



# ***The Buffer Pool Hit Ratio is Dead !!***

**Joel Goldstein  
Responsive Systems**

**(732) 972-1261**

**Email: [joel@responsivesystems.com](mailto:joel@responsivesystems.com)**

**Web Site  
[www.responsivesystems.com](http://www.responsivesystems.com)**

## **Key Presentation Points**

- Different types of hit ratios – calculations
- Relating them to system and application performance
- Higher is better... performance examples & graphs
- Hit ratio compared to the I/O Rate/Second
- Bigger pools do not always mean better performance
- Summaries and conclusions

History is important – you have to know where you came from, and where you are, to be able to move forward. We need to understand what hit ratios mean relative to performance – and then compare them to the I/O rate to see the difference. This presentation will illustrate that an I/O rate is meaningful and measurable, and that a hit ratio is not necessarily relevant to performance.

With 64bit memory, and Gigabytes of memory, is it still worth tuning, or should you just throw memory at the pools? Data shows that more memory often does not improve performance unless you can make the pool(s) larger than the data – and this still can't happen in most systems.

## **Background**

- **Database Management Systems**
- **Physical I/O – or keep Data in memory**
  - Memory is fast, I/O is slow
- **Both were expensive**
  - While memory is much less expensive every year, and CACHE memory is even less than mainframe memory, DASD is really cheap now
- **Online systems**
  - Performance became important
  - Users were waiting for a response

I/O has always been a performance problem and limiting factor for large applications. This has not changed today, and the goal of DB2 is to eliminate/reduce I/Os by keeping data in memory. As we look ahead to DB2 V8, it will allow up to a TB of memory specification for the buffer pools. Of course, this depends upon your processor having this much or more available.

## *Access to data*

- All access was random, in theory
  - Of course you could still read the whole database
- Processing was Record oriented, not Set oriented
  - Hierarchical
  - Network
- There was no anticipatory reading of data (prefetch)

When all access was (is) random, a hit ratio had some relevance to performance – but was (and is) not measurable.

## **Measuring Performance**

- **Transaction elapsed times**
  - Data Transmission *4800/9600 Baud vs. T1 or better today*
  - Input/Output message queuing to disk
  - CPU
  - I/O wait
- **I/O avoidance**
  - Keep data in memory
  - Buffer Pools
- **Processors did not have a lot of memory**
  - Systems were smaller, more efficient – rigorous design
  - Buffer Pools were small



Many of the major factors of transaction response time have changed over the years. In the early 70's, it was common to see data transmission times for (basically) text messages of 1600 bytes take 3-6 Secs. I/O times took more than an order of magnitude longer than the averages on well running dasd subsystems today. Yet, there are many poorly performing dasd subsystems in the world today.

On the processor CPU side, the average desktop PC is approaches the mainframe power of a decade past.

The same perspectives exist for memory. Most desktop machines today start at 256meg.

Current mainframes can have 64 Gig of memory, and this number will increase rapidly now that we have 64bit operating systems.

We have always filled (and often exceeded) available memory - and this won't change within this decade.

## *Measuring DB2 Pool Performance*

- **The first measurement at the system level**
  - Hit ratio
  
- **Application hit ratio**
  - Ignores any pages/blocks prefetched in advance
  - Is only useful to measure application delay
  - Useless as a system tuning metric
  
- **System hit ratio**
  - Measures pool efficiency by factoring in pages/blocks prefetched in advance

When all access was random, and there was no concept of prefetching multiple blocks of data, determined by the DBMS software.

So measurements were simpler. Of course, if we knew that data would be accessed sequentially, we could buffer some data in memory outside a program using a `bufno=x` parameter on the JCL - but this still did not help data accessed by a DBMS.

## *Measuring Pool Performance*

- **Application hit ratio**  
 $(GP - \text{Read I/Os})/GP$
- **System hit ratio**  
 $(GP - \text{Sum of all PagesRead})/GP$
- **Convergence**
  - When access is all random, and only Synch I/O – the ratios are the same

The initial purpose of the first Hit Ratio we used for DB2 effectively measured only application delay, and did not really address the activity taking place in the buffer pools. Then we came to the System Hit Ratio that factored in the effect of pages read into the pool by prefetch reads.

When access is all random, so there is no sequential prefetch, and dynamic prefetch is not being used by the buffer manager, then the application and system hit ratios will be identical.

## Measuring Pool Performance

### ● Divergence

- When access is sequential with heavy prefetch
  - » The application hit ratio is always high, 95-98%
    - One I/O is 32 pages (97% hit)
    - » The system hit ratio is always low
      - A complete scan, hit ratio is Zero
- Dynamic prefetch reduces the system hit ratio, *and it may be negative*



Let's take the case of a pool where all objects are accessed using sequential prefetch. The application hit ratio will always be high, at least 97%, because 32 pages are read with a single I/O. A prefetch "request" can read anywhere from 0 to 32 pages, because the buffer manager knows which (if any pages) are already in the pool.

So - if all 32 pages are read, the application hit ratio is 97%; however, the system hit ratio is ZERO.

Additionally, it is quite common that many pages read by dynamic prefetch are never actually accessed by the application, so

No getpage requests are ever issued for them. This drives the system hit ratio down, and it can actually have a negative number.



## *Pool Performance*

- **Is rarely linear**
  - More memory does not always improve performance
  - Small pool increases often don't show any improvement
  - Sometimes additional “smallish” increases can provide substantial improvements
  - Many times large increases do not help
    - » Sometimes they do....
    - » It may depend on what “large” means to you...
      - A number, percentage, or it depends...?



We will see from data later in this presentation that pool performance is not linear. Doubling the pool size does not double the hit ratio, or cut the I/O rate in half. We will always reach a point of diminishing returns, when adding buffers to a pool.

Now, I realize that some of the above items seem contradictory, and will explain them in more detail during the presentation.

The data shown in future slides will also illustrate all the above points.

## ***As you Start to Tune Pool Performance***

- **You will find many interesting application performance problems that your online monitors may not be highlighting**
  - Objects that are used in a very different manner than the application designers expected
  - Tablespaces with lots of sequential scan
  - Indexes with sequential scan

***Burn those CPU engines.....***



As start to analyze your pool performance, and the performance of the objects in each pool, you often find objects that are not being accessed as you expect they should be. Perhaps an Index with heavy scan, or a tablespace that is monopolizing a pool - and this can be either sequential, or a very large random object.

The key to performance, is grouping objects based on access type, and working set size.

# **A 60% increase provides nothing- 160% Does**

Component: >>> IMS Buffer Pool Simulation <<<

Results of Simulation by Object, ~~SPool~~ ~~GetB~~ total.....1,098,780

Results for Object.....IMSVS.BET.WKLD.DTHSD0D1

Object ~~GetB~~ total.....74,980 ( 6.8% of SP ~~GetB~~)

Pool Size	GetB used	Num. of Hits	Hit Ratio	Avg. WSet	Max. WSet
500	73,961	1450	2.0 %	7	100
600	73,465	1458	2.0 %	12	159
700	73,459	1469	2.0 %	25	170
800	73,434	1499	2.0 %	36	196
1300	72,089	7976	11.1 %	96	293

I/O rate decreased 62 > 56.9

**IMS generally doesn't use large pools compared to DB2**

doubled

tripled

Here I'm using an IMS example to show that even large percentage increases may not show any improvement until you pass a certain "critical" threshold for the amount of data that can remain in a pool. IMS systems do not use the amount of buffer pool sizes that we commonly use for DB2.

## Another IMS example

```
VSAM Buffer SubPool simulation Results
Pool Name.....TST1
Pool Type.....VSAM
SubPool Buffer Size.....16,384
Lower Block Size.....8,192
Number of Objects accessed.....13
Total number of Get Block.....150,169

SPool Size   GetB used   Num. of Hits   Hit Ratio   Elapsed Time
    50       150,119           91         0.0 %      00:08:15
                . Missing lines .
    370       149,799          751        0.5 %      00:08:12
    450       149,692       45,356      30.3 %      00:08:11
    530       149,577       59,991      40.1 %      00:08:10
    610       149,462       74,279      49.7 %      00:08:10

SPool Size   Blocks Read
    50        303.1 /S
                . Missing lines .
    370        302.9 /S
    450        212.5 /S
    530        182.6 /S
    610        153.4 /S
```

← No improvement until the pool is 9 times the starting size

Here again, there is no real performance improvement until the number of buffers is increased by a factor of 9.

Increasing from 370 to 450 buffers reduced the I/O rate by 90/Sec.  
However, subsequent increases save 30 I/Os/Sec,  
then 29 I/Os/Sec, so the reduction starts to taper off....

## **Pool Performance**

- **Eliminating I/O reduces CPU costs**
  - The CPU cost for an I/O does not change when you improve DASD performance
- **Throughput and Productivity Improvements**
- **Eliminating I/O**
  - Usually will not reduce the processor “busy rate”
  - It may increase the processor “busy rate”
    - » *But this is good*



The huge amounts of cache memory on today's dasd control units are essentially an external extension of the DB2 buffer pools, provided up to 512 Gig of memory on some models.

However, the CPU cost of I/O remains the same whether the data is in the cache, or has to be read from the disk subsystem.

The only difference is the elapsed time. Finding the data in cache can provide a response in 1 Ms, but a cache miss often takes more than 20 Ms.

## *Pool Performance*

- **Larger pools – more memory**
  - Do not always improve performance
  - May increase system paging and hurt performance
    - » Might look better statistically, but response time can be worse
- **Memory is a *system* resource**
  - DB2 is not the *only* sub-system on your processor
  - You will run out of memory on a 64 bit machine too...



Just throwing more memory at pools does not always provide better performance, and I will show you data to illustrate this.

Over allocating overall memory resources may initially look like you are getting better performance because the I/O rate drops, but if the system paging rate increases too much, overall response time will be worse.

