This presentation addresses the approaches, values, thresholds, triggers, and relationships of performance data analysis. Performance monitoring, analysis, and tuning should be a pro-active and continuous process; however, this is often neglected because people are too busy with other work, or they are not sure what data is really important.

Certain thresholds and data provide quick insights about problems both at the application level, and the system level.

As you peel back the layers of performance information, each area that we improve has an impact upon other areas. These performance layers are a lot like the layers of an onion. We start from the obvious outside metrics, and work our way in to lower levels of performance measurements and tuning opportunities.
1. DB2 Data Sources
   a. Application Accounting Data
   b. System Statistics Data
   c. DB2 performance trace data

2. The important performance variables and indicators:
   a. Variables and relationships
   b. Inter-relationships between Statistics and Application data
   c. You have a problem – how bad is it?

3. Top down tuning approach - The Eyeball approach
   a. Tuning from available DB2 data sources
   b. How much can you tune - the payback potential
   c. How long will it take?

4. Peeling back the layers of performance relationships
   a. Data interpretation
   b. Determining where the benefits will be

5. Analysis of performance data from large systems. Some of the interesting things that appear...

6. Summary, Guidelines and recommendations
Two pieces for the presentation

• First, the basic approaches for performance analysis
  • Meaningful
  • Not so meaningful
  • Maybe, or maybe not – dig a bit deeper….

• Then we will look at both application and system performance data

• Knowing where to go saves a lot of time
  • A is often caused by B. B may be caused by C….
As we look at performance, we see a problem, and take remedial action. Great, that performance got better. Whoops…. What is this? Another problem!! Every time we fix an issue, we remove one stress point, and another pops up.

As examples, tuning buffer pools reduces I/O delay, or maybe major bottlenecks. Removing the I/O delay often increases the processor CPU busy rate, because more work gets through the system in the same measurement intervals. Decreasing the I/O rate/second increases the Getpage rate/second.
The basics of performance have not changed within the last 40 years.

- Processor
  - CPU speed
  - Number of engines

- Memory

- I/O
  - Elapsed time
  - CPU cost of I/O
Some management people may be surprised....

- Reducing the I/O rate
  - Often increases the processor busy rate
  - Inverse relationship between I/O and Getpages ↑
    - Latent demand

- Some pro-active tuning often saves million$ per year from reduced CPU on large systems
  - Avoid a processor upgrade
  - Avoid the huge software upgrade charges from a processor upgrade
  - Productivity improvements for everyone..... $$$
The amount of work, and the way it’s done, affects both elapsed times, and CPU processing resources.
Data from multiple sources must be used for tuning. If you don’t know what data you need, how to get, and how to analyze it, your tuning effectiveness will.... be less than management wants/needs. Producing Stats records every 60 seconds is only 1440 records, not a problem. This lets you find problems with a low level of granularity, but start from a higher level of initial analysis like 15 or 30 minutes.

While using anything is vastly better than not tuning, it’s important to understand the limitations of data. While Lstat displays, and 199 object usage records will provide some useful information, tuning from there is simply a guess. There is no validation without changing your system. Can you afford guesswork on your production system?

Making a pool larger and monitor performance won’t hurt if you don’t impact system paging. Move an object to the wrong pool, and you’re in big trouble.
Data, Data, Everywhere…

- Never as we want it…
  - Too much
  - Too little
  - Wrong intervals or duration
    - Too much of short duration, we can summarize
    - Too little, or too long a duration, and it’s useless
  - Hindsight is always perfect, if you know what you want/need
    - People frequently send data that isn’t useful…

We need the ability, and tools, to slice/dice data many different ways. Putting data into Excel can open a whole new world of analysis.
Tuning Goals

- Reduce processing costs
  - Save $$
    - Reduce transaction/business function/job CPU processing cost
    - Improve user productivity by reducing elapsed times
    - Avoid or delay processor upgrades
      - Software upgrade costs are often much more than hardware
- Need History data
  - How does today’s pain compare to last week/month?

