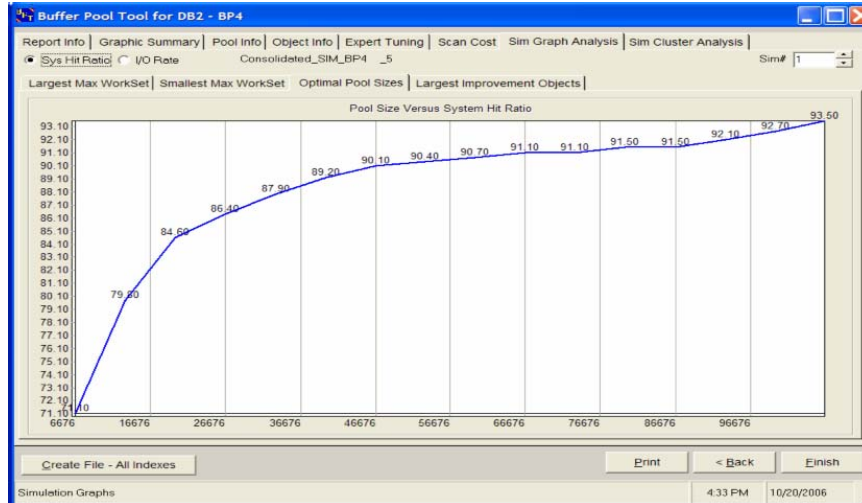
The IO rate can increase, and the hit ratio can increase.

The IO rate can decrease, and the hit ratio can decrease.

This is the opposite the “expectation”

Changes in the workload, the type of accesses taking place, and the objects in use, cause the counter-intuitive swings of the hit ratio. But it’s counter-intuitive only if you are looking uniquely at getpages and IOs.

Bigger is *not* always better - 1



© Responsive Systems 2007

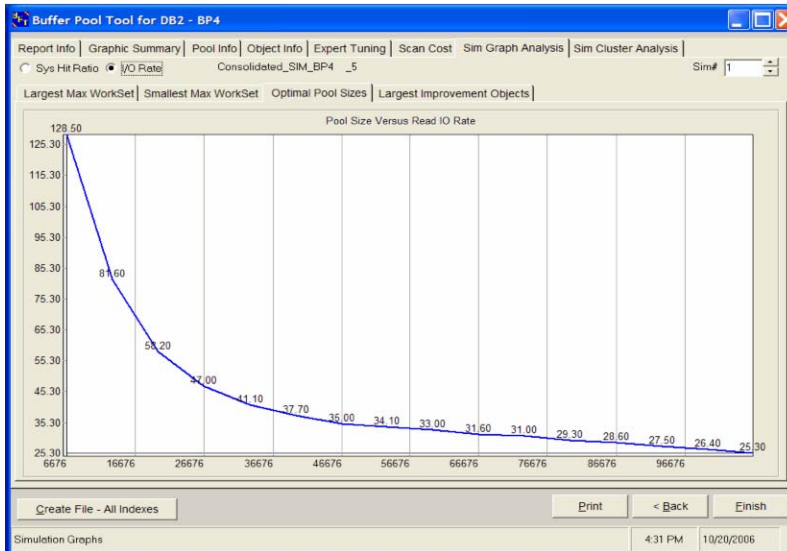
13

GoFurther

The increase for the buffer pool hit ratio flattens, and drops to .4% for every 10,000 buffers, 40 meg of memory.

As stated many times, these gains cannot be equated to elapsed times or CPU reductions.

Bigger is *not* always better - 2



The IO reduction/gain from increasing the pool size flattens, and eventually drops to only 1 IO/Sec per 10,000 buffers. This is not considered a useful payback for 40 Meg of memory.

The previous slide with graph showed..

- When a pool is much too small, more memory will provide substantial improvements
- There is a point of limited, and possibly no return
 - Further increases provide very little gain, not enough to justify the added memory
 - The first tripling of size cut the IO rate 50%, **70 IOs**
 - Then a doubling cut the IO rate by **15 IOs**
 - The third doubling cut the IO rate by **5 IOs**
- Well look at some pool size increase examples later...

Pool/Object Access Types

- Random
- Sequential
- Dynamic Sequential
 - Optimizer access path is random
- List Prefetch

Pool Usage Methods

- LRU
 - Default
- MRU
 - Utilities
- FIFO
 - Optional specification
 - Slightly less overhead, no LRU queue management

LRU – Least recently used

MRU – Most recently used

FIFO – First in, First out

Buffer Pool Access

- The pool is not scanned, DB2 uses hash tables to determine if a page is in a pool
- Using hash tables is very efficient, with only a few entries (generally less than 5) accessed - microseconds
- The cost to find and access a page remains relatively constant regardless of pool size
 - So too big doesn't cost you cpu cycles, it just wastes memory

Sequential Prefetch Operation

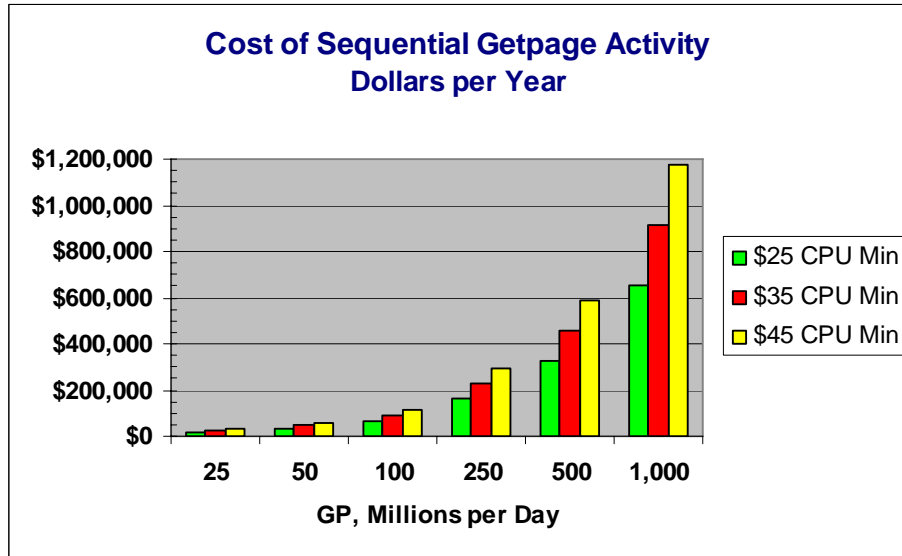
- The access path is sequential
- Starts with a Synch IO for page 1
- Concurrent prefetch requests for pages 2-32 and 33-64
 - Aggressive approach to be sure pages are in the pool when needed
- When page 32 is accessed, the Buffer Manager issues a read for pages 65-96...

Sequential Page Read Quantity – 4K Pools

- Based on pool size
- < 256 buffers 8
- 256 – 999 16
- 1000 and higher 32
- DB2 V9 32 - 64
 - If $vpseq * \#buffers > 40,000$, then 64

Sequential access is bad, *or maybe good...*

- If you need lots of data, it's often the fastest and lowest cost approach – batch jobs
- When objects are scanned *repeatedly*, this is very costly, and a performance problem
 - Indexes – SP access is *almost* always bad
 - I see a lot of this in some online systems, especially SAP and PeopleSoft applications, and other canned vendor apps
 - Data Propagator...



© Responsive Systems 2007

22

GoFurther

This is based on a 2064 processor with 210 MIP engine speeds.

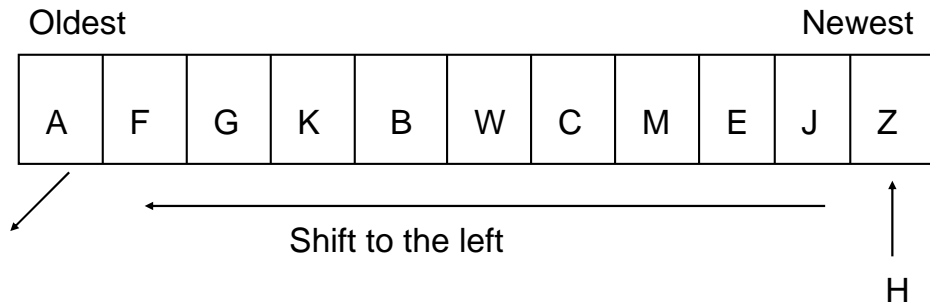
Pool Thresholds

- **VDWQT** Vertical deferred write threshold
- **DWQT** Deferred write threshold
- **SPTH** Sequential prefetch threshold
- **DMTH** Data manager threshold
- **IWTH** Immediate write threshold
 - Might not be an issue.....

Green is good, yellow is a warning, and red is bad. We want to see most of the write threshold hits for VDWQT, and a low number for DWQT.

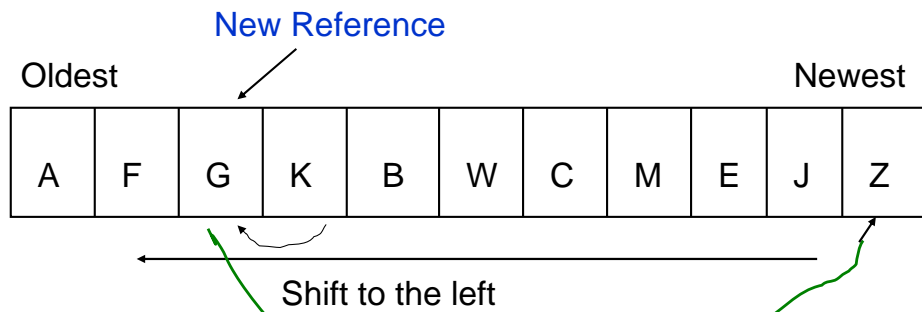
IWTH isn't a problem is there aren't any hits for SPTH and DWTH

LRU Queue



Page A is the oldest page on the Q chain, so is freed to allow a new page H to be read into the pool. For our purposes here, it's easiest to think that the pages just shift one place to the left.