## System Summary

Buffer Pool Tool for DB2 - BP2

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

**Collection**

Date: 2002-03-28  
Time: 15:22:06  
Elapsed Time: 00:58:09

**System Info**

System: AS01  
Sub System: FP01  
Db2 Version: 6.1

Pool	I/O	Get Pages	Updates	Hit Ratio	I/O Sec	Pages/Write
BP0	627	8085	215	92.8	0.18	2.24
BP1	2498	285414	246080	98.7	0.72	15.84
BP2	910275	6305745	165309	35.7	260.90	1.86
BP3	342429	12283232	710194	85	98.15	2.57
BP4	69097	1235173	305302	96.7	19.80	7.03
BP5	952	180295	93559	99.5	0.27	12.72
BP6	80393	264913	6663	20.7	23.04	5.28
BP7	30405	175029	16726	32.3	8.71	2.79
BP10	30909	542523	14871	83.9	8.86	2.12
BP11	48690	124815	4987	-38.4	13.96	5.70
BP12	4083	409667	18273	99	1.17	5.14
BP14	81848	217214	4486	-76.9	23.46	4.47
BP15	6006	3152388	0	99.7	1.72	0.00

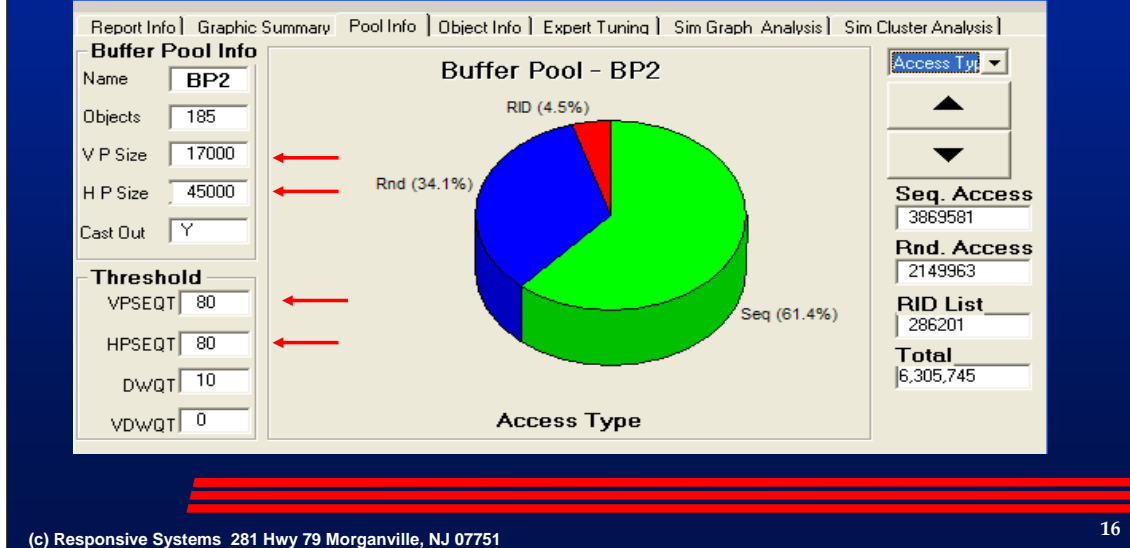
Total Read/Write IO: 1,623,452    Total Get Pages: 25,436,064  
Overall Sys Hit Ratio: 72.64    Total I/Os per second: 465.31  
Total Updates: 1,595,929    Pages per write: 2.47

Too many pools    The problem

Use the Eyeball method of problem analysis.... What's a big value compared to everything else?

Note that BP3, that has a high 85% hit ratio, higher than BP10, shows a high I/O rate.

## Desired Pool Access?



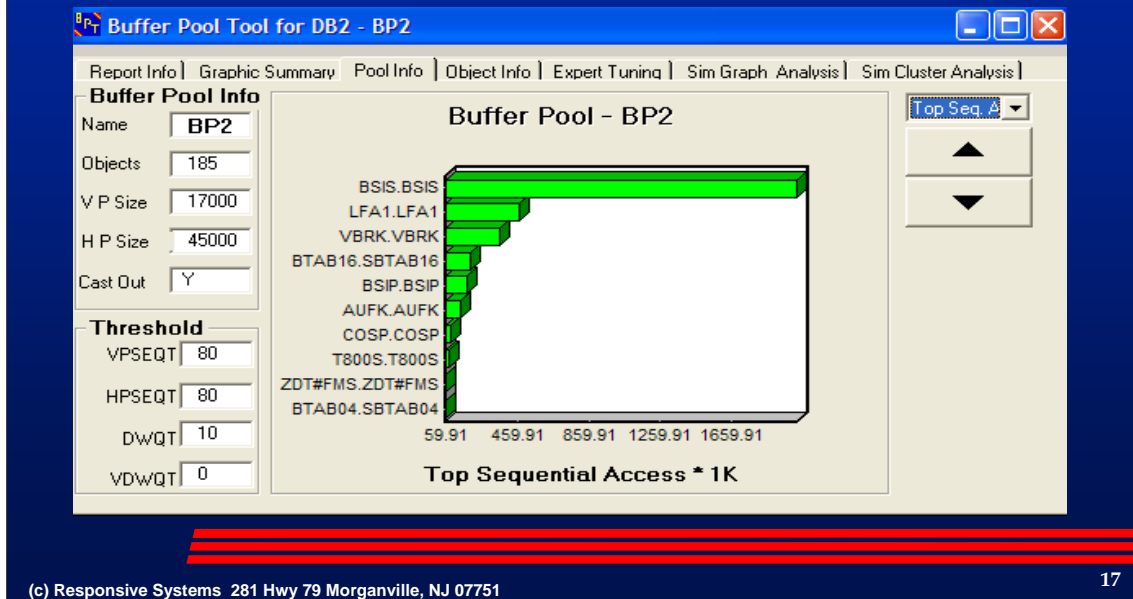
What is the “desired” access we want to see for this pool? Is this what we expected?

The pool has a large number of buffers, yet it has a very high I/O rate.

What is causing the I/O rate to be so high, and what can we do about it?



## What is Sequential?



Since we saw very large amounts of SP, we want to see which objects are monopolizing the pool with this type of access.

Lowering the vpseqt and hpseqt will not reduce the amount of SP activity, but will prevent it from forcing as many random

Pages out of the pool.

## How is it Accessed?

BSIS is 1/3 of the overall pool GP

Section	Metric	Value
Buffer Pool Info	Name	BP2
	Objects	185
	V P Size	17000
	H P Size	45000
Threshold	VPSEQT	80
	HPSEQT	80
	DWQNT	10
	VDWQNT	0
Total Get Pages	Total Get Pages	2034351
	Get Page Rand	6182
Get Page Seq	Get Page Seq	2025502
	Get Page RidList	2667
Avg Synchron IO (ms)	Avg Synchron IO (ms)	4.00
	Avg SP IO (Seq Pref)	18.00
Pages Read Sync	Pages Read Sync	518836 ???
	Pages Read Seqpr	1669674
Pages Read Listpr	Pages Read Listpr	1595
	Pages Read Dynpr	0
App Hit Ratio	App Hit Ratio	73.2
	System Hit Ratio	-7.7
Read IO Rate/sec	Read IO Rate/sec	164.62
	Pages / Write	1.62

Object BSIS has, by far, the greatest amount of SP activity. Now what else can we find out about it?

This object is also causing 148.7 I/Os/Sec using Synchron I/O. Why are we seeing this?

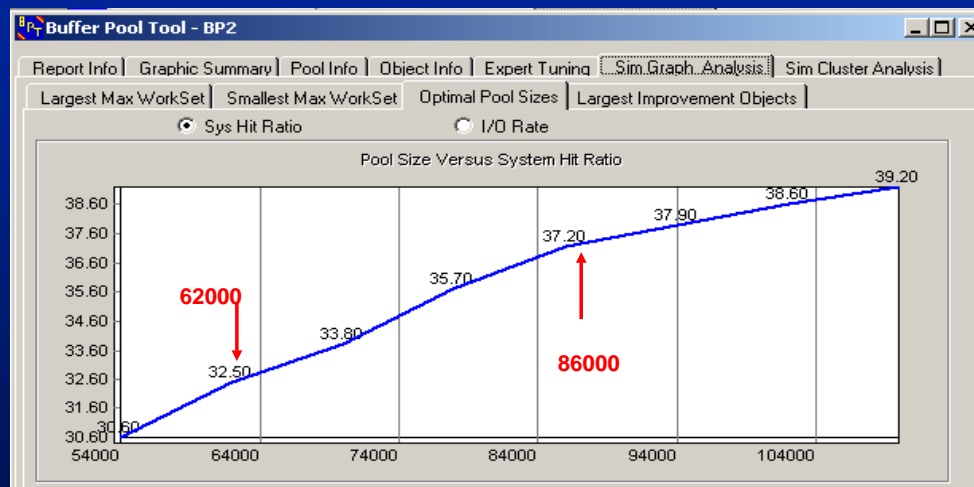
The number of SP I/Os is about 10% of the number of Synchron I/Os.

## *How can we Eliminate I/Os?*

- **Give the pool more buffers – memory**
  - How much will we need?
    - » How much can we gain?
  - Is it available on the system?
- **Move objects into different pools**
- **Create a useful Index on the Object**

There are three options to reduce the heavy I/O activity against BP2.  
Creating a useful index on the object will provide a large reduction in the number of pages referenced, and the greatest overall performance improvement – and cpu reduction. Unfortunately, this may not always be possible.

## How much is 5% ?



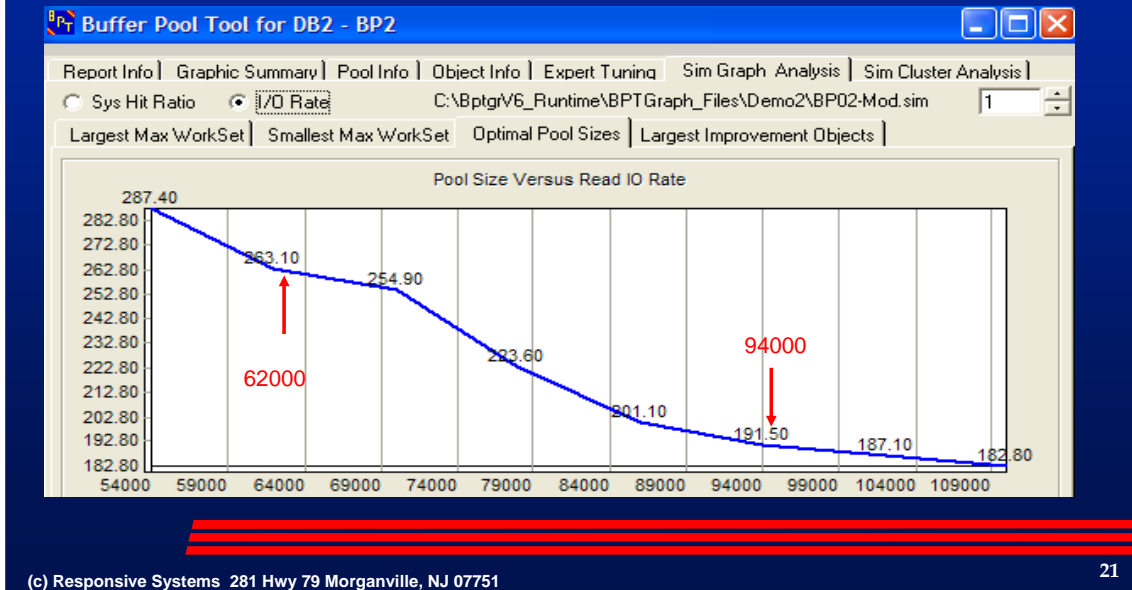
The current pool has 62,000 buffers in total. What can we gain by throwing more memory at it?

If we look at the Hit Ratio, an additional 24,000 buffers (96 Meg) gets us 5%, but the slope of the curve flattens after that, so even more memory provides diminishing returns.

Just what does 5% really mean? Can we convert this into an elapsed time saving, or CPU saving?

The answer is - **no**.

## 72 I/Os a Second



Looking at the I/O side, we can see the saving of I/Os/Sec.

Since the duration of the data we were looking at was 58 Mins and 9 Sec, this is 3,489 Sec, times 72 I/Os = 251,208 I/Os.

Object BSIS showed an average Synch I/O time of 4 Ms, and this is most of where our saving would be.

