Common Misconceptions in the IBM i Performance Area

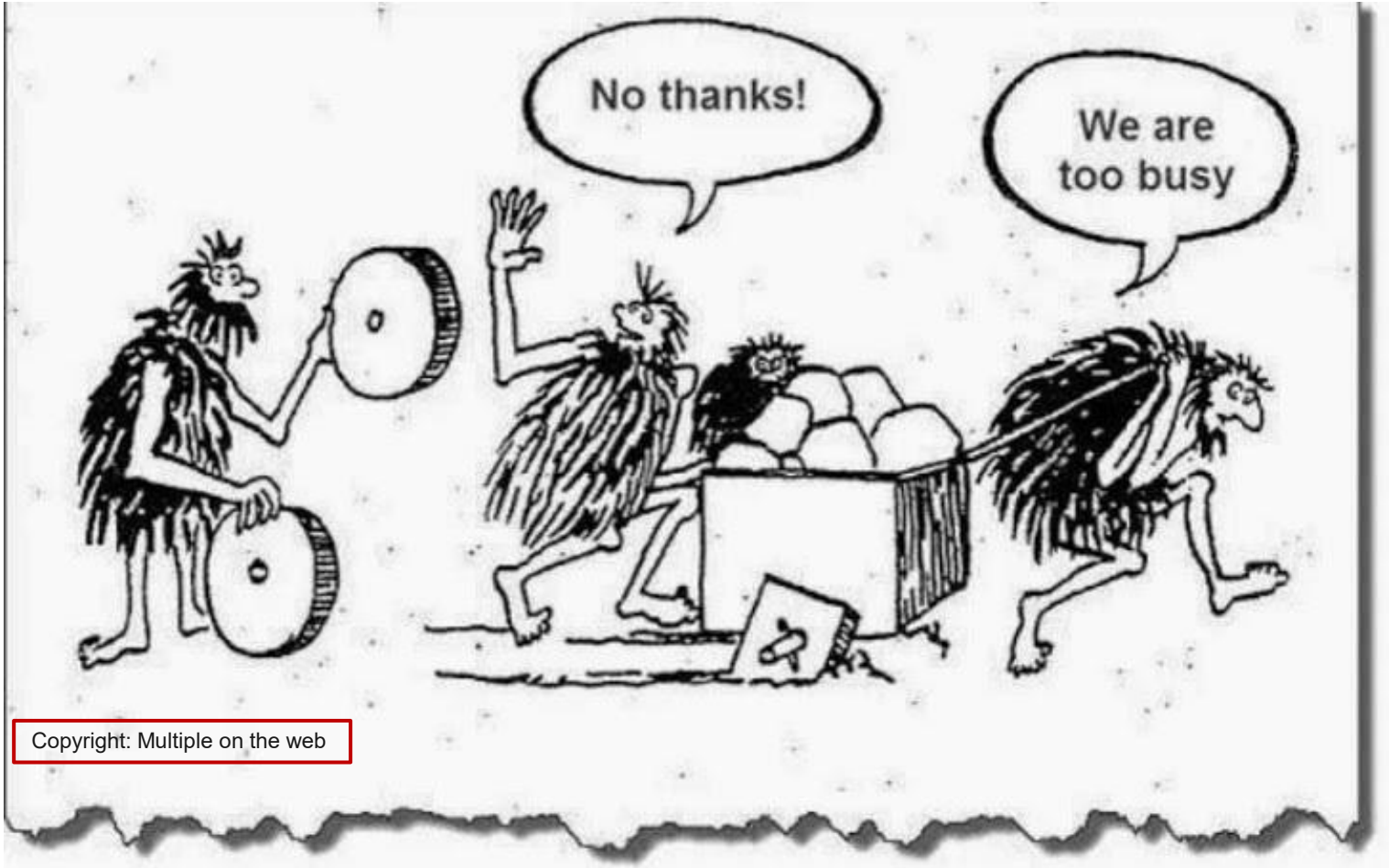
Morten Buur Rasmussen
IBM Expert Labs
Power Performance Specialist
Morten.Buur.Rasmussen@dk.ibm.com

Scope of this presentation

IBM i performance considerations and misconceptions in some areas

- You will hear nothing new, nothing about what is coming next
- Hear about where others have had of misunderstandings
- Talk about CPU utilization
- Cover some disk I/O's with differences on synchronous and asynchronous
- Disk page faults or not
- Differences on SQL access of the DB and native access of the DB
- A simple practical example showing some job characteristics and statistic
- Using iDoctor data combined with ACS