LRU Queue



G moves to the right, top of the queue

GoFurther

When a page G, that is in the pool is re-referenced, it is placed at the top (right) of the LRU Q, and the other pages are shifted left in terms of age in the Q.

Determine how pools and objects are used

- Buffer pool displays are rarely useful, and only provide high level summary information – *over a period, or interval*
- Online monitors are much better at showing important information, but –
 - Aside from showing that you have a problem, and might be hitting thresholds...
 - They can't *predict the effect of any changes*
 - Program your own alerts and thresholds...
- We need *detailed information about object access*

There are many places to obtain performance data, depending upon the level of information you need. For general analysis, detailed statistics information is available from the SMF 100 record. Online monitors access this same information from DB2, and may use varying intervals to obtain data. When we get to the aspect of pool tuning, we really need detailed object access information so we can predict the effect of pool tuning changes, and not make mistakes with a critical production system.

Where does detailed information come from?

- Detailed pool level information
 - Lower level detail or trace reports from the SMF 100 statistics records – shows all the important access and usage information for a pool, and the pool operational parameters
 - This is where to start, to determine if there are problems and/or performance opportunities.

All online monitors have a reporting facility to format and print this data. It may come from the SMF 100 records, or from data created by the monitor and placed in a file or database.

Where does detailed information come from?

- Detailed object level information
 - Buffer manager trace records
 - IFCID 198
 - IO Trace records
 - IFCIDs 6, 7, 8, 9, 10
 - Shows IO start, IO end, pages read/written
- This data volume is too high to use SMF
- GTF may pose other problems – overhead, wraps dataset

Now we get down to the real guts of DB2 to see how objects are really accessed.

Instrumentation – V7 buffer manager trace info

```

QW0198 DSECT
QW0198DB DS H DATABASE ID
QW0198OB DS H PAGESET OBID
QW0198BP DS X BUFFERPOOL ID
QW0198FC DS C FUNCTION CODE
* **.....QW0198FC CONSTANTS.....**
QW0198GP EQU C'G' GET PAGE REQUEST
QW0198SW EQU C'S' SET WRITE INTENT REQUEST
QW0198RP EQU C'R' RELEASE PAGE REQUEST
QW0198PS DS C PAGE STATUS IN BUFFER POOL
* APPLICABLE ONLY WHEN QW0198FC = 'G', QW0198PS=X'00' WHEN QW0198FC='S' OR 'R'
* **.....QW0198PS CONSTANTS.....**
QW0198H EQU C'H' PAGE HIT IN VIRTUAL BUFFERPOOL
QW0198M EQU C'M' PAGE MISSED IN VIRTUAL BUFFERPOOL
QW0198N EQU C'N' NOREAD REQUEST
QW0198AT DS C ACCESS TYPE - QW1098AT IS
* NOT APPLICABLE WHEN QW0198FC = 'S' QW0198AT = X'00' WHEN QW0198FC = 'S'
* **.....QW0198AT CONSTANTS.....**
QW0198SQ EQU C'S' SEQUENTIAL ACCESS (GET PAGE)
QW0198RN EQU C'R' RANDOM ACCESS (GET PAGE)
QW0198RL EQU C'L' RIDLIST ACCESS (GET PAGE)
QW0198SR EQU C'N' STANDARD REQUEST (RELEASE PAGE)
QW0198DR EQU C'D' DESTRUCTIVE REQUEST (RELEASE PAGE)
QW0198MR EQU C'M' MRU SCHEME APPLIED (RELEASE PAGE)
QW0198PN DS F PAGE NUMBER
QW0198AC DS A ACE ADDRESS
QW0198PR DS C Page refresh status.
* Applicable only when QW0198PS='M' (virtual buffer pool miss).
* **.....QW0198PR CONSTANTS.....**
QW0198RH EQU C'H' Page retrieved from Hiperpool
QW0198RG EQU C'G' Page retrieved from Group buffer pool
QW0198RD EQU C'D' Page retrieved from DASD
DS XL3 Reserved
    
```



These apply only to Reads, not Writes



The field reference data and constant values are critical pieces of information to interpret the records properly.

Instrumentation – V7

IO trace info

```

*****
* IFC ID 0006 FOR RMIID 10 RECORDS THE ID OF THE DATA SET BEFORE *
* A READ I/O OPERATION *
*****
QW0006 DSECT          IFCID(QWHS0006)
QW0006DB DS XL2      DATABASE ID (DBID)
QW0006OB DS XL2      PAGESET OBID
QW0006BP DS F        BUFFER POOL INTERNAL ID (0-49 and
*                      80-89)
QW0006PN DS XL3      FIRST PAGE NUMBER TO BE READ (for
*                      non large table space)
QW0006F DS C        FLAG FOR TYPE OF READ
QW0006FS EQU C'S'    SEQUENTIAL PREFETCH REQUEST
QW0006FL EQU C'L'    LIST PREFETCH REQUEST
QW0006FD EQU C'D'    DYNAMIC SEQUENTIAL PREFETCH REQUEST
QW0006FR EQU C'R'    READ REQUEST
QW0006AC DS F        ACE TOKEN OF REQUESTOR
QW0006PG DS F        FIRST PAGE NUMBER TO BE READ
QW0006FG DS C        FLAG FOR TYPE OF TABLE SPACE
QW0006F1 EQU C'N'    NON LARGE TABLE SPACE
QW0006F2 EQU C'L'    NON-EA LARGE TABLE SPACE
QW0006F3 EQU C'V'    EA-LARGE TABLE SPACE
*                      DS CL3      RESERVED

*****
* IFC ID 0007 FOR RMIID 10 RECORDS THE COMPLETION CODE AFTER *
* THE READ I/O OPERATION *
*****
QW0007 DSECT          IFCID(QWHS0007)
QW0007MM DS F        MEDIA MANAGER RETURN CODE - 0 SUCCESSFUL
QW0007DB DS XL2      DATABASE ID (DBID)
QW0007OB DS XL2      PAGESET OBID
QW0007AC DS F        ACE TOKEN OF ACTUAL REQUESTOR.
*                      THIS MAY DIFFER FROM THE ACE TOKEN IN THE STANDARD
*HEADER FOR THIS RECORD, EG IN SEQUENTIAL PREFETCH.
•QW007NP DS H        Number of Pages Read

```

© Responsive Systems 2007

31

GoFurther

Knowing how many pages were actually read by a read is critical, since the buffer manager only reads the pages it needs.....

It knows the pages that are already in the pool

Instrumentation – V8 *New/Additional Data*

```
*****
* IFC ID 0006 FOR RMID 10 RECORDS THE ID OF THE DATA SET BEFORE *
* A READ I/O OPERATION *
*****
```

QW0006	DSECT		IFCID(QWHS0006)
QW0006DB	DS	XL2	DATABASE ID (DBID)
QW0006OB	DS	XL2	PAGESET OBID
QW0006BP	DS	F	BUFFER POOL INTERNAL ID (0-49 and
*			80-89)
QW0006PN	DS	XL3	FIRST PAGE NUMBER TO BE READ (for
*			non large table space)
QW0006F	DS	C	FLAG FOR TYPE OF READ
QW0006FS	EQU	C'S'	SEQUENTIAL PREFETCH REQUEST
QW0006FL	EQU	C'L'	LIST PREFETCH REQUEST
QW0006FD	EQU	C'D'	DYNAMIC SEQUENTIAL PREFETCH REQUEST
QW0006FR	EQU	C'R'	READ REQUEST
QW0006AC	DS	F	ACE TOKEN OF REQUESTOR
QW0006PG	DS	F	FIRST PAGE NUMBER TO BE READ
QW0006FG	DS	C	FLAG FOR TYPE OF TABLE SPACE
QW0006F1	EQU	C'N'	NON LARGE TABLE SPACE
QW0006F2	EQU	C'L'	NON-EA LARGE TABLE SPACE
QW0006F3	EQU	C'V'	EA-LARGE TABLE SPACE
	DS	CL3	RESERVED
QW0006PT	DS	F	Partition number or 0 if non-
*			partitioned



This let's us see usage by partition

GoFurther

This new Partition information let's us see how each partition is really used.

Instrumentation – V8 Additional Data

```

*****
* IFC ID 0007 FOR RMID 10 RECORDS THE COMPLETION CODE AFTER      *
* THE READ I/O OPERATION                                         *
*****
*
QW0007  DSECT          IFCID(QWHS0007)
QW0007MM DS  F          MEDIA MANAGER RETURN CODE - 0 SUCCESSFUL
QW0007DB DS  XL2        DATABASE ID (DBID)
QW0007OB DS  XL2        PAGESET OBID
QW0007AC DS  F          ACE TOKEN OF ACTUAL REQUESTOR.
*
* THIS MAY DIFFER FROM THE ACE TOKEN IN THE STANDARD
* HEADER FOR THIS RECORD, EG IN SEQUENTIAL PREFETCH.
QW0007NP DS  H          NUMBER OF PAGES READ
*
* DATA SECTION 2 FOR THE IFCID 7 RECORD IS THE ID OF THE PAGES THAT
* ARE PREFETCHED SUCCESSFULLY VIA AN IO OPERATION. THIS IS ONLY
* PRESENT WHEN QW0006F IS S, L, OR D. THIS IS A REPEATING GROUP.
QW00072  DSECT
QW0007PF DS  F          PAGE PREFETCHED VIA AN IO OPERATION
    
```

Now we know *exactly which pages* are read into the pool, critical information for Dynamic and LP operations.

We have been asking for this data since DB2 Version 3...