So if this might be representative of of a batch jobstream, this will reduce the elapsed time by almost 17 Minutes.

## **Cause of the Pain?**

The screenshot shows the Buffer Pool Tool - BP2 interface. It includes a menu bar with options: Report Info, Graphic Summary, Pool Info, Object Info, Expert Tuning, Sim Graph Analysis, and Sim Cluster Analysis. The main area is divided into sections: Pool Usage Intent (with radio buttons for Sequential and Random), Pool Size (set to 54000), and Cluster Radius (set to 2.7). Below these is the Cluster Info section, which contains two tables: Clusters and Objects.

Object	Smallest Max WS	Largest Max WS
1	31028	42224
2	5109	10592
3	10	3327

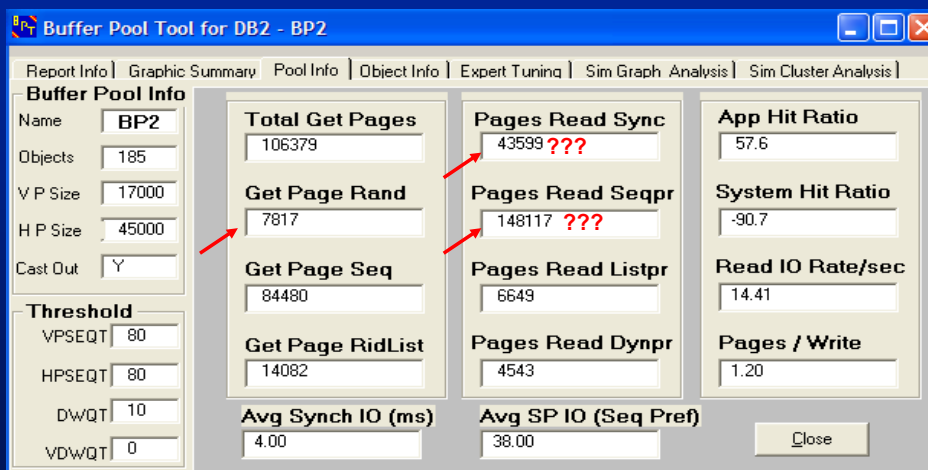
Type	Object	Max Work Set
T	BSIS.BSIS	42224
T	COSP.COSP	31028

Maybe we need to look at COSP as well...

Each uses > 50% of the pool at some point

We can't stop with BSIS, let's look at the number two object as well.

## **The other large sequential**



COSP is not a major impact, less than 2% of GP, control with vpseqt...

Again we see mostly SP, but a high number of Synch I/Os.  
But, we issued 106K getpage requests and read in 198K pages.

# Object growth

The screenshot shows the Buffer Pool Tool - BP2 interface. The Pool Size is set to 78000 and the Cluster Radius is 1.8. The Pool Usage Intent is set to Sequential. The Cluster Info section contains two tables: Clusters and Objects.

Object	Smallest Max WS	Largest Max WS
1	47689	59154
2	17803	17803
3	6521	15129
4	10	5248

Type	Object	Max Work Set
T	BSIS.BSIS	59154
T	COSP.COSP	47689

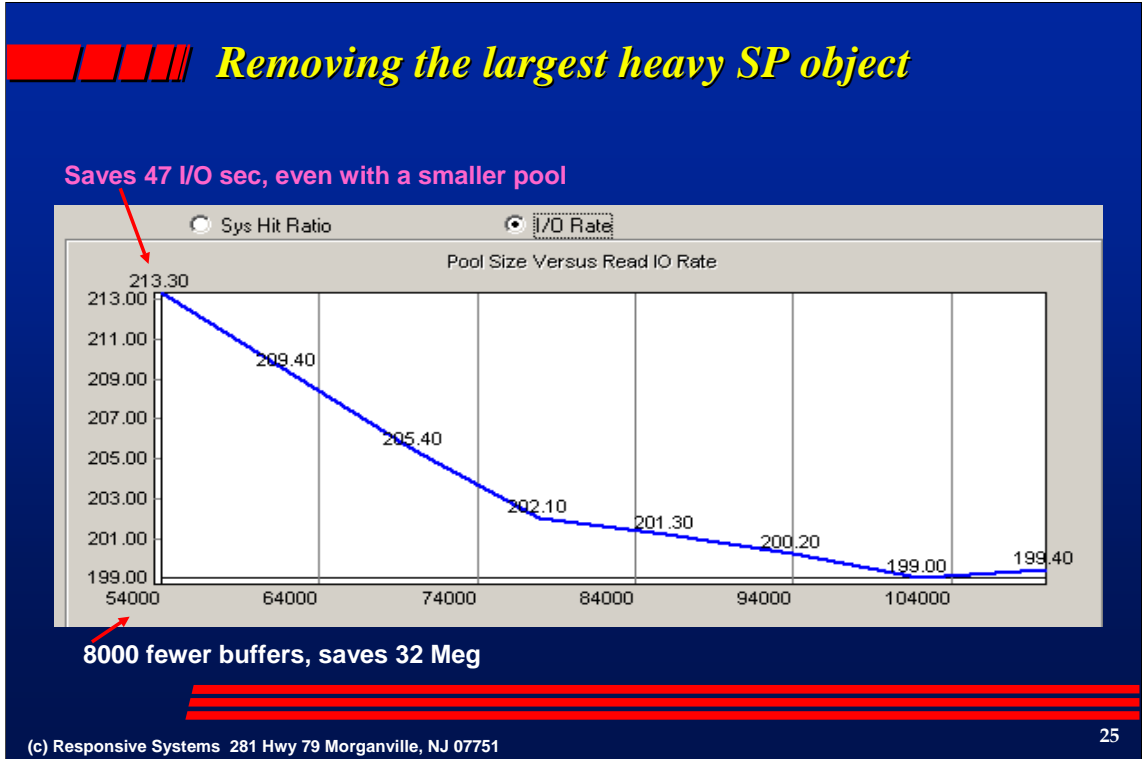
They both get much larger if the pool size is increased

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 24

Increasing the number of buffers in the pool from 54,000 to 78,000, and the objects still monopolize the pool.

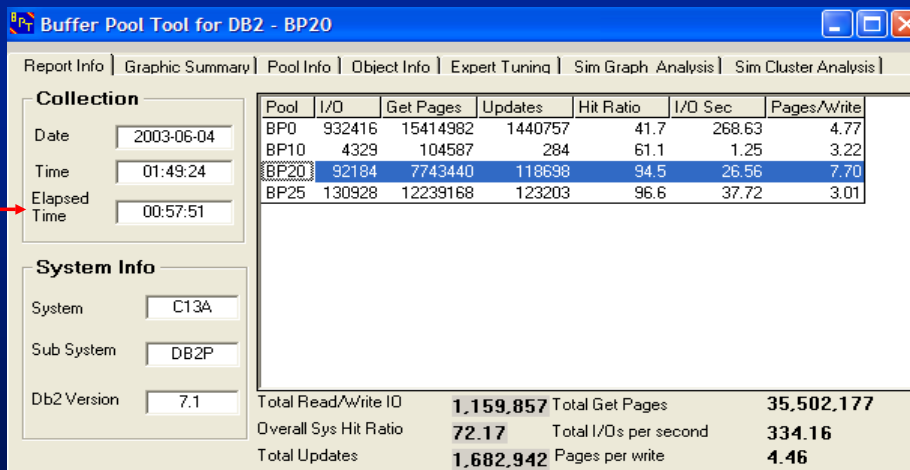
So just giving the pool a lot more memory will not help performance a lot, since these objects will continue to monopolize the pool resources.





Removing object BSIS can save 42 I/Os per second against the other mostly random objects, while saving 32 meg of memory by reducing the pool size by 8,000 buffers (if there are system memory constraints). Object BSIS can be placed into another pool with sequential objects, or a new pool for sequential access. Objects that are heavily scanned all the time do not need large memory allocations (unless you can make the pool larger than the entire object).

## *A Different DB2 System*



The screenshot shows the 'Buffer Pool Tool for DB2 - BP20' application window. It features a menu bar with options: Report Info, Graphic Summary, Pool Info, Object Info, Expert Tuning, Sim Graph, Analysis, and Sim Cluster Analysis. The main interface is divided into several sections:

- Collection:** Includes input fields for Date (2003-06-04), Time (01:49:24), and Elapsed Time (00:57:51). A red arrow points to the Elapsed Time field.
- System Info:** Includes input fields for System (C13A), Sub System (DB2P), and Db2 Version (7.1).
- Table:** A table with columns: Pool, I/O, Get Pages, Updates, Hit Ratio, I/O Sec, and Pages/Write. The data is as follows:

Pool	I/O	Get Pages	Updates	Hit Ratio	I/O Sec	Pages/Write
BP0	932416	15414982	1440757	41.7	268.63	4.77
BP10	4329	104587	284	61.1	1.25	3.22
BP20	92184	7743440	118638	94.5	26.56	7.70
BP25	130928	12239168	123203	96.6	37.72	3.01
- Summary:** A summary section at the bottom right with the following data:

Total Read/Write IO	<b>1,159,857</b>	Total Get Pages	<b>35,502,177</b>
Overall Sys Hit Ratio	<b>72.17</b>	Total I/Os per second	<b>334.16</b>
Total Updates	<b>1,682,942</b>	Pages per write	<b>4.46</b>

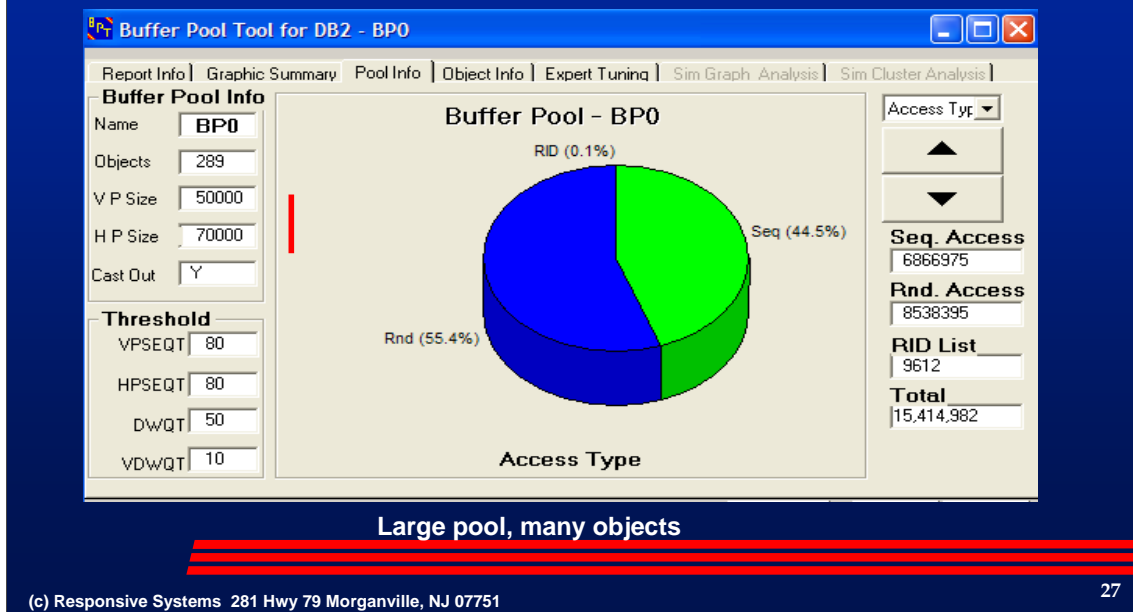
Let's look at data from a different DB2 subsystem.