History data lets you see if this is a new problem, or an old issue that nobody complained about before. Has it always been this bad, or has it gotten worse over time?
Long measurement intervals hide performance problems by showing averages over a long period. An interval spanning a hour or two during peak load periods would be valid in most cases to get average response times. However, when periods of low system activity are factored into the data, it becomes meaningless – does not represent the times you should be most concerned with – the peak periods.
Shorter periods show the peaks and valleys. From here you should go to 5 minute periods between 9:45 – 10:15, and 12:45 – 13:15.

Then look into detailed application and system performance data to determine the cause of the response time spikes.

What can be done to reduce the I/O wait time issues?
Data relationships, and %, are vital

- Sometimes the raw numbers get our attention
  - Transaction elapsed times of 5, 10, 20 seconds

- Sometimes percentages are more meaningful
  - Transaction elapsed .1 secs, .06 I/O wait
    - I/O wait is 80% of the elapsed time

- What other factors or events might have impacted the numbers you are looking at?
  - Elapsed times are usually < 1 sec, but spike to 10 secs for some periods
    - Who did what, to whom…??

Eyeball method – what looks big or out of proportion to other data?
We can play many games with data, depending upon what we want to achieve, or prove.

Lies, darn lies, and statistics........

Statistics can prove, or disprove anything, depending upon the data sample.
Finding one object with heavy scan activity, continually, will pay for your salary many times over.
Changes because of tuning activity are expected to have a positive impact. However, changes that sneak into your system from application changes, re-binds, or migration to new releases can destroy performance and throughput.
Application accounting report for a batch job. Job is running too long, DBA said they had a buffer pool tuning problem, and the systems people would not tune the pools.

While pool tuning might reduce overall I/O and thus the elapsed time, the underlying issues are:

a. Death by synch I/O – possibly sort the input into the key order of the data

b. The cache on the DASD controller is too small

c. The synch I/O elapsed time is poor, that relates to b.

Speaking with the sysprogs, and analyzing system performance data, pools cannot be increased without causing the system to page. They know they have a DASD performance issue, the cache is too small, and DASD upgrades are a few months in the future.
One of the most difficult problems when comparing performance, is having workloads that are reasonably similar. Unless you have a stand-alone test system, with a specific driven workload, you will never achieve an exact comparison. So the key for comparisons is – “reasonably similar”.

Made tuning changes is performance better now?

- Workload comparisons must be *reasonably* similar
  - Quite common to have significant differences of object usage, objects accessed, type of access
    - Same timeframe?
    - Length of comparison interval
  - Is your workload infinitely variable?
    - Very difficult to measure
      - Object activity inter-relationships
      - Dynamic queries?
      - Batch jobs all day?
Client made pool tuning changes to measure the improvement of a batch workload. Adamant that they ran the exact same workload, and complained that the I/O rate increased.

However 27 additional (new) objects were accessed during the second performance measurement period. The Getpage rate was 10% higher, number of pages written, and write I/O was higher.

The workload in the overall system was quite different.
The following performance data is after moving to a larger, faster machine, with more memory – and other workloads.

High overall I/O rate, spread across several pools

Before tuning, making some pools larger
Predicted saving by doubling BP2 from 20,000 to 40,000 buffers, a bit less than 200 I/O second

Before tuning, making some pools larger....
Predicted saving by doubling BP11 from 20,000 to 40,000 buffers, about 50 I/O sec, 1165 at 40,000.

Before tuning, making some pools larger....
High overall I/O rate, spread across several pools

The I/O rate dropped for both BP2 and BP9 that were increased.

We predicted 720 I/O second at 40,000 buffers, we got 737.