What can we do with this new data?

- Track exact page usage and re-reference page data
- Improve the accuracy of performance prediction techniques
- Shine a light into the previous black holes of List prefetch especially, and also Sequential prefetch and Dynamic prefetch IOs
 - The previous difficulty....

How do you know that your pools need tuning?

- Users are complaining about response/throughput
 - IO wait is a substantial component of elapsed times
 - Percentage
- Online monitors (or other tools) indicate performance issues
 - Hitting thresholds, running out of read/write engines
 - Poor pool performance indicators
 - Critical transactions exceeding response targets
- I have a problem - where do I go?
 - *Dive into the pool....*

We probably all know that tuning should be a pro-active process. Unfortunately most installations are under-staffed, and tuning becomes a fire-fighting role.

Performance Hierarchy – Page access

- Memory – Buffer Pool
 - Microseconds
- DASD cache
 - Milliseconds
 - 1 Ms or less if there is no contention
- Those *multiple* spinning disks (cache miss)
 - Anyplace from 10 > 30 Ms
 - Sometimes 50 – 60 Ms

These are huge performance differentials, and remember that an IO, even when the page is in the Cache, causes a huge difference in CPU cost compared to accessing a page in pool memory.

Pool Tuning Hierarchy

- Thresholds
 - Critical
 - IWTH, DMTH, SPTH
 - Important
 - DWQT, VDWQT
- Other problem indicators?
 - Need detailed performance data
 - Summary statistics are not adequate

Start from the most critical items and work your way down. If critical thresholds aren't being hit, then start looking at the IO rates, percentage of sequential access to objects, etc, etc.

Pool Tuning Hierarchy

- IO Rate/Second
 - Pool level
 - Object level
 - Partition
- Hit Ratio
 - Historically interesting but useless as a metric
 - Must use the System Hit %, not Application Hit %

The System Hit ratio factors in the number of pages actually read into the system.

Lots of Dynamic Prefetch may make the Hit Ratio Negative...

An application hit ratio is really useless, and ignores the number of pages actually read.

Tuned the pools, and performance is worse...

- What was your tuning methodology
 - Did you have one, or was it a SWAG?
 - What were your expectations?
 - Based on what?
- What metrics are you using?
 - At what level?
- *If your high level metrics really imply a degradation....*

Tuning should not be a guessing game. If you used a methodology for tuning, has it failed for some reason? If so, why.

Maybe it really hasn't, and you need to look deeper into the real data and numbers.

Tuned the pools and performance is worse

- **Let's make sure.....**
 - **Is it really worse?**
 - Or has the before vs. after workload changed?
 - Removing one bottleneck will bring others to the top of the list
 - Reducing IO may not reduce CPU busy rates
 - Like pushing on a water balloon.....

In my experience, a very large percentage of perceived performance degradations are really changes in the workload.

Remember, you need to compare an apple to an apple. If the workload isn't reasonably repeatable, there will always be performance variations.

Comparing Performance Measurements

- Something changed....
 - Determine **what** changed...
 - Sometimes it's obvious, often hard to determine
 - Workload may be completely different at the object access and usage levels
- Easy things
 - Different time frame
 - Different time duration
 - **Increased Getpage activity, or shift from one pool to another**
 - Where and why

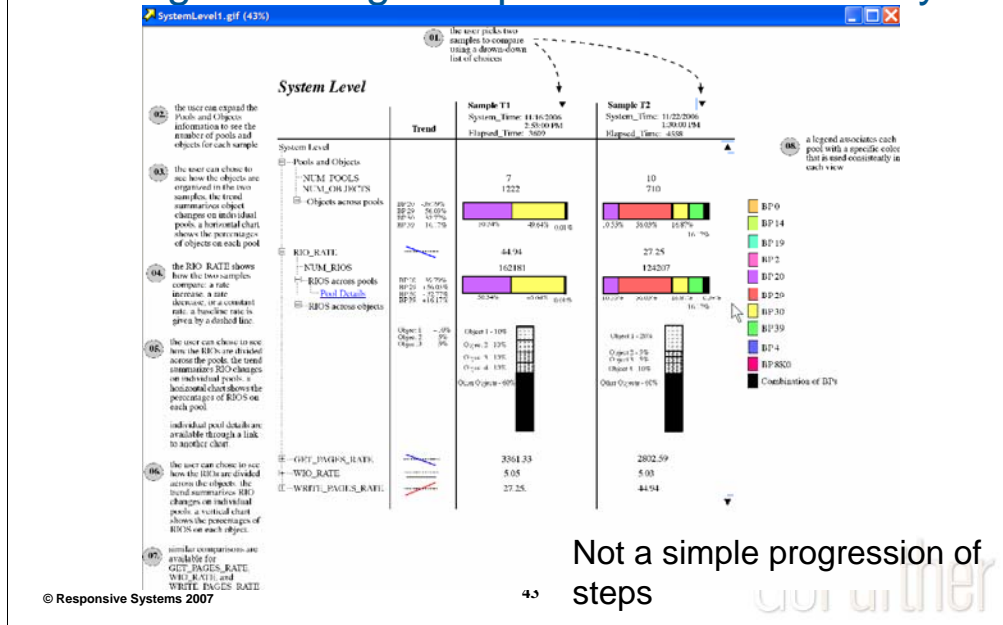
Take the easy points first, and then keep digging. Start with a big shovel...

Comparing Performance Measurements

- Has the mix or ratio of access changed?
 - Random vs. Sequential
 - List Prefetch, Dynamic Prefetch
 - Dynamic Prefetch is still Random Access
- Different objects in use, or the most heavily accessed objects are *not the same as the last run*
- Did some large batch jobs slip into your online performance period?
xxx

At some point you may be working with a needle and a magnifying glass to find and detail the workload shifts.

Looking for changes in performance & activity

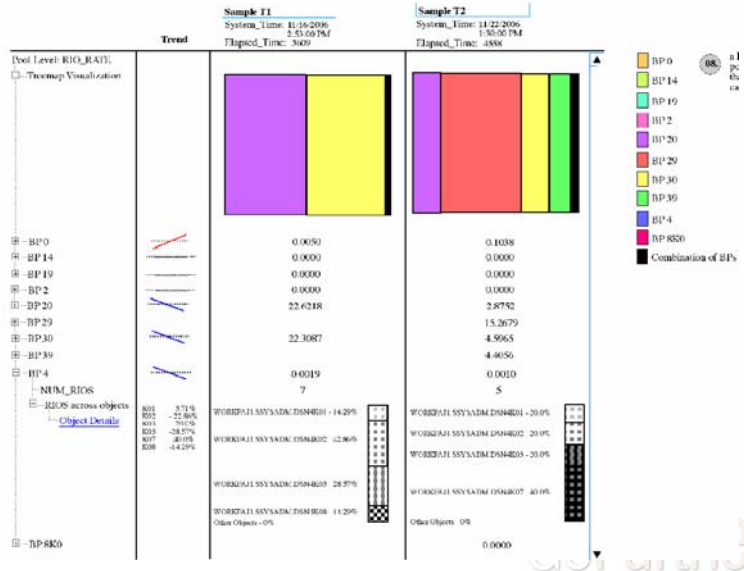


Not a simple progression of steps

An automated approach to comparing data would be nice. Well, we're working on that. It's quite complex, and not like a simple numeric progression of steps. Any given finding will engender it's own next analysis step and path...

Looking for changes in performance & activity

Pool Level



© Responsive Systems 2007

Some things may be obvious, such as seeing different pools used in two sets of data we are comparing. The upward slanting red lines are negative changes, downward slanting lines are positive changes.

Looking for changes in performance & activity

IO Rates/Second

Object Level

	Trend	Sample T1	Sample T2
		System_Time: 11/16/2006 2:53:00 PM Elapsed_Time: 3609	System_Time: 11/22/2006 1:30:00 PM Elapsed_Time: 4588
Object Level: RIO_RATE			
[-] WORKPAJ1.SSYSADM.DSN4K01		0.00028	0.00022
[-] NUM_RIOS		1	1
[-] WORKPAJ1.SSYSADM.DSN4K02		0.00083	0.00022
[-] WORKPAJ1.SSYSADM.DSN4K03		0.0000	0.00022
[-] WORKPAJ1.SSYSADM.DSN4K05		0.00055	0.0000
[-] WORKPAJ1.SSYSADM.DSN4K06		0.0000	0.0000
[-] WORKPAJ1.SSYSADM.DSN4K07			0.00044
[-] WORKPAJ1.SSYSADM.DSN4K08		0.00027	0.0000

Looking for changes in performance & activity

Individual metric level

OBJECTS	RIO_RATE	GET_PAGES_RATE	WIO_RATE	WRITE_PAGES_RATE	POOL	POOL.PERCENTAGE	SYSTEM.PERCENTAGE
WORKPAJ1SSYSADM.DSN4K01					BP-4		
WORKPAJ1SSYSADM.DSN4K02					BP-4		
WORKPAJ1SSYSADM.DSN4K03					BP-4		
WORKPAJ1SSYSADM.DSN4K05					BP-4		
WORKPAJ1SSYSADM.DSN4K06					BP-4		
WORKPAJ1SSYSADM.DSN4K07					BP-4		
WORKPAJ1SSYSADM.DSN4K08					BP-4		
.							
.							
.							

Performance tuning complaint...

"I modified a bufferpool to set the DWQT from the default 50% value to DWQT=4 and VDWQT = 0 - No other change -

After the change , a program doing the inserts (in ascending key), and delete on it, took a lot of time, 40min instead of 17min (it is a temporary table), STROBE shows that 90% of the time was on the Insert, and 80% of the Wait was on "OTHER WRITE".

