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.
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.  \textit{Not necessarily}

• If more buffers reduce the IO rate/second
  • The amount of \textit{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 \textit{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 ratio is the reciprocal of a hit ratio, and is a useless metric. Since most of the world still thinks about a hit ratio, 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

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.

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

<table>
<thead>
<tr>
<th>DATE</th>
<th>TIME</th>
<th>INTV</th>
<th>GTV</th>
<th>SYNC</th>
<th>SEQ</th>
<th>LIST</th>
<th>DYN</th>
<th>IORATE</th>
<th>HIT</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-02-21</td>
<td>00:00:00</td>
<td>1,187,068</td>
<td>43,315</td>
<td>620,296</td>
<td>156,611</td>
<td>203</td>
<td>459</td>
<td>30.47%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>00:30:00</td>
<td>10,913,350</td>
<td>342,248</td>
<td>342,248</td>
<td>342,248</td>
<td>342,248</td>
<td>342,248</td>
<td>54.04%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>01:00:00</td>
<td>671,955</td>
<td>31,854</td>
<td>173,032</td>
<td>49,720</td>
<td>933</td>
<td>264</td>
<td>29.24%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>01:30:00</td>
<td>741,700</td>
<td>47,202</td>
<td>235,841</td>
<td>2,948</td>
<td>95</td>
<td>281</td>
<td>31.05%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>09:30:00</td>
<td>4,105,758</td>
<td>224,423</td>
<td>2,961,328</td>
<td>12,364</td>
<td>69,213</td>
<td>1,760</td>
<td>22.85%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>10:00:00</td>
<td>4,232,240</td>
<td>100,715</td>
<td>3,086,771</td>
<td>9,277</td>
<td>64,952</td>
<td>1,813</td>
<td>22.88%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>10:30:00</td>
<td>3,127,959</td>
<td>70,646</td>
<td>2,462,744</td>
<td>18,310</td>
<td>58,303</td>
<td>1,464</td>
<td>16.08%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>11:00:00</td>
<td>3,890,741</td>
<td>112,328</td>
<td>2,520,929</td>
<td>9,911</td>
<td>51,934</td>
<td>1,497</td>
<td>10.73%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>11:30:00</td>
<td>4,011,848</td>
<td>127,173</td>
<td>2,334,283</td>
<td>11,760</td>
<td>76,235</td>
<td>1,443</td>
<td>35.20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>12:00:00</td>
<td>4,580,930</td>
<td>109,463</td>
<td>2,339,109</td>
<td>639,288</td>
<td>68,607</td>
<td>1,754</td>
<td>31.05%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007-02-21</td>
<td>12:30:00</td>
<td>6,150,020</td>
<td>188,803</td>
<td>3,202,333</td>
<td>11,852</td>
<td>64,002</td>
<td>1,926</td>
<td>43.61%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 I/Os.
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.
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 $\text{vpseqt} \times \#\text{buffers} > 40,000$, 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…
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 if there aren’t any hits for SPTH and DWTH.
The buffer pool can be considered to have three general areas, as shown above. The leftmost area, for pages that have been updated, but not yet written out, shows as the pages-in-use in buffer pool statistical information, and the online monitors. It’s important to understand that ALL pages in pool always exist, are allocated, and contain data. The Least Recently Used (LRU) queues affect the length of time any given page remains in the pool, available for re-use. As pool sizes increase, there are really multiple LRU queues in use for each general type of access – random or sequential.
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.
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
QW0198G DS H DATABASE ID
QW0198GB DS H PAGESET GRID
QW0198GP DS H BUFFERPOOL ID
QW0198FC DS C FUNCTION CODE

**..............QW0198FC CONSTANTS.......................**
QW0198GR EQU C'G' GET PAGE REQUEST
QW0198SR EQU C'S' SET WRITE INTENT REQUEST
QW0198RP EQU C'R' RELEASE PAGE REQUEST