There are still variations in the workload.
Additional doubling BP11 from 40,000 to 80,000 buffers will save about 20 I/O second. Not a large payback.
Doubling BP2 from 40,000 to 80,000 saves almost 100 I/O second. There might be a bit more to be gained, but the curve is flattening rapidly.
Doubling BP9 from 40,000 to 80,000 saves about 20 I/O second. Not a large payback, and the curve flattens as we give it more memory.
This illustrates the objects with the highest I/O rates across the system, shows how many CPU seconds of CPU are caused by the I/O, and the application elapsed time effects of the I/O. The underlying reason for the huge difference between the IO elapsed times, is because of the DASD response times.
48B has a higher number of GP, a lower I/O rate, and lower Sync I/O times.

48B found more pages in the pool, and more pages in the DASD cache. The cache hit rate was about 100%. 
BP11 is 100% random access, we can see the top 10 I/O objects in the pool, and then see the performance characteristics of each object. Overall avg. synch I/O times of 2 Ms, is very good, means almost all pages are found in the DASD cache. That also tells us that making the pool larger will find more pages in the BP, and reduce the I/O rate, and save CPU cycles.
Working set size is often a crucial factor used for splitting objects into pools.

The general methodology, is Ramos/Samos – (randomly accessed, mostly), and (sequentially accessed mostly); then within those criteria, very large working set objects from the rest. The goal is to separate objects that monopolize a pool.

The first obstacle we encounter in this overall workload, is that it’s almost all random.

Based on the working set sizes in each pool, there probably isn’t much opportunity for gain by splitting out objects.

BP9 is a maybe, based on 4 objects in each cluster.
There is 2 Gig of memory available on the LPAR, can we increase any pools?
Even with 2.4 Gig available, the system has shown paging activity over the monitored interval. Pool increases might hurt overall performance.
Paging indicates that the overall system memory requirements are too high. Paging impacts the entire LPAR, not only DB2.

There are Synch writes, but the DM threshold has not been reached, so this isn't a problem.
Logging delays, and cross-system contention with another member in the data sharing group.

While the elapsed time is good, 96% is wait time. This is one of little item often overlooked during performance analysis.

Because the elapsed time is good, nobody looks for the delay problems.
EDM Pool usage is fine, there are no failures. However, the pool is over allocated, and some of the excess memory might be better utilized elsewhere.
RID Pool has some failures that should be investigated. The RDS limit problem can only be eliminated by changing SQL to get a different access path.
Monster pool, and running very tight on space.

<table>
<thead>
<tr>
<th>EDM POOL</th>
<th>QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGES IN EDM POOL</td>
<td>62500.00</td>
</tr>
<tr>
<td>HELD BY DBS</td>
<td>4810.00</td>
</tr>
<tr>
<td>HELD BY CTX</td>
<td>77.94</td>
</tr>
<tr>
<td>HELD BY SRTS</td>
<td>3123.79</td>
</tr>
<tr>
<td>HELD BY SRTS</td>
<td>51155.16</td>
</tr>
<tr>
<td>HELD BY PTX</td>
<td>3703.64</td>
</tr>
<tr>
<td>FREE PAGES</td>
<td>4440.00</td>
</tr>
<tr>
<td>% PAGES IN USE</td>
<td>92.89</td>
</tr>
<tr>
<td>% NON STEAL. PAGES IN USE</td>
<td>6.05</td>
</tr>
<tr>
<td>FAILS DUE TO POOL FULL</td>
<td>0.00</td>
</tr>
<tr>
<td>DED REQUESTS</td>
<td>6280.6K</td>
</tr>
<tr>
<td>DED NOT IN EDM POOL</td>
<td>458.00</td>
</tr>
<tr>
<td>DED HIT RATIO (%)</td>
<td>99.99</td>
</tr>
<tr>
<td>CT REQUESTS</td>
<td>1453.0F</td>
</tr>
<tr>
<td>CT NOT IN EDM POOL</td>
<td>460.00</td>
</tr>
<tr>
<td>CT HIT RATIO (%)</td>
<td>99.93</td>
</tr>
<tr>
<td>PT REQUESTS</td>
<td>33597.3K</td>
</tr>
<tr>
<td>PT NOT IN EDM POOL</td>
<td>40772.00</td>
</tr>
<tr>
<td>PT HIT RATIO (%)</td>
<td>99.99</td>
</tr>
<tr>
<td>PAGES USED IN STMT POOL</td>
<td>68679.00</td>
</tr>
<tr>
<td>FAILS DUE TO STMT POOL FULL</td>
<td>0.00</td>
</tr>
<tr>
<td>PAGES IN STATEMENT POOL</td>
<td>89042.00</td>
</tr>
<tr>
<td>FDRK POG IN STMT FDRK CHAIN</td>
<td>963.00</td>
</tr>
</tbody>
</table>
So, DB2 just ate your system. However, with the memory usage mentioned, and NO paging activity, it’s unlikely that the pool increases caused the problem.