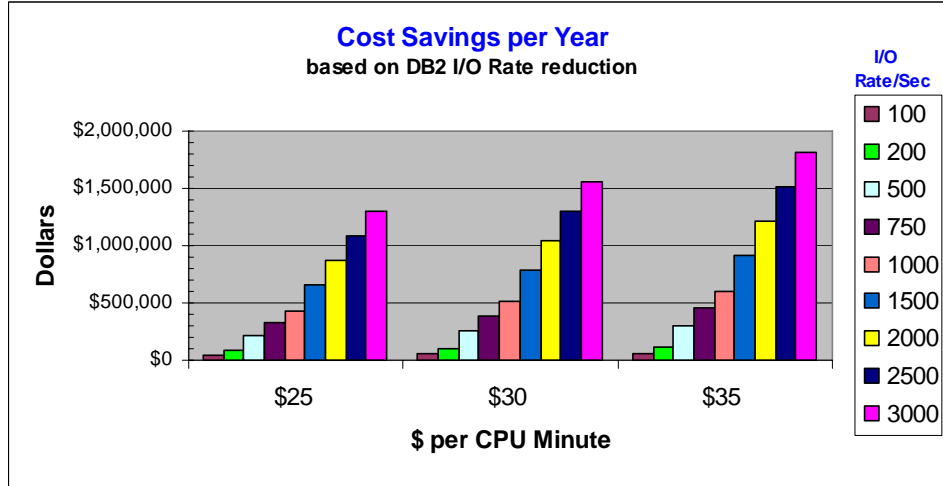
I proved to them that the INDEX and TS has never been organized , and an increase in volume can magnify the problem. (Statistics show 70,000 inserts instead of 10,000 during this day) **That's 7 times the workload**

They told me that the problem comes from the change, because even if it was disorganized , other executions were good... "

Three blind mice....

Wow, do you think that increasing the workload by a factor of 7 has any impact?

Eliminating I/O Saves Money !!



© Responsive Systems 2007

48

GoFurther

These I/O rate per sec savings, up through 2,500 per second, have been achieved by clients

Buffer Pool Performance Data - Red Flags

BP5	GENERAL	QUANTITY	/SECOND	/THREAD	/COMMIT
	CURRENT ACTIVE BUFFERS	410.10	N/A	N/A	N/A
	UNAVAIL.BUFFER-VPOOL FULL	0.00	0.00	0.00	0.00
	NUMBER OF DATASET OPENS	8446.00	0.10	0.01	0.00
	BUFFERS ALLOCATED - VPOOL	7507.00	N/A	N/A	N/A
	DFHSM MIGRATED DATASET	0.00	0.00	0.00	0.00
	DFHSM RECALL TIMEOUTS	0.00	0.00	0.00	0.00
	VPOOL EXPANS. OR CONTRACT.	0.00	0.00	0.00	0.00
	VPOOL OR HPOOL EXP.FAILURE	0.00	0.00	0.00	0.00
	CONCUR.PREF.I/O STREAMS-HWM	300.00	N/A	N/A	N/A
	PREF.I/O STREAMS REDUCTION	3382.00	0.04	0.00	0.00
	PARALLEL QUERY REQUESTS	88003.00	1.02	0.12	0.03
	PARALL.QUERY REQ. REDUCTION	215.00	0.00	0.00	0.00
	PREF.QUANT.REDUCED TO 1/2	866.2K	10.06	1.21	0.34
	PREF.QUANT.REDUCED TO 1/4	73224.00	0.85	0.10	0.03

What critical data items are missing from these sets of data?

There are a lot of red flags in this data report, all related to a lack of buffers available for prefetch.

Buffer Pool Performance Data - Red Flags

BP5	READ OPERATIONS	QUANTITY	/SECOND	/THREAD	/COMMIT
	BPOOL HIT RATIO (%)	31.59			
	GETPAGE REQUEST	179.4M	2083.30	250.37	70.40
	GETPAGE REQUEST-SEQUENTIAL	130.1M	1510.58	181.54	51.05
	GETPAGE REQUEST-RANDOM	49309.8K	572.71	68.83	19.35
	SYNCHRONOUS READS	5285.0K	61.38	7.38	2.07
	SYNCHRON. READS-SEQUENTIAL	1797.4K	20.88	2.51	0.71
	SYNCHRON. READS-RANDOM	3487.7K	40.51	4.87	1.37
	GETPAGE PER SYN.READ-RANDOM	14.14			
	SEQUENTIAL PREFETCH REQUEST	3856.3K	44.79	5.38	1.51
	SEQUENTIAL PREFETCH READS	3684.0K	42.79	5.14	1.45
	PAGES READ VIA SEQ.PREFETCH	109.6M	1273.45	153.05	43.04
	S.PRF.PAGES READ/S.PRF.READ	29.76			
	LIST PREFETCH REQUESTS	1328.8K	15.43	1.85	0.52
	LIST PREFETCH READS	392.2K	4.56	0.55	0.15
	PAGES READ VIA LIST PREFETCH	2049.5K	23.80	2.86	0.80
	L.PRF.PAGES READ/L.PRF.READ	5.23			
	DYNAMIC PREFETCH REQUESTED	197.4K	2.29	0.28	0.08
	DYNAMIC PREFETCH READS	184.4K	2.14	0.26	0.07
	PAGES READ VIA DYN.PREFETCH	5726.7K	66.51	7.99	2.25
	D.PRF.PAGES READ/D.PRF.READ	31.06			
	PREF.DISABLED-NO BUFFER	0.00	0.00	0.00	0.00
	PREF.DISABLED-NO READ ENG	74.00	0.00	0.00	0.00
	PAGE-INS REQUIRED FOR READ	0.00	0.00	0.00	0.00
			50		

© Responsive Systems 2007

GoFurther

Buffer Pool Performance Data - Red Flags

BP5	WRITE OPERATIONS	QUANTITY	/SECOND	/THREAD	/COMMIT

	BUFFER UPDATES	18460.7K	214.41	25.77	9.39
	PAGES WRITTEN	2220.6K	25.79	3.10	1.13
	BUFF.UPDATES/PAGES WRITTEN	8.31			
	SYNCHRONOUS WRITES	25517.00	0.30	0.04	0.01
	ASYNCHRONOUS WRITES	1063.6K	12.35	1.48	0.54
	PAGES WRITTEN PER WRITE I/O	2.04			
	HORIZ.DEF.WRITE THRESHOLD	1.00	0.00	0.00	0.00
	VERTI.DEF.WRITE THRESHOLD	451.00	0.01	0.00	0.00
	DM THRESHOLD	0.00	0.00	0.00	0.00
	<u>WRITE ENGINE NOT AVAILABLE</u>	<u>85.00</u>	0.00	0.00	0.00
	PAGE-INS REQUIRED FOR WRITE	0.00	0.00	0.00	0.00

Running out of read engines & write engines is often a sign of DASD performance problems.

What critical data items are missing from these sets of data?

The number of buffers in the pool, and pool thresholds.. And the ELAPSED TIME for the data !!

Sometimes bigger is better – *if you have memory*

- Let's look at a situation where bigger is better
- Large system
- High IO rate
- A lot of random access
- Working set sizes don't monopolize the pool

Bigger is sometimes better - 1

Buffer Pool Tool for DB2 - BP2

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | I/O Cost | Sim Graph Analysis | Sim Cluster Analysis

Collection

Date: 2007-02-23
Time: 08:44:24
Elapsec Time: 00:23:16

System Info

System: MLP1
Sub System: DBSP
DB2 Version: 8.1
DS Group: *NA*

Pool	RI/O/Sec	Get Pages	Updates	Hit Ratio	I/O	WIO/Sec	Pages/Write	Write I/Os	Pages Written	Avg F
BP0	0.75	621243	475	99.4	1204	0.11	2.61	156	407	
BP1	416.04	7472286	108325	48	619260	27.56	1.51	38475	58088	
BP2	716.19	16950394	403796	76.2	1100325	72.01	1.92	100522	193425	
BP3	8.72	286163	1240	76.7	12660	0.35	1.67	489	819	
BP4	148.01	1625281	19469	-4.7	210419	2.72	2.13	3804	8099	
BP5	0.00	222	0	100	0	0.00	0.00	0	0	
BP6	0.07	343599	8247	100	244	0.11	3.94	147	579	
BP7	0.82	3824097	876631	99.8	1415	0.20	29.87	276	8243	
BP9	15.14	364584	22908	82.5	23477	1.68	3.04	2343	7123	
BP10	105.29	616879	199	-38.6	147050	0.04	1.95	59	115	
BP11	39.05	449399	7466	-11.8	55453	0.68	1.83	946	1731	
BP13	0.00	1	1	100	1	0.00	1.00	1	1	
BP15	14.00	207860	5916	75.8	20371	0.59	2.07	823	1701	
BP16	136.78	4897467	864	94.8	191654	0.50	1.18	704	828	
BP17	15.77	807118	28501	96.4	30701	6.23	1.68	8691	14588	
BP20	21.80	791535	59526	92.1	33509	2.21	2.66	3083	8207	
BP21	18.94	664894	68165	87.7	33575	5.11	3.07	7135	21916	
BP22	187.42	1510879	8730	78.4	267010	3.85	1.45	5376	7817	
BP30	2.23	876207	8700	99.5	3736	0.44	1.60	616	985	
BP31	12.64	416561	20739	93.3	25457	5.59	1.73	7810	13547	
BP49	0.06	1600	624	95.1	138	0.04	2.92	59	172	
BP12K	0.02	76556	55681	99.9	66	0.01	1.33	36	48	