**..............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)
QW0198SL EQU C'L' RIDLIST ACCESS (GET PAGE)
QW0198SH EQU C'N' STANDARD REQUEST (RELEASE PAGE)
QW0198SH EQU C'S' DESTRUCTIVE REQUEST (RELEASE PAGE)
QW0198ML EQU C'L' MRU SCHEME APPLIED (RELEASE PAGE)
QW0198AC DS A ACE ADDRESS

**..............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

These apply only to Reads, not Writes

The field reference data and constant values are critical pieces of information to interpret the records properly.
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 *
* 50-89)
QW0006PN DS XL3  FIRST PAGE NUMBER TO BE READ (for *
* non large table space)
QW0006F DS C  FLAG FOR TYPE OF READ
QW0006PDS EQU C'S'  SEQUENTIAL PREFETCH REQUEST
QW0006LDLS EQU C'L'  LIST PREFETCH REQUEST
QW0006PDLS EQU C'D'  DYNAMIC SEQUENTIAL PREFETCH REQUEST
QW0006F  DS  C  READ REQUEST
QW0006AC DS F  ACE TOKEN OF REQUESTOR
QW0006PG DS F  FIRST PAGE NUMBER TO BE READ
QW0006PS DS C  FLAG FOR TYPE OF TABLE SPACE
QW0006PSL EQU C'N'  NON LARGE TABLE SPACE
QW0006PS1 EQU C'N'  NON-EA LARGE TABLE SPACE
QW0006PS2 EQU C'V'  EA-LARGE TABLE SPACE
QW0006PT DS F  Partition number or 0 if non-partitioned

This let’s us see usage by partition

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 I/O 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 I/O 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 firefighting 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.
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 its own next analysis step and path…
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.
## Object Level

<table>
<thead>
<tr>
<th>Object Level</th>
<th>Trend</th>
<th>Sample T1</th>
<th>Sample T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKPAI1.SYSADM.DSN14K01</td>
<td><img src="Trend1" alt="Trend" /></td>
<td>0.00028</td>
<td>0.00022</td>
</tr>
<tr>
<td>NCMLR2B8</td>
<td><img src="Trend2" alt="Trend" /></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WORKPAI1.SYSADM.DSN14K02</td>
<td><img src="Trend3" alt="Trend" /></td>
<td>0.00003</td>
<td>0.00022</td>
</tr>
<tr>
<td>WORKPAI1.SYSADM.DSN14K03</td>
<td><img src="Trend4" alt="Trend" /></td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>WORKPAI1.SYSADM.DSN14K05</td>
<td><img src="Trend5" alt="Trend" /></td>
<td>0.00055</td>
<td>0.00000</td>
</tr>
<tr>
<td>WORKPAI1.SYSADM.DSN14K06</td>
<td><img src="Trend6" alt="Trend" /></td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>WORKPAI1.SYSADM.DSN14K07</td>
<td><img src="Trend7" alt="Trend" /></td>
<td>0.00027</td>
<td>0.00044</td>
</tr>
<tr>
<td>WORKPAI1.SYSADM.DSN14K08</td>
<td><img src="Trend8" alt="Trend" /></td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

Looking for changes in performance & activity

IO Rates/Second
Looking for changes in performance & activity
Individual metric level

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>RD_RATE</th>
<th>GET_PAGES_RATE</th>
<th>WHO_RATE</th>
<th>WRITE_PAGES_RATE</th>
<th>POOL</th>
<th>POOL_PERCENTAGE</th>
<th>SYSTEM_PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKSPACE MYSALES WSN0404</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSPACE MYSALES WSN0402</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSPACE MYSALES WSN0409</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSPACE MYSALES WSN0418</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSPACE MYSALES WSN0426</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WORKSPACE MYSALES WSN0430</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Responsive Systems 2007
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 !!