This system is not using enough pools, and has not separated the objects out of BP0.

BP0 is generating 80% of the entire system I/O.

This is probably part of an early morning batch processing scenario.

## **Different System**

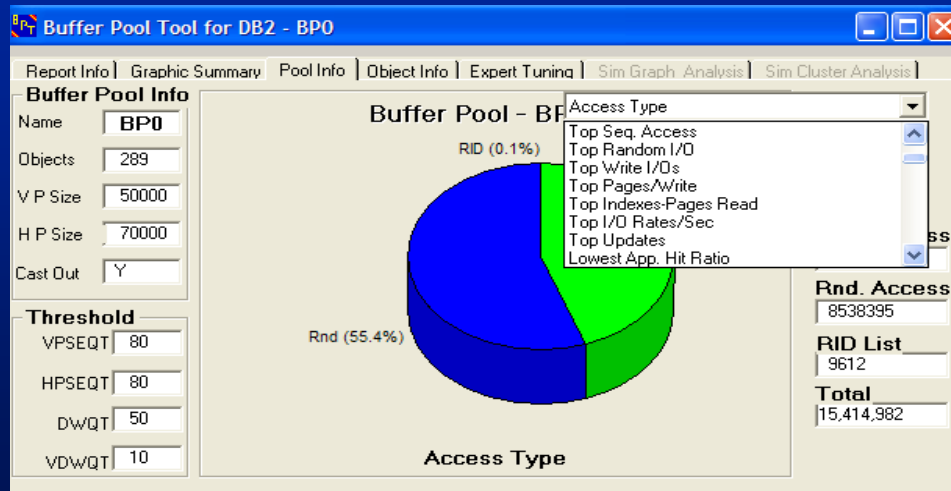


**Large pool, many objects**

BP0 has a large number of buffers allocated, in an attempt to reduce I/O and help performance.

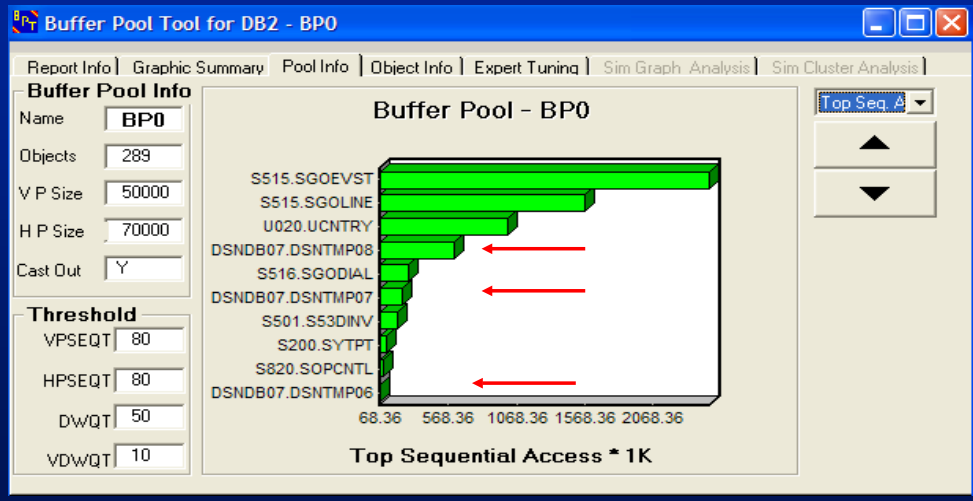
A lot of the objects in the system (289) are in BP0.

## Look at the Top 10 in many Categories



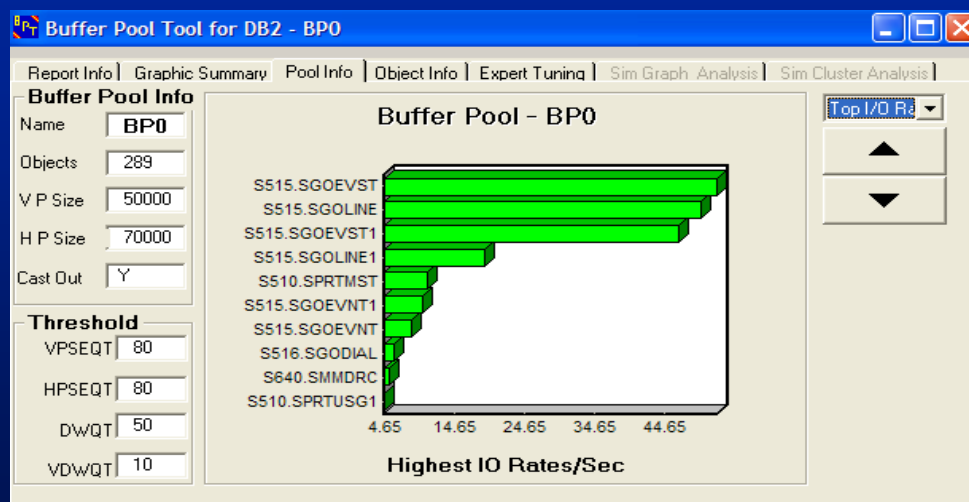
45% of the I/O is sequential, so the objects we are most concerned with, are those with the most sequential access - that are hurting the randomly accessed objects. We also want to look at the objects with the highest I/O rates,

***This is a major problem***



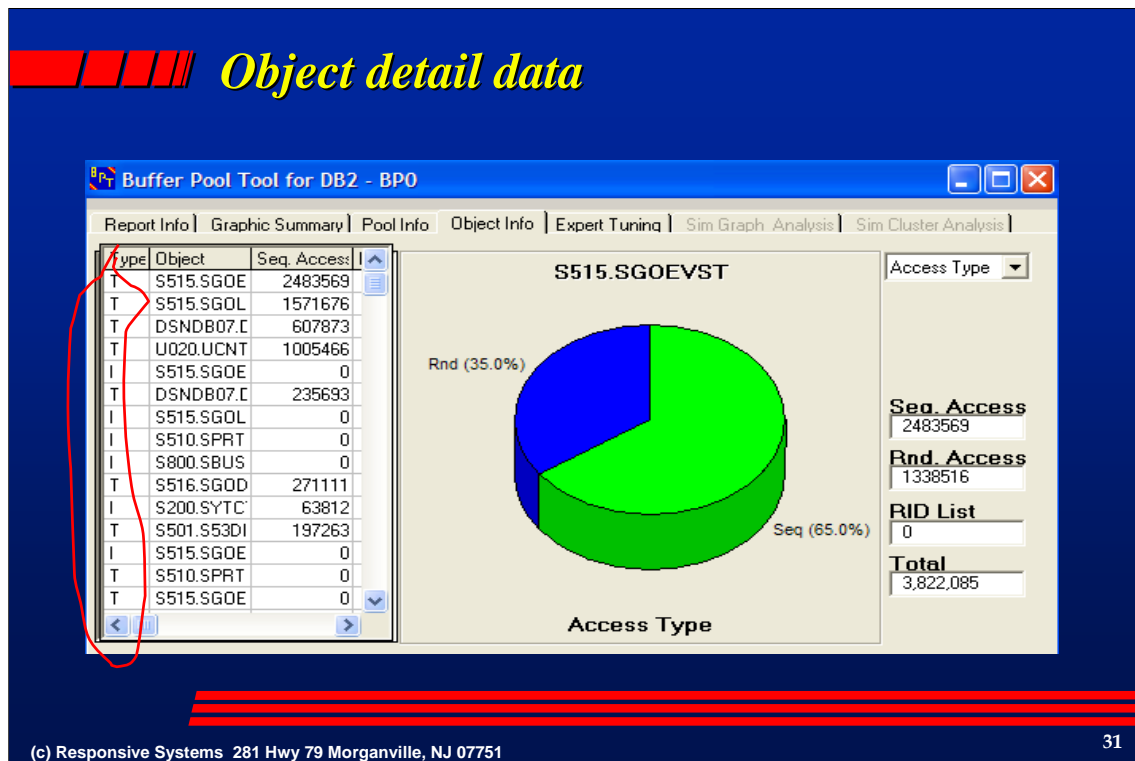
Hey guys, we need to get back to tuning basics..... Separate the Sort objects into their own pool.

## Top 3 have > 50% of the I/O



The top two sequentially accessed objects, are also the top two in the I/O rate/second category. This is not always the case in every system.

## Object detail data



We also have most of the indexes mixed in BP0 with the tablespaces, and the sort objects.

Developing some useful indexes on the top two objects will provide the highest benefit for reducing both I/O and CPU.

## BP20 also heavily SP

Buffer Pool Tool for DB2 - BP20

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

**Buffer Pool Info**

Name:

Objects:

V P Size:

H P Size:

Cast Out:

**Threshold**

VPSEQT:

HPSEQT:

DWQT:

VDWQT:

**Buffer Pool - BP20**

Access Type	Percentage
Seq	66.1%
Rnd	33.3%
RID	0.7%

Access Type:

**Seq. Access**:

**Rnd. Access**:

**RID List**:

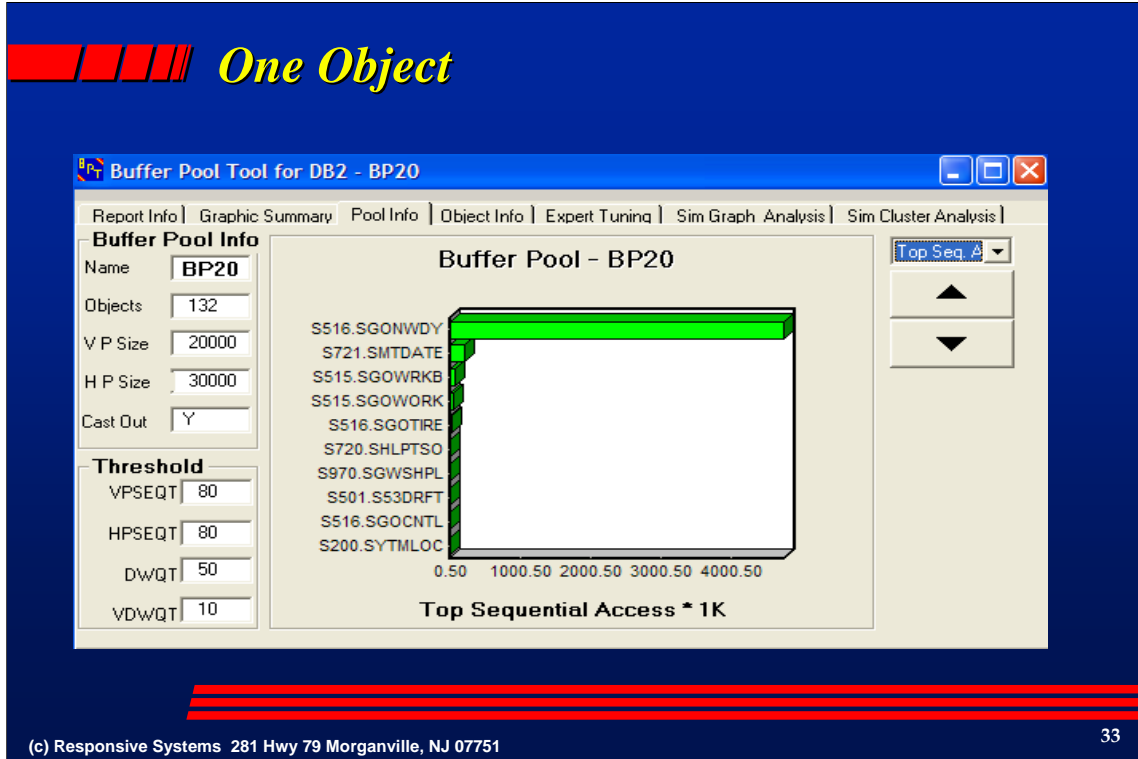
**Total**:

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 32

BP20 is also large, and heavily sequential, with the thresholds set too high.