Total 4K Buffer: **509,600**

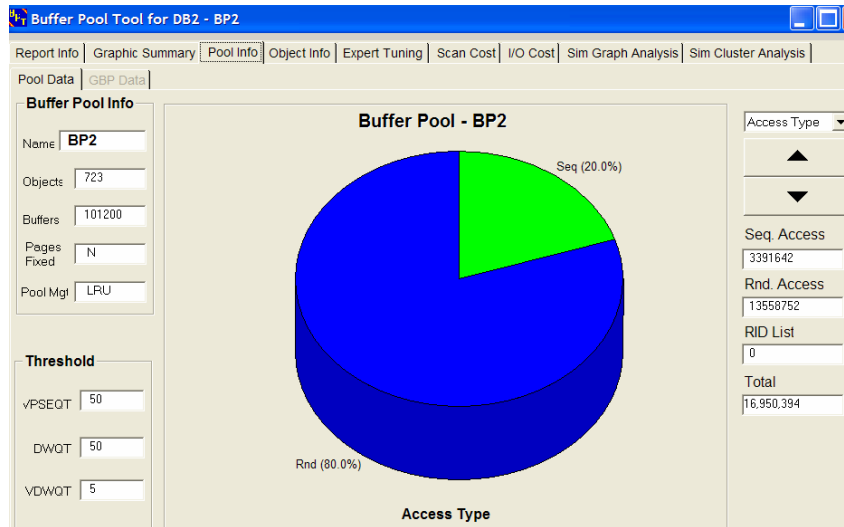
Total Read/Write IO	2,778,000	Total Get Pages	46,273,095
Overall Sys Hit Ratio	74.14	Total I/Os per second	1,989.97
Total Updates	1,706,202	Pages per write	1.92

© Responsive Systems 2007

53

The system is using more than 2 Gig of memory for the pools. BP2 is the heavy IO pool, and the pool is all indexes.

Bigger is sometimes better - 2



© Responsive Systems 2007

54

Go further

BP2 is the heavy IO pool, and the pool is all indexes. There are 723 objects in the pool, there is a lot of SP access on some of the indexes.

Bigger is sometimes better - 3

Object	Seq. Access	Rnd. Access	RID
DCUST.CL	2191678	296	
DBROP.BF	1133	881052	
DCUST.CL	44	869533	
DBROP.BF	0	696582	
DBROP.BF	0	694230	
DBROP.BF	0	501178	
DBROP.BF	0	487206	
DAOPI.AOI	0	425055	
DBROP.BF	0	394561	
DAR.AR.IB	0	366484	
DBROP.BF	193	359438	
DAOPI.AOI	0	336997	
DSYSCON	0	320683	
DBROP.BF	0	295379	
DBROP.BF	0	244626	
DAR.AR.IB	0	230515	
DBROP.BF	0	214129	
DBROP.BF	0	212171	
DBROP.BF	0	202405	
DBROP.BF	103313	98112	
DAR.AR.IB	0	167286	
DBROP.BF	0	158170	
DEDI.EDI>	0	158144	
DBROP.BF	0	138352	

The heaviest SP index. Objects with the green bars are partitioned.

Bigger is sometimes better - 4

Buffer Pool Tool for DB2 - BP2

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | I/O Cost | Sim Graph Analysis | Sim Cluster Analysis

Overall Collection Time = 1,396.000 System CPU Seconds Cost = 85.6825
 Projected System 24Hr IO CPU Cost = 5,302.98 BP2 CPU Seconds Cost = 32.993
 Projected System 24 Hrs IO Delay = 329,672.57 System IO Delay Secs = 5,326.65

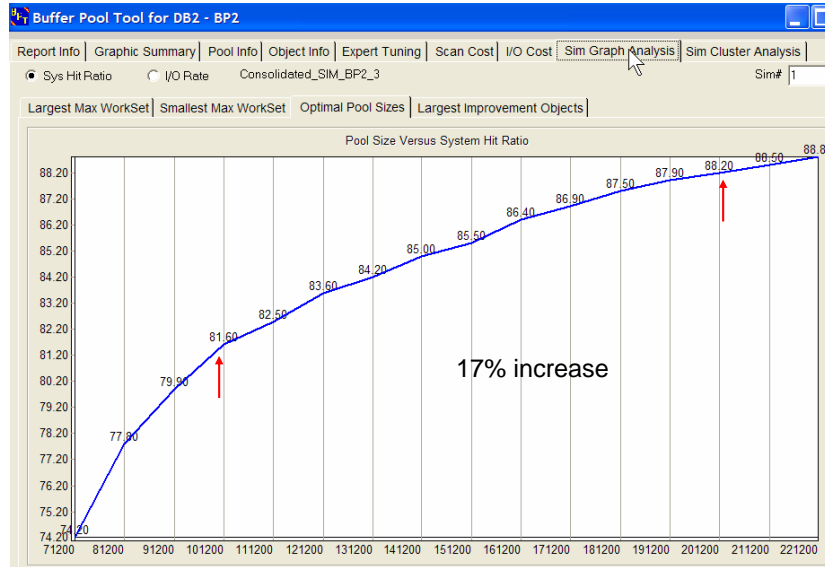
System IO Cost | Pool IO Cost

Object Name	Pool	Total I/O	CPU Sec Cost	I/O Elap Sec	24Hrs IO Delay
DBROP.DB2PDBA.SBROP607	BP22	261634	8.634	1036.248	64134.547
DBROP.DB2ADM1.SBROP001	BP4	206615	6.818	526.824	32605.726
DBROP.BROP.XITMBALP	BP16	152103	5.019	301.086	18634.549
DBROP.DB2PDBA.SBROP300	BP10	146991	4.851	243.172	15050.187
DBROP.DB2PDBA.SBROP613	BP1	57136	1.885	266.13	16471.083
DBROP.DB2ADM1.SBROP003	BP11	54507	1.799	152.976	9467.856
DBROP.DB2PDBA.SBROP641	BP1	50940	1.681	305.64	18916.401
DBROP.BROP.XPIMNBH1	BP2	50782	1.676	203.128	12571.819
DWEBMRO.DB2PDBA.SMR00010	BP1	47169	1.557	15.294	946.563
DBROP.BROP.XITMPRO3	BP2	42996	1.419	85.202	5273.247
DCUST.CUST.XCOPRDSTP	BP2	39006	1.287	3.912	242.118
DBROP.BROP.XPROUSE1	BP2	38513	1.271	153.836	9521.082
DBROP.DB2PDBA.SBROP615	BP1	34933	1.153	77.577	4801.327
DBROP.BROP.XITMPRO1	BP2	34347	1.133	26.175	1620.
DBROP.BROP.XPOHDRP	BP2	31857	1.051	80.004	4951.537
DBROP.BROP.XITMBAL2	BP16	28546	0.942	109.58	6782.029
DCUST.DB2ADM1.SCUST002	BP1	28117	0.928	82.893	5130.34
DBROP.DB2PDBA.SBROP313	BP1	26704	0.881	12.471	771.844
DBROP.DB2PDBA.SBROP612	BP1	25843	0.853	89.396	5569.953
DBROP.DB2ADM1.SBROP002	BP9	21134	0.697	114.954	7114.632
DBROP.BROP.XORDHDRP	BP2	20246	0.668	63.584	3935.285

© Responsive Systems 2007 56 GoFurther

The object with the heaviest IO rate for the entire system is really in BP22.
While the difference in total IOs between the top two objects is less than 30%,
note the difference in the elapsed seconds.
This comes from factoring in the avg IO times for the objects.

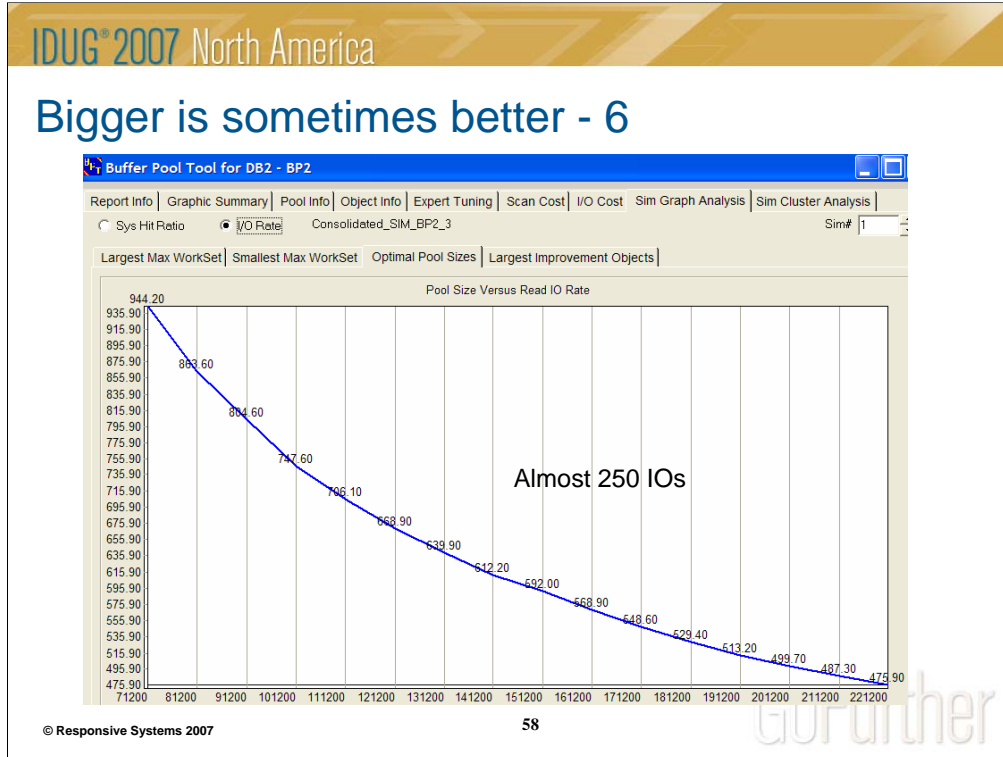
Bigger is sometimes better - 5



Coming from the original pool size of 101,200 buffers, doubling the pool size increases the hit ratio by 17%.