These I/O rate per sec savings, up through 2,500 per second, have been achieved by clients.
## Buffer Pool Performance Data - Red Flags

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

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

<table>
<thead>
<tr>
<th>Buffer Pool Data</th>
<th>Quantity (Units)</th>
<th>/SECOND</th>
<th>/THREAD</th>
<th>/COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>31.59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GETPAGE REQUEST</td>
<td>179.4M</td>
<td>2083.30</td>
<td>250.37</td>
<td>70.40</td>
</tr>
<tr>
<td>GETPAGE REQUEST-SEQUENTIAL</td>
<td>130.1M</td>
<td>1510.58</td>
<td>181.54</td>
<td>51.05</td>
</tr>
<tr>
<td>GETPAGE REQUEST-RANDOM</td>
<td>49309.8K</td>
<td>572.71</td>
<td>68.83</td>
<td>19.35</td>
</tr>
<tr>
<td>SYNCHRONOUS READS</td>
<td>5285.0K</td>
<td>61.38</td>
<td>7.38</td>
<td>2.07</td>
</tr>
<tr>
<td>SYNCHRON. READS-SEQUENTIAL</td>
<td>1797.4K</td>
<td>20.88</td>
<td>2.51</td>
<td>0.71</td>
</tr>
<tr>
<td>SYNCHRON. READS-RANDOM</td>
<td>3487.7K</td>
<td>40.51</td>
<td>4.87</td>
<td>1.37</td>
</tr>
<tr>
<td>GETPAGE PER SYN. READ-RANDOM</td>
<td>14.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQUENTIAL PREFETCH REQUEST</td>
<td>3856.3K</td>
<td>44.79</td>
<td>5.38</td>
<td>1.51</td>
</tr>
<tr>
<td>SEQUENTIAL PREFETCH READS</td>
<td>3684.0K</td>
<td>42.79</td>
<td>5.14</td>
<td>1.45</td>
</tr>
<tr>
<td>PAGES READ VIA SEQ. PREFETCH</td>
<td>109.6M</td>
<td>1273.45</td>
<td>153.05</td>
<td>43.04</td>
</tr>
<tr>
<td>S.PRF. PAGES READ/S.PRF. READ</td>
<td>29.76</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIST PREFETCH REQUESTS</td>
<td>1328.4K</td>
<td>15.43</td>
<td>1.85</td>
<td>0.52</td>
</tr>
<tr>
<td>LIST PREFETCH READS</td>
<td>392.2K</td>
<td>4.56</td>
<td>0.55</td>
<td>0.15</td>
</tr>
<tr>
<td>PAGES READ VIA LIST PREFETCH</td>
<td>2049.5K</td>
<td>23.80</td>
<td>2.86</td>
<td>0.80</td>
</tr>
<tr>
<td>L.PRF. PAGES READ/L.PRF. READ</td>
<td>5.23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMIC PREFETCH REQUESTED</td>
<td>181.7K</td>
<td>2.29</td>
<td>0.28</td>
<td>0.08</td>
</tr>
<tr>
<td>DYNAMIC PREFETCH READS</td>
<td>184.4K</td>
<td>2.14</td>
<td>0.26</td>
<td>0.07</td>
</tr>
<tr>
<td>PAGES READ VIA DYN. PREFETCH</td>
<td>5726.0K</td>
<td>66.51</td>
<td>7.99</td>
<td>2.25</td>
</tr>
<tr>
<td>D.PRF. PAGES READ/D.PRF. READ</td>
<td>31.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREF.DISABLED-NO BUFFER</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PREF.DISABLED-NO READ ENG</td>
<td>74.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PAGE-INS REQUIRED FOR READ</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