## One Object



One object is causing almost all the SP activity.

**Object is pool resident**

**75% of pool GP**

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 34

This object has 5.7 million getpage requests, and 4.7 million of them are sequential.

*The object is pool resident, with no I/O activity.*

*This is a real CPU burner!!!*

## *First thought for this problem*

- Add an Index the application can use to eliminate the scan activity
- *A first guess may not always be right...*
- Let's look at some more information about this object

Let's dig a bit deeper into the available data before jumping to easy conclusions.

## Really small object

Buffer Pool Tool for DB2 - BP20

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

Pool Usage Intent:  Sequential  Random

Pool Size: 35000

Cluster Radius: 1.2

Cluster Info

Object	Smallest Ma	Largest Max
1	2207	3954
2	481	835
3	2	322

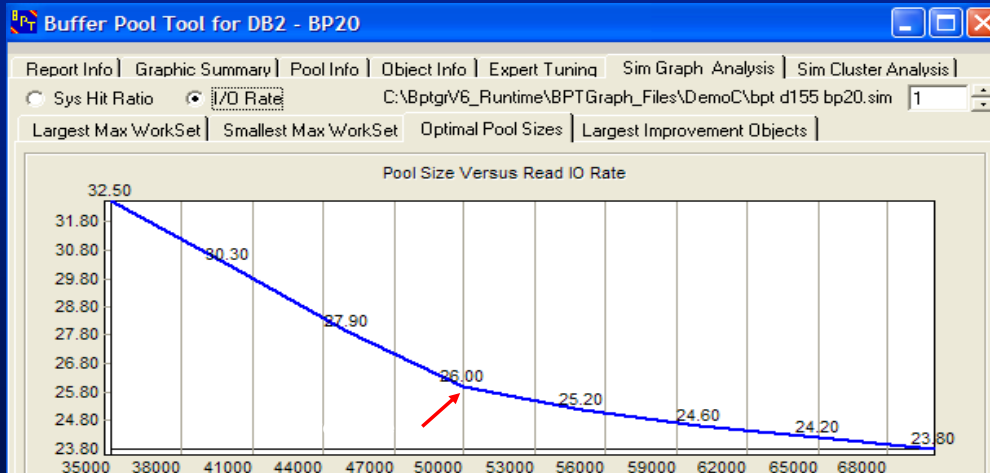
Type	Object	Max Work S
T	S516.DB2ADM.SGONW DY	6
T	S721.T9999DA.SMTDATE	3
T	S515.T9999DA.SGOWRKB	493
T	S720.T9999DA.SMTRANS	13
T	S720.DB2ADM.SHLPTSQ	63
T	S501.T9999DA.S53DRFT	322
T	S501.T9999DA.S53TIME	62
T	S200.T9999DA.SYTMLOC	6
T	S970.T9999DA.SGWSHPR	191
T	S516.T9999DA.SGOTIPI	63

The object has only 6 pages that are getting all that access and scan activity. But, since it is so small, it is not impacting the overall pool performance and I/O rate. It *is having a huge impact on CPU cycles* because of the number of pages scanned.

Now, if you noticed, the data was collected starting at 1:49 am... an obvious batch cycle period. So a good possibility, is that a batch program is repeatedly accessing this object for every transaction/process it executes.

## **No gain from memory**

Increasing from 50,000 to 65,000 buffers not a big improvement, only 1.8 I/O's



The pool currently has 50,000 buffers. If we added 15,000 buffers, 60 megabytes of memory, this would only save 1.8 I/Os a Sec. Not a worthwhile usage of memory. That very heavily scanned object is not having any real impact on the I/O rate, because it only occupies 6 pages.

## *Gain from an Index?*

- **If it would go directly to the desired page**
  - It would reduce the data access Getpages by 5/6
    - » But it would add an Index access
      - Maybe Index only access?
  
- **Saving....**
  - 3,950,000 Getpages for Index only access
  - But saving only 3,135,000 Getpages if data access required
  
- **Adding another Index will add some overhead to other processes**

Adding another Index would save more than 3 million Getpage requests, but might add some overhead to other processes if they update the data.

## *Other Alternatives?*

- **A possibility**

- Read the table into memory at the start of the batch job
- Would save 5.7 Million Getpage requests

Would eliminate a huge amount of Getpage activity – but might not be feasible if any other processes update the table.

## Next Pool BP25

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

**Buffer Pool Info**

Name:

Objects:

V P Size:

H P Size:

Cast Out:

---

**Threshold**

VPSEQT:

HPSEQT:

DWQNT:

VDWQNT:

**Buffer Pool - BP25**

Access Type

Access Typ. ▾

▲

▼

**Seq. Access**

**Rnd. Access**

**RID List**

**Total**

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751
40

Again we see a pool with a lot of buffers, and having a lot of sequential access. The sequential thresholds should be lowered to favor the randomly accessed objects.



# One killer object Problem1

The screenshot shows the Buffer Pool Tool for DB2 - BP25. The main window displays a bar chart titled "Buffer Pool - BP25" showing the "Top Sequential Access \* 1K" for various objects. The x-axis ranges from 0.88 to 4000.88. The y-axis lists objects. The highest bar is for S501.S53PROM1, labeled "Problem 1". The second highest bar is for S501.S53DINV1, labeled "Problem 2".

Object	Top Sequential Access * 1K
S501.S53PROM1	~4000.88
S501.S53DINV1	~1000.88
S515.SGOWORK1	~500.88
S516.SGOTIRE1	~500.88
S540.SDUPCHK1	~500.88
S820.SOPHEAD1	~500.88
S501.S53TIME2	~500.88
S516.SGODIAL2	~500.88
S501.S53DSRC2	~500.88
S200.SYTDSDR1	~500.88

Buffer Pool Info

Name: BP25

Objects: 212

V P Size: 25000

H P Size: 30000

Cast Out: Y

Threshold

VPSEQT: 80

HPSEQT: 80

DWQNT: 50

VDWQNT: 10

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

Top Seq. A

Chart area

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 41

Aside from the obvious problem object PROM1, DINV1 will also show some interesting problem information.

## Mostly Resident Problem 1

1/3 of pool GP

Buffer Pool Info		Total Get Pages		Pages Read Sync		App Hit Ratio	
Name	BP25	Total Get Pages	4534934	Pages Read Sync	31	App Hit Ratio	100
Objects	212	Get Page Rand	66961	Pages Read Seqpr	1115	System Hit Ratio	100
V P Size	25000	Get Page Seq	4467973	Pages Read Listpr	0	Read IO Rate/sec	0.03
H P Size	30000	Get Page RidList	0	Pages Read Dynpr	207	Pages / Write	0.00
Cast Out	Y	Avg Synchron IO (ms)	35.00	Avg SP IO (Seq Pref)	21.00		

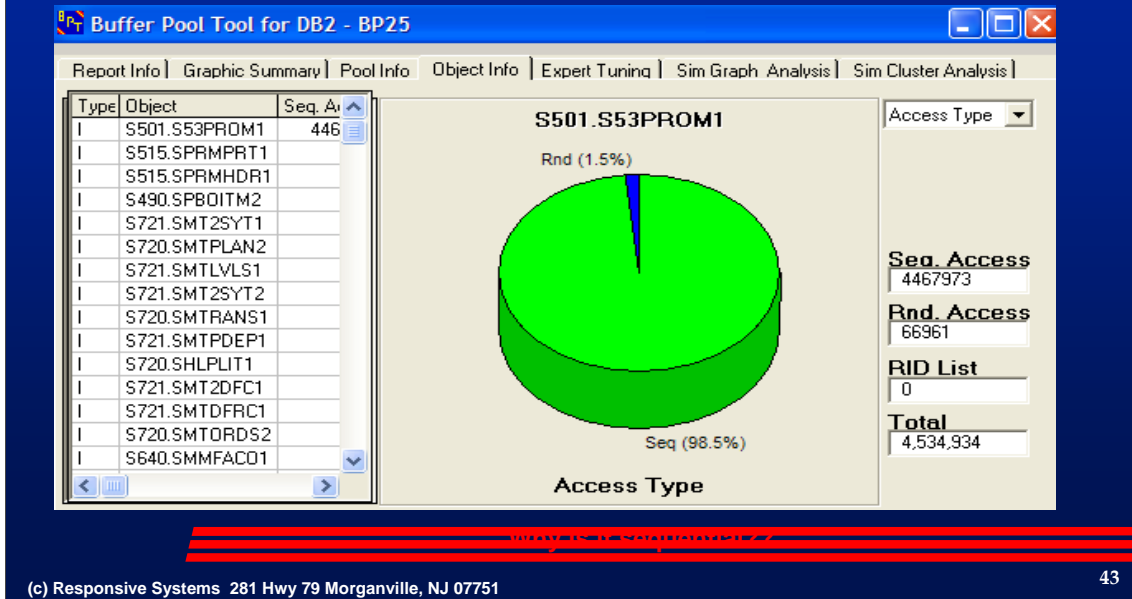
Thresholds: VPSEQT: 80, HPSEQT: 80, DWQT: 50, VDWT: 10

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 42

Prom1 has 1/3 of the pool getpage activity, and that's 4.5 million getpages - and almost all sequential activity.

When synchron I/Os are necessary, there is a cache miss and poor response time.

## It is an Index...



PROM1 is an Index..... So the huge scan activity indicates either poor design, or an sql coding problem. This is an opportunity to achieve a large cpu saving from reduced getpage activity.

The object is mostly pool resident as we will see later too, from the wkset size.