Impressive increase, but what does it really mean?

Bigger is sometimes better - 6



Here we see the real payback. To start with, the simulated IO rate at the original pool size is about 30/sec higher than the statistics indicated – and this is less than a 5% difference. More than acceptable for a simulation, and shows that the simulation has good accuracy. Doubling the pool size will save about 250 IO/sec. A great payback, if memory is available. Every additional 10,000 buffers past that size saves about 12 IO/sec. Note that the curve is starting to flatten....

Bigger is sometimes better – 7 this is why...

Cluster Info

Clusters

Object	Smallest Max	Largest Max
1	13730	16437
2	8789	9979
3	5949	7083
4	1982	4475
5	359	1823
6	1	353

Object Data

Type	Object	Max Work S
I	DBROP.BROP.XITMPRO1	13730
I	DSYSCON.SYSCON.XSCMAST1	14873
I	DBROP.BROP.XSHPHDRP	16437

The working set sizes of the three objects in the first cluster, at the original pool size of 101,200.

Bigger is sometimes better – 8 this is why...

Buffer Pool Tool for DB2 - BP2

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | I/O Cost | Sim Graph Analysis | Sim Cluster Analysis

Pool Usage Intent: Sequential Random Consolidated_SIM_BP2_3

Sim #: 1 Pool Size: 201200 Cluster Rad: 1.2

Cluster Info

Object	Smallest Md	Largest Md
1	15872	18128
2	14129	14304
3	9208	10345
4	4815	6949
5	2617	4293
6	464	2376
7	1	457

Type	Object	Max Work Set
I	DSYSCON.SYSCON.XSCMAST1	15872
I	DBROP.BROP.XSHPHDRP	18128
I	DBROP.BROP.XPIMNBH1	16177

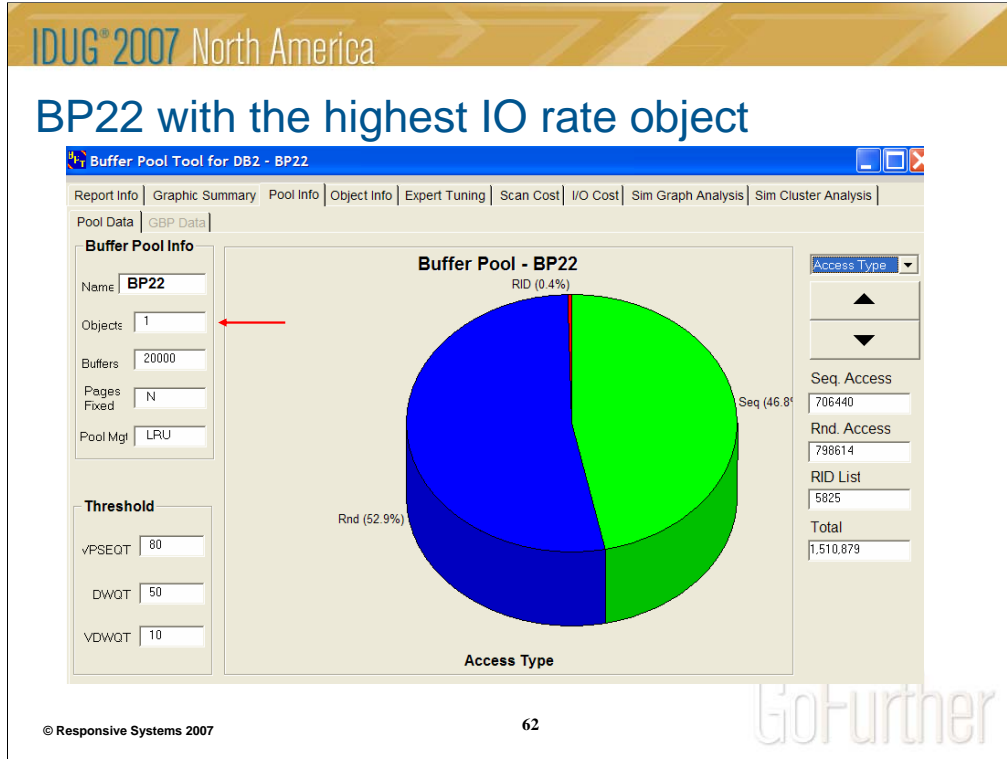
No real growth of Wksets...

The working set sizes of the three objects in the first cluster, at a pool size of 202,400. Two of the objects in the original cluster did not show substantial growth, and the object that dropped out of the cluster showed even less growth. This indicates that many of the objects have reached a “critical mass” of working set size, and most of the frequently accessed pages are staying in the pool.

Remember BP22?

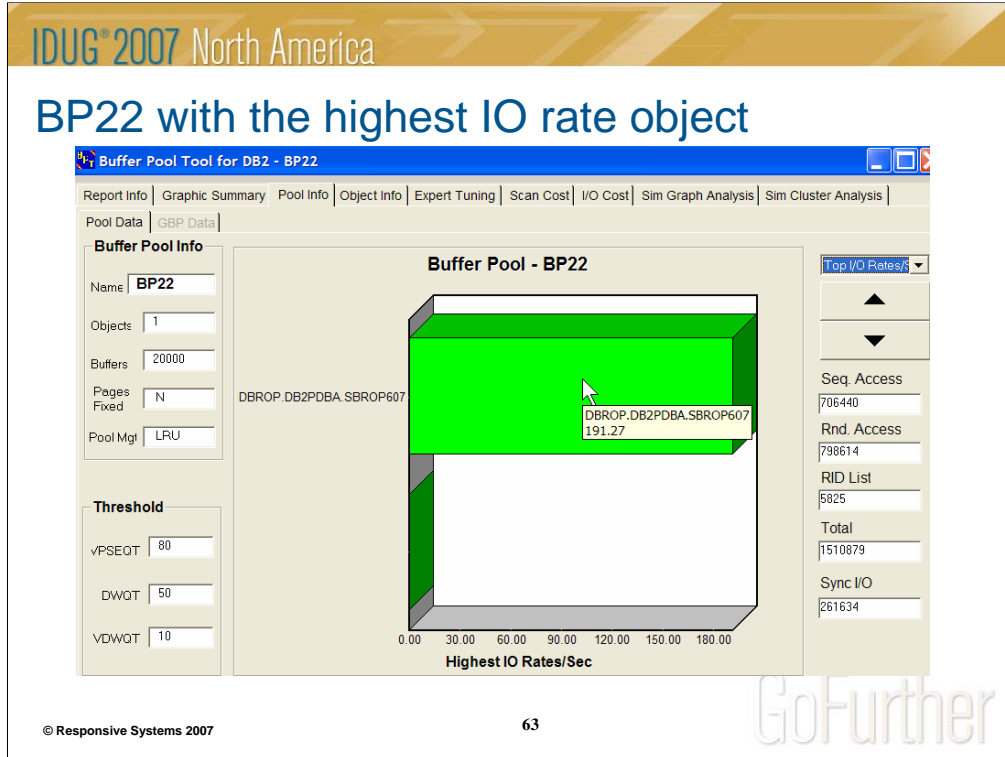
- That had the object with the highest number of IOs
- At the top of the IO cost chart
- Let's see what we can find there...

BP22 with the highest IO rate object



Access is close to 50% SP

BP22 with the highest IO rate object



Well, only one object, and 20,000 buffers.

BP22 with the highest IO rate object

Buffer Pool Tool for DB2 - BP22

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | I/O Cost | Sim Graph Analysis | Sim Cluster Analysis

Pool Data | GBP Data

Buffer Pool Info

Name: BP22
 Objects: 1
 Buffers: 20000
 Pages Fixed: N
 Pool Mgt: LRU

Threshold

VPSEQT: 80
 DWQT: 50
 VDWQT: 10

App Hit Ratio 82.9	Pages Read Sync 259062	Total Get Pages 1510879
System Hit Ratio 78.4	Pages Read Seqpr 58442	Get Page Rand 798614
Read IO Rate/sec 187.42	Pages Read Listpr 1570	Get Page Seq 706440
Pages / Write 1.45	Pages Read Dynpr 6784	Get Page RidList 5825
Reads For Seqpr 2,152	Reads For Dynpr 292	Reads For Listpr 128
Avg Synchron IO (ms) 4	Avg SP IO (Seq Pref) 8	Avg SP IO (Dyn Pref) 8
		Avg SP IO (List Pref) 21

Seq. Access: 706440
 Rnd. Access: 798614
 RID List: 5825
 Total: 1510879
 Sync I/O: 261634

Close

© Responsive Systems 2007

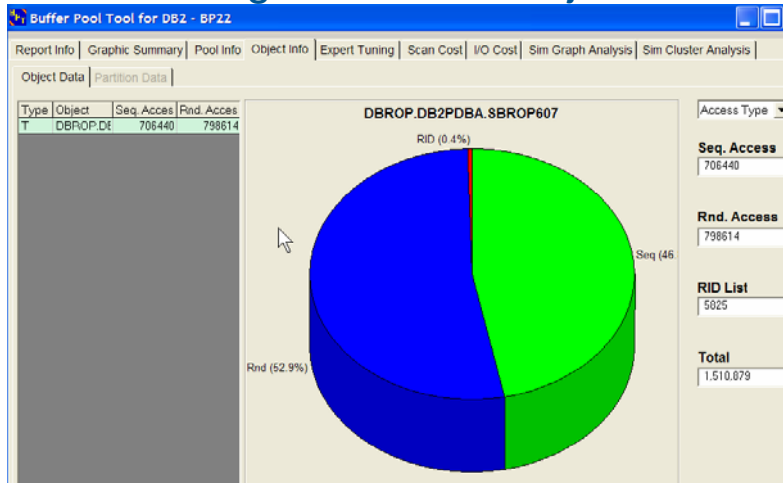
64

Really low DP...