© Responsive Systems 2007
### Buffer Pool Performance Data - Red Flags

<table>
<thead>
<tr>
<th>BP5 WRITE OPERATIONS</th>
<th>QUANTITY</th>
<th>/SECOND</th>
<th>/THREAD</th>
<th>/COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER UPDATES</td>
<td>18460.7K</td>
<td>214.41</td>
<td>25.77</td>
<td>9.39</td>
</tr>
<tr>
<td>PAGES WRITTEN</td>
<td>2220.6K</td>
<td>25.79</td>
<td>3.10</td>
<td>1.13</td>
</tr>
<tr>
<td>BUFF.UPDATES/PAGES WRITTEN</td>
<td>8.31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNCHRONOUS WRITES</td>
<td>25517.00</td>
<td>0.30</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>ASYNCHRONOUS WRITE</td>
<td>1063.6K</td>
<td>12.35</td>
<td>1.48</td>
<td>0.54</td>
</tr>
<tr>
<td>PAGES WRITTEN PER WRITE I/O</td>
<td>2.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HORIZ.DEF.WRITE THRESHOLD</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>VERTI.DEF.WRITE THRESHOLD</td>
<td>451.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DM THRESHOLD</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>WRITE ENGINE NOT AVAILABLE</td>
<td>85.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>PAGE-INS REQUIRED FOR WRITE</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

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
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.
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.
The heaviest SP index. Objects with the green bars are partitioned.
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.
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?
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 – this is why…

The working set sizes of the three objects in the first cluster, at the original pool size of 101,200.
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
Well, only one object, and 20,000 buffers.
BP22 with the highest IO rate object

Really low DP...

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 ½ SP access. Object is partitioned.
BP22 with the highest IO rate object

<table>
<thead>
<tr>
<th>Partition</th>
<th>Object</th>
<th>Seq Acc</th>
<th>Red Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>DBROP.DBFDBAS</td>
<td>115614</td>
<td>150603</td>
</tr>
<tr>
<td>5</td>
<td>DBROP.DBFDBAS</td>
<td>97233</td>
<td>133256</td>
</tr>
<tr>
<td>6</td>
<td>DBROP.DBFDBAS</td>
<td>131558</td>
<td>62062</td>
</tr>
<tr>
<td>7</td>
<td>DBROP.DBFDBAS</td>
<td>111041</td>
<td>88227</td>
</tr>
<tr>
<td>8</td>
<td>DBROP.DBFDBAS</td>
<td>71848</td>
<td>36833</td>
</tr>
<tr>
<td>9</td>
<td>DBROP.DBFDBAS</td>
<td>69959</td>
<td>63276</td>
</tr>
<tr>
<td>10</td>
<td>DBROP.DBFDBAS</td>
<td>54724</td>
<td>63105</td>
</tr>
<tr>
<td>11</td>
<td>DBROP.DBFDBAS</td>
<td>29864</td>
<td>70119</td>
</tr>
<tr>
<td>12</td>
<td>DBROP.DBFDBAS</td>
<td>7244</td>
<td>35182</td>
</tr>
</tbody>
</table>

Differing access percentages for the partitions

Some are mostly SP, some are mostly random.
Doubling the pool size saves more than 45 IO/sec, because the randomly accessed pages can remain in the pool longer.

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

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

Now the 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%.*
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.
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 dataspaces.
Using the new 64bit memory

With the new 64bit memory, we’re now using about 7 gig of memory, and have saved about 500 I/O/sec. This 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%.*
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 \((\text{vpseqt} \times \#\text{Buffers}) > 40,000\)
  - 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 anyplace
        - *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.
• 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…
<table>
<thead>
<tr>
<th>Model ID</th>
<th>Processor</th>
<th>Relative Power</th>
<th>CP Count</th>
<th>zIIP Count</th>
<th>WLM Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>2094-706</td>
<td>1</td>
<td>6</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>7</td>
<td>2094-706</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>2094-706</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>2094-705</td>
<td>.85</td>
<td>5</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>2094-705</td>
<td>.85</td>
<td>5</td>
<td>2</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Presented by Ned Diehl at CMG 2006, Reno, NV.
The Information Systems Manager, Inc.
www.perfman.com
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