**It isn't that big...**

Buffer Pool Tool for DB2 - BP25

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

**Pool Usage Intent**      **Pool Size**      **Cluster Radius**

Sequential     Random      35000      1.2

**Cluster Info**

Object (A)	Smallest Max W/S	Largest Max W/S
1	25824	25824
2	1765	1773
3	2	1118

Type	Object	Max Work Set
I	S501.S.S53PROM1	1118
I	S516.S.SGOTIRE1	894
I	S540.S.SDUPCHK1	56
I	S516.S.SGODIAL2	852
I	S501.S.S53TIME2	34
I	S501.S.S53DSRC2	205
I	S200.S.SYTDSDR1	40
I	S821.S.SPNQBP1	829
I	S810.S.SMCRCOM1	62
I	S970.S.SGWSHPR2	62

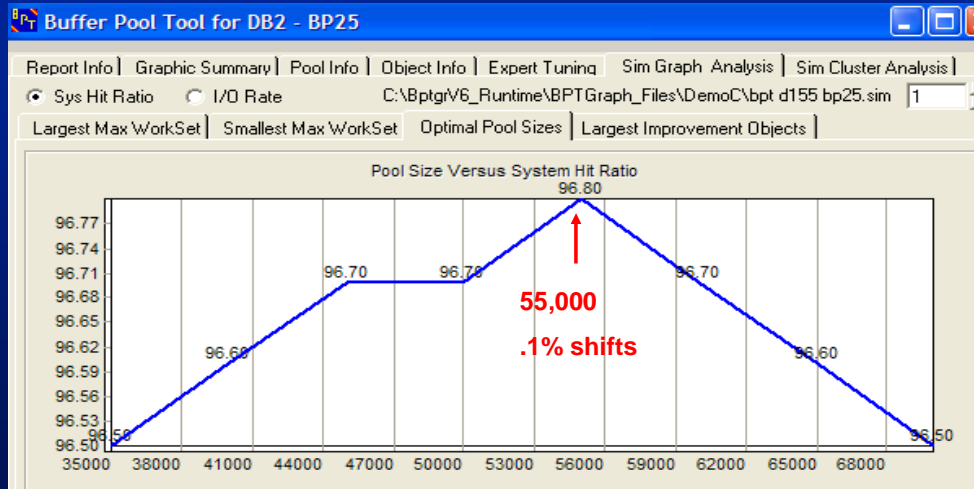
(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 44

The working set (wkset) size of PROM1 only 1118 pages in a pool of 35,000 buffers.

## *Why are we scanning an Index?*

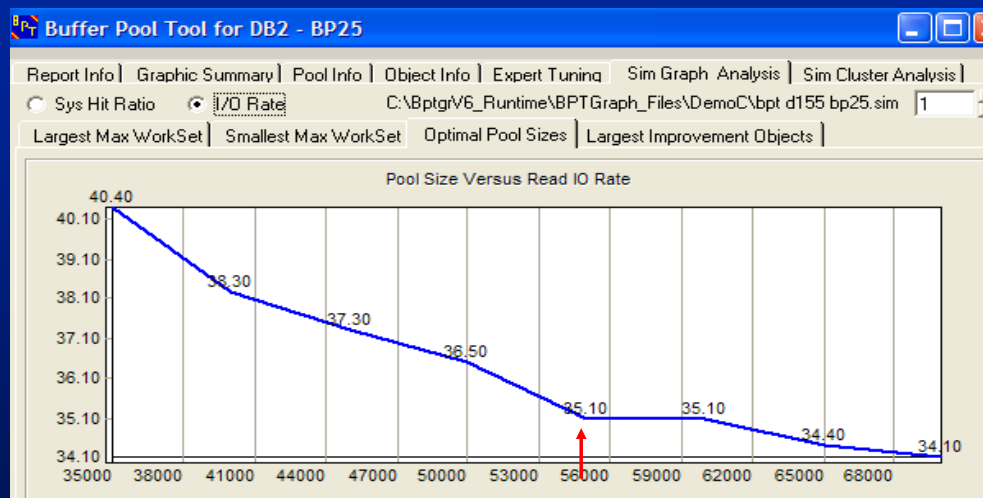
- **Poor/improper Index design**
- **SQL coding problems**
- **Fix the application.....**

## Interesting Graph



The pool currently has 55,000 buffers. So, even though the hit ratio is not the best metric to look at, it does show that reducing memory hurts performance, while increasing it does not help.

## *Less hurts, more doesn't help*



Looking at numbers, or performance relative to time....

The I/O rate is much more indicative of the performance impact of reducing or adding memory from the current 55,000 buffers.

## Monopolized the pool Problem2

The screenshot shows the Buffer Pool Tool for DB2 - BP25 interface. The 'Pool Usage Intent' is set to Sequential. The 'Pool Size' is 35000 and the 'Cluster Radius' is 1.2. The 'Cluster Info' section contains two tables:

Clusters			Objects		
Object	Smallest Max WS	Largest Max WS	Type	Object	Max Work Set
1	25824	25824	I	S501.S.S53DINV1	25824
2	1765	1773			
3	2	1118			

**The scan probably forced all the random pages out... Vpseqt=80**

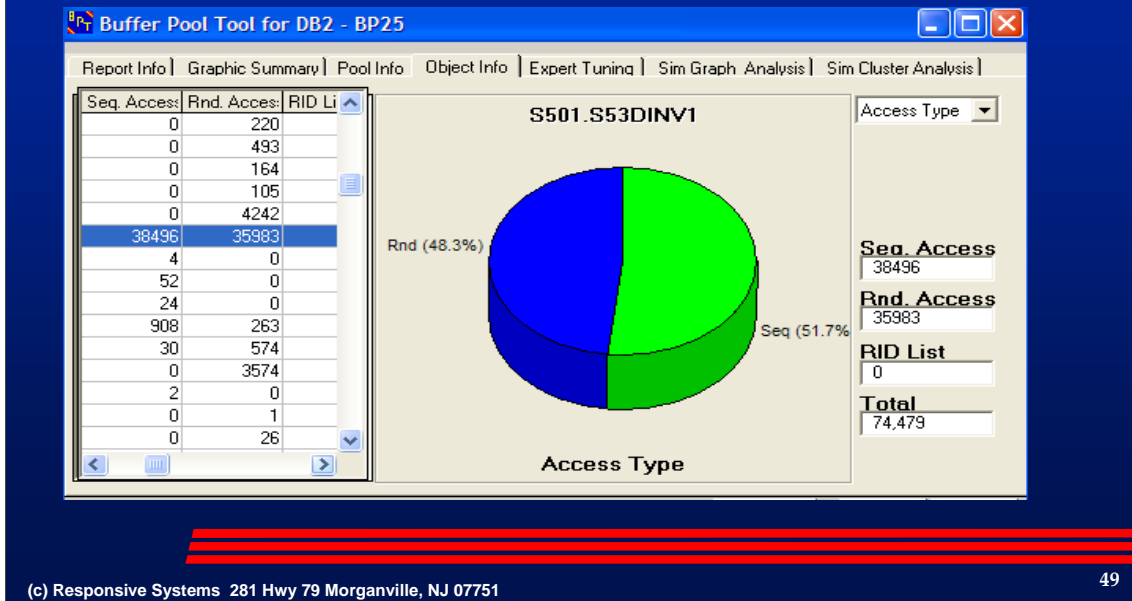
(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 48

Even though the activity against DINV1 is much lower than PROM1, at some point it monopolized the entire pool, and forced out pages of PROM1 and all the other random object index pages.

So, reducing the vpseqt to 25% or less will maintain the performance of the other objects in the pool, and reduce the overall I/O rate in this pool.



## Probably one scan, plus Random



Now if we continue to look at the activity against DINV1, we see that the access is almost evenly split between random and sequential.

So - the access was mostly random, and had one, or possibly a couple of scans. However, it was the scans that hurt the performance of everything else in the pool.

## **A Different DB2 System, High I/O**

**Buffer Pool Tool for DB2 - BP3**

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

**Collection**

Date: 2003-07-02  
Time: 10:00:01  
Elapsed Time: 01:00:00

**System Info**

System: HNB1  
Sub System: PDB2  
Db2 Version: 6.1

Pool	I/O	Get Pages	Updates	Hit Ratio	I/O Sec	Pages
BP0	667	557086	2143	99.8	0.19	
BP1	7921	3799670	2680542	98.9	2.20	
BP2	646190	3765739	152676	44	179.50	
<b>BP3</b>	<b>1261809</b>	<b>4380447</b>	<b>5990</b>	<b>43.7</b>	<b>350.50</b>	
BP4	918912	5459239	2957	69.1	255.25	
BP5	979155	4327356	91354	70.9	271.99	
BP6	205834	2831487	21522	41.9	57.18	
BP7	101370	787462	16466	87.2	28.16	
BP11	9	2783937	20	100	0.00	
BP40	97	567	331	90.3	0.03	
BP48	1813	79065	102	80.1	0.50	
BP49	106	10205	2554	99.6	0.11	

Total Read/Write IO: **4,124,183** Total Get Pages: **28,791,468**  
Overall Sys Hit Ratio: **67.61** Total I/Os per second: **1,145.61**  
Total Updates: **2,976,785** Pages per write: **2.28**

(c) Responsive Systems 281 Hwy 79 Morganville, NJ 07751 50

Here is a system with a much higher overall I/O rate.

## After Tuning the Pools

The screenshot displays the Buffer Pool Tool for DB2 interface. The 'Collection' section shows the data was collected on 2003-10-06 at 10:00:01, with an elapsed time of 01:00:00. The 'System Info' section indicates the system is HNB1, sub-system is PDB2, and the Db2 version is 6.1.

Pool	I/O	Get Pages	Updates	Hit Ratio	I/O Sec	Pages/W
BP0	1487	495316	4598	99.6	0.41	2.
BP1	32884	4246159	2689301	96.5	9.13	20.
BP2	402924	7121412	62070	92.9	111.92	2.
BP3	693605	4290430	7272	50.4	192.67	1.
BP4	452779	5476165	2842	83.2	125.77	1.
BP5	663171	11295540	84336	93.8	184.21	1.
BP6	164362	4066247	12857	61.8	45.66	1.
BP7	21090	392904	3113	59.1	5.86	1.
BP11	11	2672005	12	100	0.00	1.
BP40	49	697	358	92.4	0.01	8.
BP32K	0	64	64	101.6	0.00	0.

Summary statistics at the bottom of the tool:

Total Read/Write I/O	<b>2,432,362</b>	Total Get Pages	<b>40,056.939</b>
Overall Sys Hit Ratio	<b>84.72</b>	Total I/Os per second	<b>675.66</b>
Total Updates	<b>2,866,823</b>	Pages per write	<b>3.87</b>

Red arrows point to the Total Get Pages, Total I/Os per second, and Pages per write values.

Here is the same system 3 months later at the same timeframe, the number of pools has been reduced, the getpage activity has increased about 42%, and the I/O rate has dropped by 470 per second.