GoFurther

The low dynamic prefetch usage indicates that the pages accessed are rather random, not in an ascending sequence, and not close together.

BP22 with the highest IO rate object



It's a partitioned object

One object, almost 1/2 SP access. Object is partitioned.

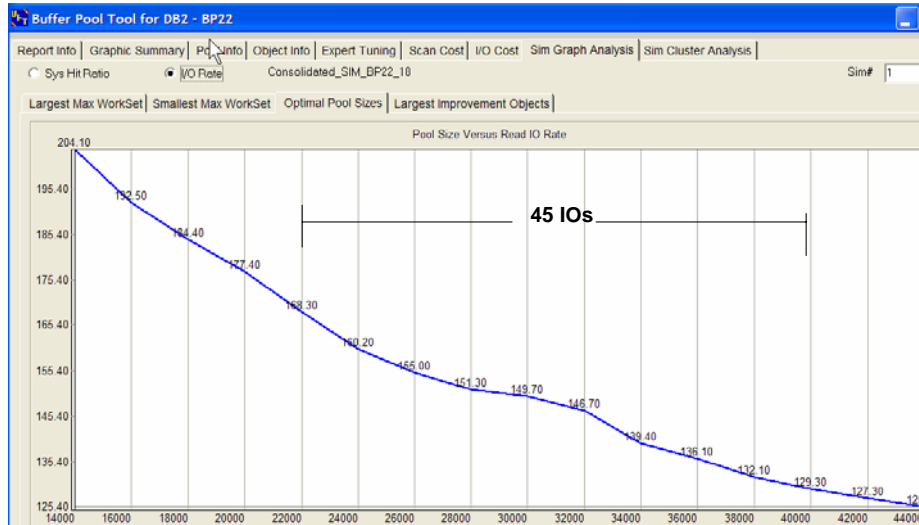
BP22 with the highest IO rate object

Partition	Object	Seq. Acc	Rnd. Acc	R
4	DBROP.DB2PDBA.S	115614	150603	
1	DBROP.DB2PDBA.S	97238	133230	
9	DBROP.DB2PDBA.S	131258	82208	
7	DBROP.DB2PDBA.S	111041	86227	
5	DBROP.DB2PDBA.S	71848	96833	
10	DBROP.DB2PDBA.S	80956	63275	
3	DBROP.DB2PDBA.S	54724	69105	
2	DBROP.DB2PDBA.S	29304	71119	
6	DBROP.DB2PDBA.S	7244	30192	
8	DBROP.DB2PDBA.S	7213	15822	

Differing access percentages for the partitions

Some are mostly SP, some are mostly random.

BP22 with the highest IO rate object



© Responsive Systems 2007

67

GoFurther

Doubling the pool size saves more than 45 IO/sec, because the randomly accessed pages can remain in the pool longer.

Reducing the vpsqz should also allow random pages to stay in the pool longer, but here too we see a case where bigger is better.

So when is bigger really better? When the pool has a large random access component, and the pages accessed are not widely dispersed across a very large object.

IDUG® 2007 North America
 Different Client
 Large system with a high IO rate > 200M GP/Hr

Buffer Pool Tool for DB2 - BPO

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | Sim Graph Analysis | Sim Cluster Analysis

Collection

Date: 2006-02-27
 Time: 12:01:26
 Elapsed Time: 00:03:00

System Info

System: SYZ2
 Sub System: DPW2
 DB2 Version: 8.1
 DS Group: DPW/G

Pool	RI/O/Sec	Get Pages	Updates	Hit Ratio	I/O	W/O/Sec	Pages/Write	Write I/Os	Pages Written	Δ
BP0	173.67	311326	42	25.3	31263	0.01	1.50	2	3	
BP1	1,797.01	3618533	72231	73.4	326365	16.13	5.73	2904	16636	
BP2	1,741.15	1207515	113379	27.1	315393	11.03	14.91	1986	29603	
BP3	5.82	495508	357766	99.4	1047	0.00	0.00	0	0	
BP4	354.53	130087	236	25.6	63836	0.11	7.05	20	141	
BP5	255.49	1997672	0	91.3	45989	0.00	0.00	0	0	
BP6	468.21	506337	216	76.5	84299	0.12	5.57	21	117	
BP7	9.91	19927	35	51.6	1791	0.04	15.13	8	121	
BP8	320.41	149384	49649	56.7	59484	10.06	18.40	1811	33319	
BP9	555.14	578578	18	81.3	99931	0.03	1.83	6	11	
BP10	389.24	144112	27	41.3	70064	0.01	3.00	1	3	
BP11	42.86	546021	25481	94.6	8179	2.58	10.93	465	5082	
BP12	150.82	357332	0	75.8	27147	0.00	0.00	0	0	

Total 4K Buffers 546,000

Total Read/Write IO	1,134,788	Total Get Pages	10,062,332
Overall Sys Hit Ratio	71.66	Total I/Os per second	6,304.38
Total Updates	619,080	Pages per write	11.77

© Responsive Systems 2007

68

GoFurther

This is a different DB2 system, at a different client site. The system is over 200Million Getpages/Hour, with a very high I/O rate.

Dramatic IO rate reduction

Buffer Pool Tool for DB2 - BP2

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | Sim Graph Analysis | Sim Cluster Analysis

Collection	Pool	RIQ/Sec	Got Pages	Updates	Hit Ratio	I/O	WIO/Sec	Pages/Write	Write I/Os	Pages Written	Avg
	BP0	7.47	62196	109	79.4	1906	0.00	0.00	0	0	0
	BP1	1,065.63	5841602	145332	85.3	276934	20.38	10.20	5198	53023	
	BP2	069.47	1746550	162273	57.4	224564	11.10	16.49	2050	47009	
	BP3	7.19	691664	481468	99.2	1833	0.00	0.00	0	0	0
	BP5	9.23	2540312	0	99.2	2353	0.00	0.00	0	0	0
	BP6	298.95	925587	291	90.6	76255	0.09	4.43	23	102	
	BP7	3.95	10731	97	94.1	1043	0.14	21.03	35	764	
	BP8	145.64	123651	25702	67.3	45712	33.62	25.09	8574	215079	
	BP9	190.70	843507	24	94	50693	0.01	1.67	3	5	
	BP10	145.45	99983	24	62.9	37091	0.00	2.00	1	2	
	BP11	111.32	536843	11167	86.5	29302	3.59	18.51	915	16941	
	BP12	7.85	754492	0	99.2	2001	0.00	0.00	0	0	0
	BP13	81.88	1289201	308	91.8	20891	0.04	13.91	11	153	
	BP14	275.82	339407	1499	54.7	70427	0.36	14.59	93	1357	

Date: 2006-07-18
Time: 13:50:04
Elapsed Time: 00:04:15

System Info:
System: SYZZ
Sub System: DPW2
DB2 Version: 8.1
DS Group: DPWG

Total 4K Buffers: **744,000**

Total Read/Write IO	841,006	Total Get Pages	15,813,806
Overall Sys Hit Ratio	86.13	Total I/Os per second	3,298.06
Total Updates	828,294	Pages per write	18.89

GP rate is up 10%, IO rate decreased 3,000/Sec

© Responsive Systems 2007

69

GoFurther

Now the system is over 223 Million Getpages/Hour, with a greatly reduced, but still high I/O rate.

The getpage rate is more than 10% higher, while the IO rate has decreased almost 50%.

Tuning changes to the system

- Two more pools were added
 - BP13 and BP14
 - Moved five indexes into BP13
 - Moved five tablespaces into BP14
- Added 198,000 buffers
 - 800 Meg of memory
- Almost 3 Gigabytes of total memory for buffer pools

The proven tuning methodology is grouping objects by access type, and working set sizes. This approach provided tremendous benefits at this client site.

Different Client
Before 64bit memory (different system)

Buffer Pool Tool for DB2 - BP1

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | Sim Graph Analysis | Sim Cluster Analysis

Collection	Pool	RIO/Sec	Get Pages	Updates	Hit Ratio	I/O	WIO/Sec	Pages/Write	Write I/Os	Pages Written	Avg Pg Res Sec
	BP0	18.07	1941222	1027	97.9	34848	0.06	1.86	120	223	1881
	BP1	724.69	17910647	31452	0.3	397024	2.50	2.50	4963	12431	5
	BP2	391.80	33529607	35919	84	759382	3.30	2.86	6343	16876	1615
	BP3	147.29	13605543	106547	90.0	310501	14.26	2.41	27411	66060	1552
	BP4	0.00	35895	24389	100	82	0.04	2.89	80	231	1921
	BP6	26.54	1062105	23104	37.3	54362	1.75	2.75	3357	9229	717
	BP7	25.89	722964	49370	77.1	57708	4.13	2.56	7944	20357	1480
	BP13	272.06	617010	923	-243.3	524716	0.14	1.64	271	444	0
	BP30	1.34	471319	396991	95.7	4074	0.78	26.18	1491	39032	1839
	BP40	3.91	04000	17	62.5	7530	0.01	1.00	17	17	1200
	BP49	0.00	906	725	100.1	0	0.00	0.00	0	0	1924
	BP32K	5.06	85487	39853	65.1	10782	0.55	1.82	1063	1934	1262
	BP8K0	0.03	411	22	3.2	67	0.01	1.42	12	17	60
	BP16K0	0.00	251	24	100	8	0.00	1.14	7	8	1921

System Info
 System: NENT
 Sub System: NBP
 DB2 Version: 7.1
 DS Group:

Total 4K Buffers: **1,017,600**

Total Read/Write IO: **3,161,892**
 Overall Sys Hit Ratio: **49.93**
 Total Updates: **790,443**

Total Get Pages: **70,076,165**
 Total I/Os per second: **1,645.11**
 Pages per write: **3.14**

VDWQT..