The fact that it went away after pools were decreased, is probably incidental luck.
The only time I ever saw DB2 completely eat a machine was during a huge backout, and that was more than a decade ago..

- We asked some questions, and asked for a DB2 Statistics report

- The next data was an RMF report
  - No paging activity, what-so-ever, nothing
  - All 6 engines very busy
  - Several devices with heavy I/O volume
    - Not paging or swap volumes

- Waiting for DB2 Statistics
A lot of rollbacks, and a huge number of reads from the Active Log. The read rate for the active logs is 1317/Second. That’s a very high I/O
Dataset opens, Prefetch Disabled-No Read Engine.
Write Engine Not Available. DWQT is high, VDWQT is low – we should see the opposite. Pages/Write is low single digit, set VDWQT=(0,40)
Note Pages for Prefetch read…
High numbers for VDWQT vs. DWQT
Both members are in the same DB2 V8 DS Group
The application design and usage cost 19 Secs of CPU during the 15 minute data period.
Common type of access and usage I see in sort pools. The VPSEQT should not be set at 100%, for a small pool like this, I would try 90%. For large pools 95-98%.
We should be able to get some really good tuning paybacks here if we have memory to work with !!!
This is a RED alert !!!!
It’s in trouble right now !!!

This is a RED alert !!!!
This is a RED alert !!!!
There are 6 pools with high I/O rates. We do want to look at all of them, but which one would you look at first?
BP4 has the objects that are hurting us the most. Eight of the top twenty for CPU cost, and four of the top five for I/O Elapsed time – that directly impacts your application response times.
Depending upon how much memory we can afford, 30,000 buffers will save us 200 I/O second.
Highest IO rate seen to date…
Huge memory…
19 Gig of 4K pools, plus \( \frac{1}{2} \) Gig as a sum of 8K, 16K and 32K pools.
We need the ability, and **tools** get data, and to slice/dice data many different ways.  

There are never enough tools.
You must know which tools you need, which tools are available – and the different pieces of performance perspective available from each tool.
Finding something new, the cause of a problem, that sudden flash of insight – is a wonderful thing!!
There are many performance presentations and white papers available on our website. Please visit and help yourself.

Joel Goldstein is an internationally acknowledged performance expert, consultant, and instructor. He has more than four decades of data processing experience, and has been directly involved with the performance management and capacity planning of online database systems for most of his professional life.

He has over 20 years of experience addressing DB2 design, performance, and capacity planning issues, and has assisted many large national and international clients with their systems and applications. He is a frequent speaker at DB2 user groups and Computer Measurement Group meetings throughout North America, Europe, and the Far East. Joel has published more than two dozen articles on DB2 performance issues, is a VP of TRIDEX, served on the CMG national program committee for 16 years, was a member of the North American IDUG conference committee for five years, and is a past director of IDUG. He is president of Responsive Systems, a software and consulting firm specializing in strategic planning, capacity planning, and performance tuning of online database systems.