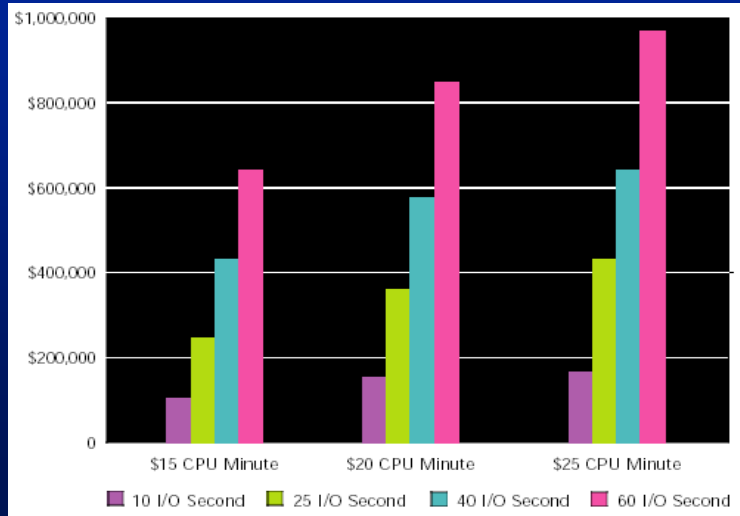
The greatly reduced I/O allows more useful work to get through the system, reflected by the increased Getpage activity.

## *Huge Performance Improvement*

- **The throughput rate (getpage activity) increased more than 42%**
- **The I/O Rate/Sec dropped more than 40%**
  - Saving 470 I/Os per second

This is easily a six figure cost saving from I/O elimination.

## **Cost Savings from I/O Elimination**



## 9 Minutes of Data, extreme I/O rate

Too many  
pools

Pool	I/O	Get Pages	Updates	Hit Ratio	RIO/Sec	WIO/Sec	Pages/Write
BP0	296	31154	914	99.2	0.37	0.17	4.06
BP1	2	368	92	100	0.00	0.00	9.00
BP2	242	3165253	2128023	100	0.44	0.00	0.00
BP3	34136	2567512	83108	95.6	59.14	3.38	13.31
BP4	39451	2283803	46286	97.8	70.58	1.68	8.29
BP5	307	3534	103	94.9	0.27	0.29	1.03
BP6	21470	326944	26341	90.5	32.35	6.98	3.20
BP7	21	50653	63	100	0.03	0.01	1.00
BP8	26	129057	22	100	0.05	0.00	1.00
BP11	236600	1045550	312	77.4	433.14	0.19	3.45
BP13	147	904	503	72.6	0.16	0.11	8.18
BP14	52490	1053955	72590	64.8	93.73	2.41	19.02
BP23	587005	4630481	379764	85.1	765.37	309.73	1.74
BP24	436740	1764411	349297	77.8	544.21	255.68	1.99
BP26	44350	496102	26334	85.8	73.12	8.11	3.26
BP33	88205	1832650	162935	65.2	150.07	11.48	7.00
BP34	19073	210266	676	-6.1	33.07	1.87	1.06
BP43	603915	4349025	91943	34.3	1,077.14	28.94	3.57
BP44	480966	6849343	95790	89.2	860.65	20.24	2.97

Total Read/Write IO	<b>2,645,442</b>	Total Get Pages	<b>30,790,965</b>
Overall Sys Hit Ratio	<b>79.18</b>	Total I/Os per second	<b>4,845.13</b> ←
Total Updates	<b>3,465,096</b>	Pages per write	<b>2.22</b>

# 15.75 Minutes of Data even higher I/O rate Different System

**Collection**

Date:

Time:

Elapsed Time:

**System Info**

System:

Sub System:

DB2 Version:

Pool	I/O	Get Pages	Updates	Hit Ratio	RI/O/Sec	WIO/Sec	Pages/Write
BP0	86914	792976	60	77	91.65	0.03	1.88
BP2	2989001	20443958	76353	-18.6	3,122.22	30.74	1.77
BP3	1304487	16649309	112401	59.4	1,321.93	54.11	1.42
BP4	836	39610	9288	96.4	0.81	0.07	5.74
BP6	619697	1602799	7192	-63.1	651.08	2.61	1.60
BP7	41327	262998	23924	68.4	37.85	5.74	2.06
BP13	39	222	104	88.3	0.03	0.01	3.75
BP30	5701	1245951	213501	98	3.45	2.57	22.46
BP49	0	500	300	100.2	0.00	0.00	0.00
BP32K	10560	82445	19188	84.7	9.29	1.85	2.04

Total Read/Write IO: **5,058,562**      Total Get Pages: **41,120,768**

Overall Sys Hit Ratio: **17.50**      Total I/Os per second: **5,336.04** ←

Total Updates: **462,311**      Pages per write: **2.14**

A new high for an I/O Rate/Sec. This is a standard AM workload.

## 43 Minutes of Data, after Pool Tuning, and fixing some Application problems

Buffer Pool Tool for DB2 - BP2

Report Info | Graphic Summary | Pool Info | Object Info | Expert Tuning

**Collection**

Date: 2004-08-09

Time: 11:20:31

Elapsed Time: 00:43:05

**System Info**

System: NENT

Sub System: NBP

DB2 Version: 7.1

Pool	I/O	Get Pages	Updates	Hit Ratio	RIO/Sec	WIO/Sec	Pages/Write
BP0	67558	2160690	841	96.4	26.06	0.08	1.71
BP2	4521410	50454995	161961	26.8	1,740.15	8.95	2.42
BP3	647494	23433289	297839	82.4	235.75	14.74	2.30
BP6	619596	17195814	36217	78.7	237.71	1.98	2.72
BP7	31187	481841	34338	78.7	9.84	2.22	2.40
BP13	38	1150	75	99.6	0.00	0.01	1.69
BP30	105778	3926607	2859138	85.8	28.69	12.23	26.89
BP40	11127	128423	18	68.7	4.30	0.01	1.00
BP49	0	511	408	100.2	0.00	0.00	0.00
BP32K	17692	154006	61300	68.9	5.87	0.98	1.87
BP8K0	34	1692	11	91.5	0.01	0.00	1.00
BP16K0	4	30	4	103.3	0.00	0.00	1.00

Total Read/Write IO: **6,021,918**    Total Get Pages: **97,939,048**

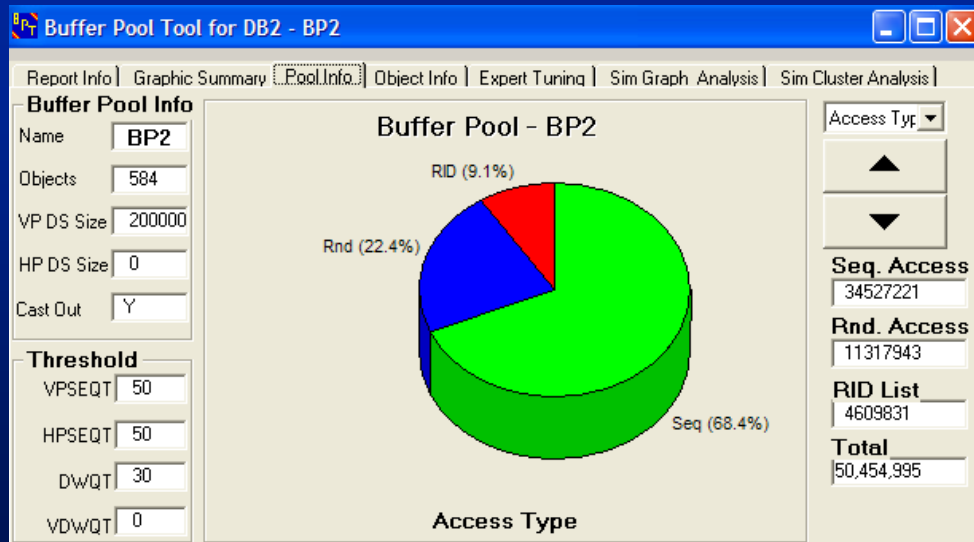
Overall Sys Hit Ratio: **53.53**    Total I/Os per second: **2,329.56**

Total Updates: **3,452,150**    Pages per write: **9.64**

After tuning the pools, and fixing some application performance problems.

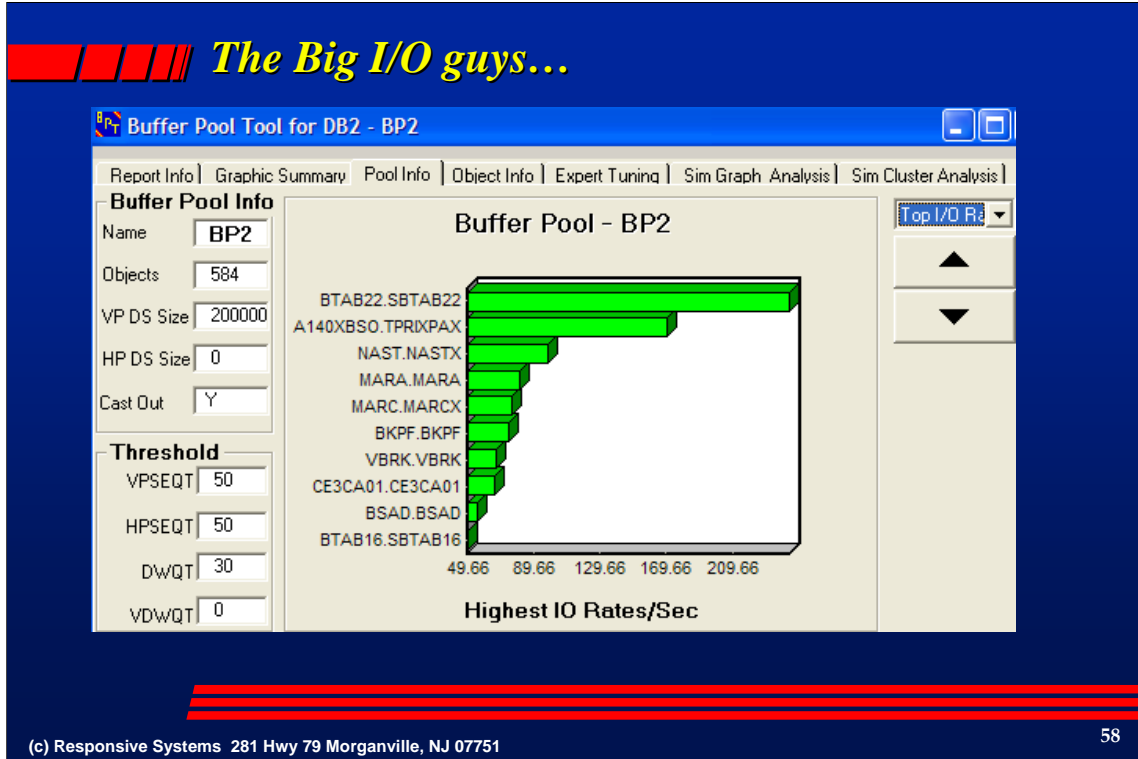


## What's happening in BP2?



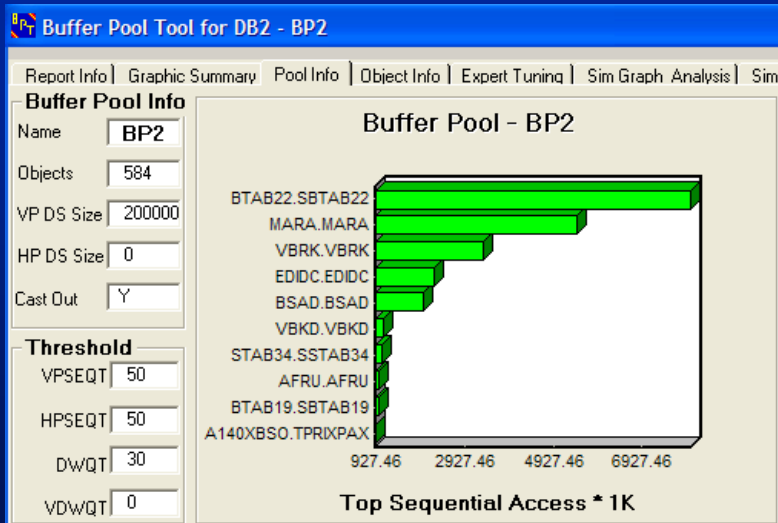
BP2 has the highest I/O rate, and the access is heavily sequential.  
Always look at the big problem areas first.