Just a bit over 4 Gig of memory for the 4K pools



We're starting off with more than 4 gig of memory allocated for buffer pools, using dataspace.

Using the new 64bit memory

Buffer Pool Tool for DB2 - BP1

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | Scan Cost | Sim Graph Analysis | Sim Cluster Analysis

Collection	Pool	RI/O/Sec	Get Pages	Updates	Hit Ratio	I/O	WIO/Sec	Pages/Write	Write I/Os	Pages Written	Avg Pg Rte
	BP0	0.02	3545001	1007	100	443	0.10	2.96	375	1074	
	BP1	242.42	9220112	23633	0.7	877299	1.27	2.54	4586	11627	
	BP2	225.87	12734737	46132	39	823621	2.92	2.58	10497	27069	
	BP3	216.26	21625620	525505	76.6	840358	17.17	2.20	61815	136042	
	BP4	0.10	128934	42694	99.7	593	0.07	5.83	242	1412	
	BP6	61.32	2875925	35264	51.9	225242	1.25	2.95	4491	13247	
	BP7	46.87	1649555	315218	37.6	186374	4.90	2.33	17636	41171	
	BP13	5.17	65920	1631	-74.1	19064	0.12	1.64	435	714	
	BP20	65.92	8714309	123	27.1	237386	0.02	1.46	65	95	
	BP21	35.08	516997	104	31.2	126363	0.02	1.41	58	82	
	BP22	13.66	1320392	367	20.1	49234	0.01	2.20	45	99	
	BP23	10.30	756483	211	29.4	37182	0.03	1.90	103	196	
	BP24	114.89	1961870	461	43.3	413811	0.06	1.68	216	363	
	BP25	1.01	31799	654	72.4	4146	0.14	1.70	496	843	
	BP26	17.15	352580	3818	66.2	62886	0.32	2.29	1163	2666	
	BP30	16.79	2813209	1035171	85	76143	4.37	22.02	15717	346150	
	BP40	4.28	94960	2199	69	16263	0.24	1.83	866	1586	
	BP49	0.00	994	779	100	0	0.00	0.00	0	0	
	BP8K0	35.11	1859262	323	92.6	126406	0.00	0.00	0	0	
	BP16K0	0.00	211436	550	100	41	0.01	1.69	29	49	
	BP32K	1.96	169945	69235	93.8	12976	1.64	1.67	5912	9853	

Total 4K Buffers	1,747,100	Total Read/Write IO	4,135,831	Total Get Pages	70,661,048
		Overall Sys Hit Ratio	50.93	Total I/Os per second	1,148.84
		Total Updates	2,105,179	Pages per write	4.76

About 7 Gig of memory for the 4K pools

© Responsive Systems 2007

72

GoFurther

With the new 64bit memory, we're now using about 7 gig of memory, and have saved about 500 IO/sec. This system is over 223Million Getpages/Hour, with a greatly reduced, but still high I/O rate.

The getpage rate is more than 10% higher, while the IO rate has decreased almost 50%.

Cost of sequential access

Buffer Pool Tool for DB2 - BP1

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning | **Scan Cost** | Sim Graph Analysis | Sim Cluster Analysis

Overall Collection Time = 1,922,000 System CPU Seconds Cost = 213,991 ←

BP0 CPU Seconds Cost = .004

Object Name	Pool	Sequential GetPages	CPU-Seconds Cost
BTAB22.SBTAB22	BP2	13399737	60.299
BTAB06.SBTAB06	BP2	10979739	49.409
MAFA.MARA	BP1	4485972	20.187
BSAD.BSAD**1	BP3	1948495	8.768
AFKO.AFKO	BP1	1721818	7.748
BTAB20.SBTAB20	BP1	1366864	6.151
BTAB22.TRFCQ.OUTPUT**4	BP3	1359293	6.117
BTAB19.SBTAB19	BP2	956720	4.305
YDRK.VDRK	BP2	847570	3.814
STAB34.SSTAB34	BP1	717459	3.229
STAB33.SSTAB33	BP2	601914	3.069
RFBLG.RFBLG	BP13	614254	2.764
LTAK.LTAK	BP1	572230	2.575
KNA1.KNA1	BP1	504747	2.271
MARC.MARCK	BP1	478663	2.154
VBAK.VBAK	BP1	465070	2.093
NAST.NASTX	BP1	436985	1.966
AFRIU.AFRU	BP6	392158	1.765
MARC.MARC**0	BP3	383730	1.727
LTBP.LTBP**201	BP3	377888	1.7

Sequential access is costing a large amount of CPU cycles per day. Easily more than a million dollars per year.

DB2 V8 and V9

- Exploitation of 64bit memory, and moving more control blocks out of the DBM1 address space
- Access to many gigabytes of memory for pools
 - Doesn't change basic pool tuning methodologies
 - Grouping by Random and Sequential, and then by working set sizes is the proven technique

DB2 Version 9

- Increasing prefetch quantities and using larger page sizes to reduce IO (vpseqt * #Buffers) > 40,000
 - IO remains a huge throughput concern for system scalability
- Supports buffer pool sizes > 5 Gigabytes
- Increased usage of 32K sort file to reduce IO
 - Larger and multiple 32K sort objects

IO remains a primary concern for system scalability, and we these concerns being addressed in every new version of DB2.

DB2 Version 9

- Automatic Pool Size Management – option `autosize=yes`
 - Function integrated with WLM
 - Can increase/decrease pool size by 25% of initial size
 - Increments? *Not much real information available yet*
 - Based on long term trends
 - This is not defined anywhere
 - *Long term performance data is a major problem...*
 - Tries to take the memory from other low activity pools first
 - Based on a random hit ratio – *hit ratios are not valid as a performance metric*
 - Seeming fallacy of this approach – *lack of prediction capability*
 - Bigger is not always better
 - Will WLM reduce it to the original size if the increase does not improve performance?

There are good long term possibilities for this type of approach, and it's obvious that this is just a first cut implementation. It remains to be seen if this provides any real benefit. I suspect it may for small to medium systems at installations where there isn't a lot of DB2 performance knowledge. Large and high performance systems still need real tuning expertise.

Also, remember that the effective way to get good performance is through the proper grouping of objects (Ramos/Samos), and not by throwing memory at few large pools.

DB2 Version 9

- What changes in regard to pool tuning?
 - *Absolutely nothing*
- Methodologies remain the same - Ramos/Samos & Wkset sizes
- There are opportunities for ever larger pools, if you have the memory.... and are sure you get a real benefit
 - Remember, paging is to *DASD*, not expanded memory, so that's 1,000 times slower
 - Just as previous environments, if you start to page, your pool performance may look better statistically, but the users performance is worse...
 - WLM is supposed to manage memory better, and it can, if it's set up properly

The basics of performance tuning have not changed over the last four decades, and certainly won't change over the next decade. CPU, Memory, and IO are the important tuning metrics.

There are other areas where *Performance Prediction* software can help you...

- zIIP engines can save your company a lot of money because their cost is only a bit more than 10% of a CP engine
- Predict utilizations based on your workload
 - Your DRDA workload is expected to grow substantially
 - Your processor is almost out of gas...

**Modeling Case Study
Double DDF Work Models**

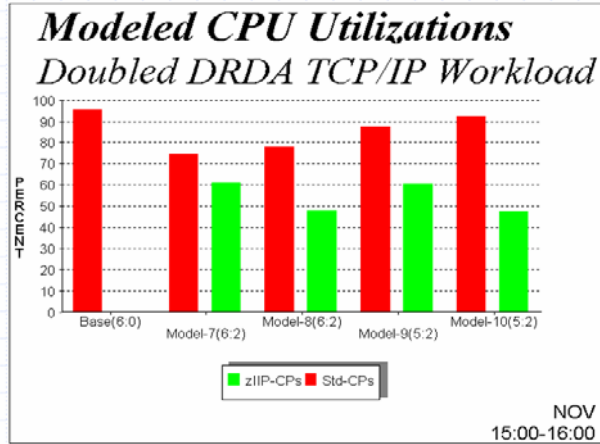
Model ID	Processor	Relative Power	CP Count	zIIP Count	WLM Crossover
Base	2094-706	1	6	N/A	N/A
7	2094-706	1	6	2	No
8	2094-706	1	6	2	Yes
9	2094-705	.85	5	2	No
10	2094-705	.85	5	2	Yes

Presented by Ned Diehl at CMG 2006, Reno, NV.

The Information Systems Manager, Inc.

www.perfman.com

Modeling Case Study
Double DDF Work Models



Copyright © The Information Systems Manager 2007 - All rights reserved.

© Responsive Systems 2007

80

Go further

Presented by Ned Diehl at CMG 2006, Reno, NV.

Information System Manager

www.perfman.com

Summary

- Buffer Pools remain one of the longest tuning levers we have
- Most systems have many performance opportunities
- Some have huge opportunities, and companies can save millions per year with some system and application tuning
 - Need the right tools
 - Need the initiative, and management understanding of the paybacks

While tuning the pools, you usually find many application performance issues, such as sequential scan for Tablespaces and Indexes. Addressing these issues provide the largest CPU cost reductions. Often some of the SP access does not appear as a problem in an online monitor, because it may not be a large cost for one scan. However, if the scan is performed thousands or tens of thousands of times per day... it will jump to the top during your pool performance analysis.

Thank you for attending this presentation

Joel Goldstein

Responsive Systems

joel@responsivesystems.com

www.responsivesystems.com