## *The Big I/O guys...*



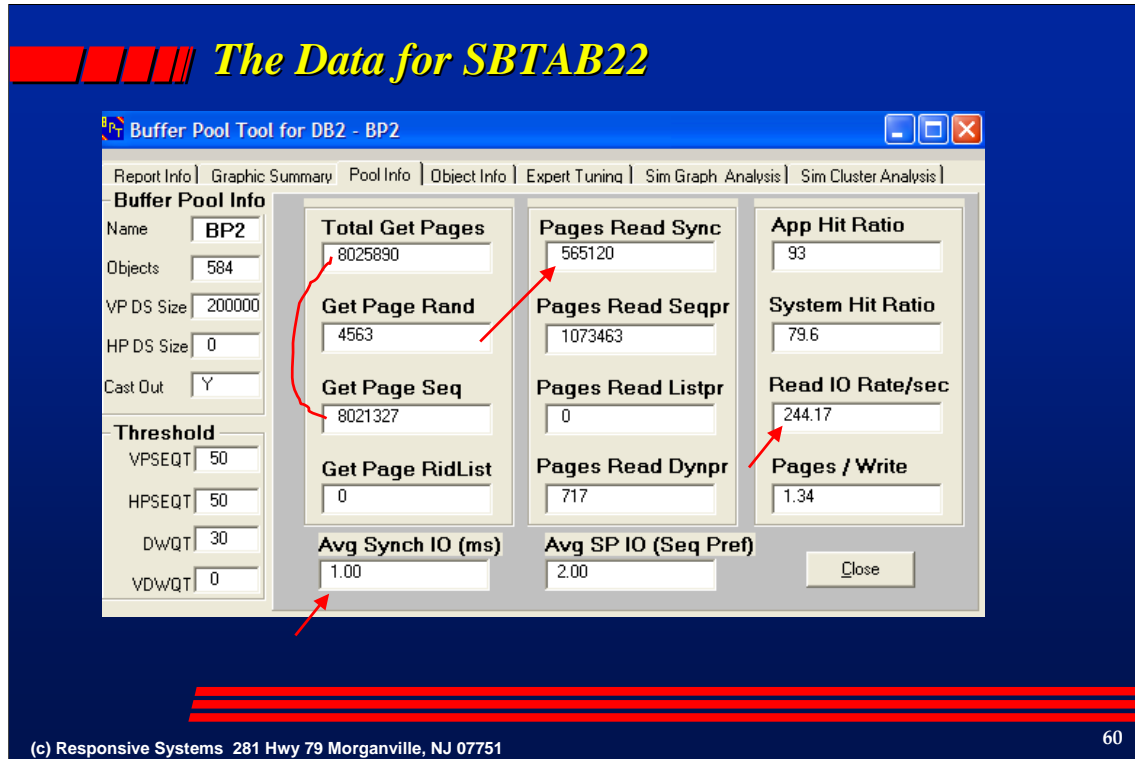
The bad guys....

## **The Heavy Sequential Objects**



The *really bad* guys.... So let's take a lower level look at SBTAB22

## The Data for SBTAB22



We have several interesting things here:

A: almost all the getpage requests are sequential

B: high synch I/O activity

C: high I/O rate

D: vpseqt of 50% can allow up to 100,000 pages for one sequential scan

**Analysis:** 1/8 of the requested pages are read into the pool with prefetch, but sequential data falls off the LRU queues before the application can read the data – and the pages are read back in using Synch I/O. The 1 Ms avg Synch I/O time shows that the requested pages are all found in the cache of the dasd control unit.

**So** – if it can live at the dasd cache level, maybe a larger pool will be able to keep it in memory?

## Working Set Size in the Pool

Buffer Pool Tool for DB2 - BP2

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

Pool Usage Intent:  Sequential  Random

Pool Size: 200000

Cluster Radius: 0.8

C:\Bptqrv6 Runtime\BPTGraph Files\DemoBiqPool\d080904x2.sim

Cluster Info

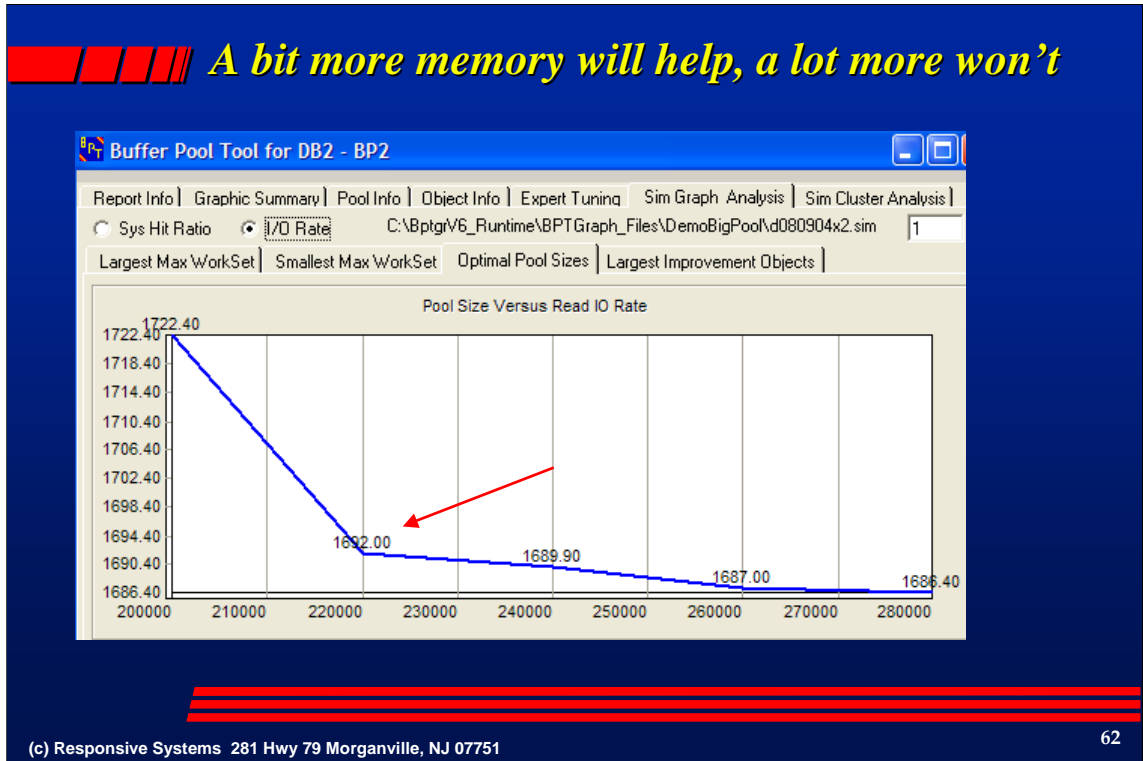
Object	Smallest Ma	Largest Max
1	65195	98660
2	41603	56677
3	14400	33669
4	1	12916

Type	Object	Max Work Se
T	BTAB22.SAPR3.SBTAB22	9282
T	LTBK.SAPR3.LTBK	4906
T	STAB33.SAPR3.SSTAB33	12916
T	LFA1.SAPR3.LFA1	8480
T	TSP02.SAPR3.TSP02	6943
T	POOL46.SAPR3.SPOOL46	354
T	BSIS.SAPR3.BSIS	11722
T	TSP01.SAPR3.TSP01	4199
T	LFB1.SAPR3.LFB1	5217
T	LX42XR9Q.SAPR3.XSAP	1127

Since the Wkset is not really big compared to other sequential objects, there are probably many scans/accesses of the object, and it uses less than 5% of the pool.

**Analysis:** It's probably accessed a lot by a batch job, that does not have a high enough priority to get the pages read in by prefetch, and they have to be re-read with Synch I/O a second time when the actual getpage is issued.

**So** - again, will a larger pool help?



If the pool is increased by 20,000 buffers (80Meg) to 220,000 buffers, this will save 30 I/O sec. Increasing beyond this size is a complete waste of memory. We started with a very large pool – in this case making it even larger (+10%) will help.

**But** – just throwing huge amounts of memory at pools does not always improve performance!

**64Bit memory** has some great possibilities, but more memory resources do not necessarily mean better performance.

**The proper grouping of objects, and effective use of memory can provide substantial performance improvements.**

## *Perspectives of Performance data*

- Lies
- Darn Lies
- And Statistics
  - What is the sample?
  - Is it meaningful
  - Averages over long time periods (many hours) are useless....
  - Mini snapshots, at measured intervals, violate established statistical standards
  - Averages, of averages, of averages, violate statistical techniques
- There are Sampling techniques
  - And there can be built in “bias” for a sample
    - » Sometimes deliberate, sometimes a lack of knowledge....

There are proper ways to collect data, that provide statistical validity, and these techniques are clearly documented by the National Bureau of Standards. Data collection techniques that violate accepted standards, rarely, if ever, produce reliable results.

When comparing sets of performance data, the data must be from the same timeframe, same duration, and hopefully a workload that is reasonably close to the first. Now, unless you have a well documented, canned, and repeatable benchmark process, there will always be some degree of variation in your workload. You always need to look at the level and type of activity that occurred within your data intervals, and make you own determination if they are close enough for reasonable comparisons.

If you take small snapshots of data across a day, they need to be compared individually - and grouping the data together into averages usually produces gross errors of both performance and perspectives. Looking at performance data across long periods like 12 or 24 hours is completely useless for tuning and analysis purposes - because it will mask all performance spikes and problems.

## Summary

- The Hit Ratio is not a *measurable* performance metric
  - But still nice to track
    - » If you track the *System Hit Ratio*
- The I/O rate *is measurable*
  - This can be tracked quite easily from many data sources
  - Calculate both CPU and elapsed time savings
  - CPU savings can be converted to \$\$
- Throwing memory at pools, arbitrarily, usually *doesn't* provide better performance
- 64 Bit large memory is not a panacea - yet!

Most sites can optimize performance using 6 to 8 pools, some larger ones may need a dozen pools. A small number of very large pools, *absolutely will not*, provide good performance. The key to good performance remains the proper grouping of objects into pools based upon access type, and working set size. Until we can provide TeraBytes of memory, pool tuning will remain an important performance issue.



 *Thank you for coming*

Are there any Questions?

Joel Goldstein

Responsive Systems Company

[Joel@responsivesystems.com](mailto:Joel@responsivesystems.com)