Working in Expert labs



Copyright: Multiple on the web

Memo to users

#You only need to read Memo to Users ones

The Memo to Users are updated, so always download the newest version before working with upgrades.
If jumping over versions, remember to read the newest version about the version you are dropping.
So, when upgrading from 7.4 to 7.6, you should read both Memo to Users 7.4 and 7.5

CPU

#The CPU utilization is too low for a job

Example
Having 20 VPs with 20 EC in an IBM i partition will limit a single threaded job to use maximum 1.57% CPU utilization on a Power 9 system with SMT8.
On Power 10 this is 1.49%

The maximum can only be achieved if the job is using the CPU all the time.

When more than 20 threads are running, the average CPU utilization will be lowered.
The jobs CPU utilization will be evenly distributed and always with a maximum of 1.57%

Processor Utilization

SMT8 on Power 9

Run Threads	∑Run PURR %
0	0%
1	31.4%
2	62.9%
3	73.2%
4	81.2%
5	86.6%
6	89.0%
7	95.2%
8	100%

Max thread CPU utilization at 20 VPs:
31.4/20 = 1.57%

SMT8 on Power 10

Run Threads	∑Run PURR %
0	0%
1	29.7%
2	59.4%
3	70.3%
4	81.2%
5	86.7%
6	92.3%
7	96.1%
8	100%

29.7/20 = 1.49%

SMT8 on Power 11

Run Threads	∑Run PURR %
0	0%
1	29.4%
2	58.9%
3	70.0%
4	81.1%
5	86.6%
6	92.1%
7	96.0%
8	100%

29.4/20 = 1.47%

#The CPU utilization can be high for a job without doing much

Example
Having 20 VPs with 1 EC in an IBM i partition will limit a single threaded job to use maximum 31.7% CPU utilization on a Power 9 system with SMT8.
On Power 10 this is 29.4%

The maximum CPU utilization the partition can drive is 20 * 100% = 2000%

So, always be aware of your configuration

High CPU utilization is not always high throughput

Full opens

Full opens

The full opens is one of the most common causes prohibiting you from scaling.
It is also prohibiting us from use all the benefits of Power systems with a very advanced cache hierarchy.

What we want is scaling and be able to scale.
So, adding hardware allows you to get more throughput

A full open needs to take place before we can access data in the DB. A part of a full open is to create an ODP (Open Data Path). This process does several important processes, like validating authority to the data.
The alternative to the full open is to keep the data path open and avoid having to re-create the ODP.
A common cause of full opens is the use of SETLR in RPG programs.

#A full open is a full open

Typically, we will address full opens by a measurement of how many full opens take place per second.
This is needed to address the issue and establish focus on sorting out the bottleneck.
This count per second can cause the believe that a full open is a full open, but much more is into a full open.

- The ODP can be small or large all depending on the complexity of the access
- SQL access and the ODP's created are typically more complex and more intelligent than an ODP created from native access. One of the reasons is the calculation of the access method.
- So, in general the SQL full opens takes longer time than the native full opens. This causes the number of possible SQL full opens for a job to be lower per second.

Full opens

It is often observed that a job can generate up to 2.000 to 4.000 full opens per second.

It depends, if one or more jobs are doing similar workload and causing full opens on the same files.

The best practice is to rewrite a little bit in the programs around the read of data to be able to re-use the ODP's.

For the SQL Full open topic, there are simple things that the SQL programmer can do to reduce full opens.

A part of our services in Expert Labs is to identity programs that causes most full opens and advise what program lines should be reviewed.

Wait Accounting

Wait Accounting

Wait Accounting is the patented technology built into the IBM® i operating system that tells you what a thread or task is doing when it appears that it is not doing anything.
When a thread or task is not executing, it is waiting.
Wait accounting, a concept exclusive to IBM i, is a very powerful capability for detailed performance analysis.

https://www.ibm.com/support/pages/ibm-i-wait-accounting

Wait Accounting

Elapsed time *is accounted for by*
harvesting ***Time*** *&* ***Count*** *values and categorizing them into 32 different* ***wait buckets***

... for every job, thread, task on the system

COUNT

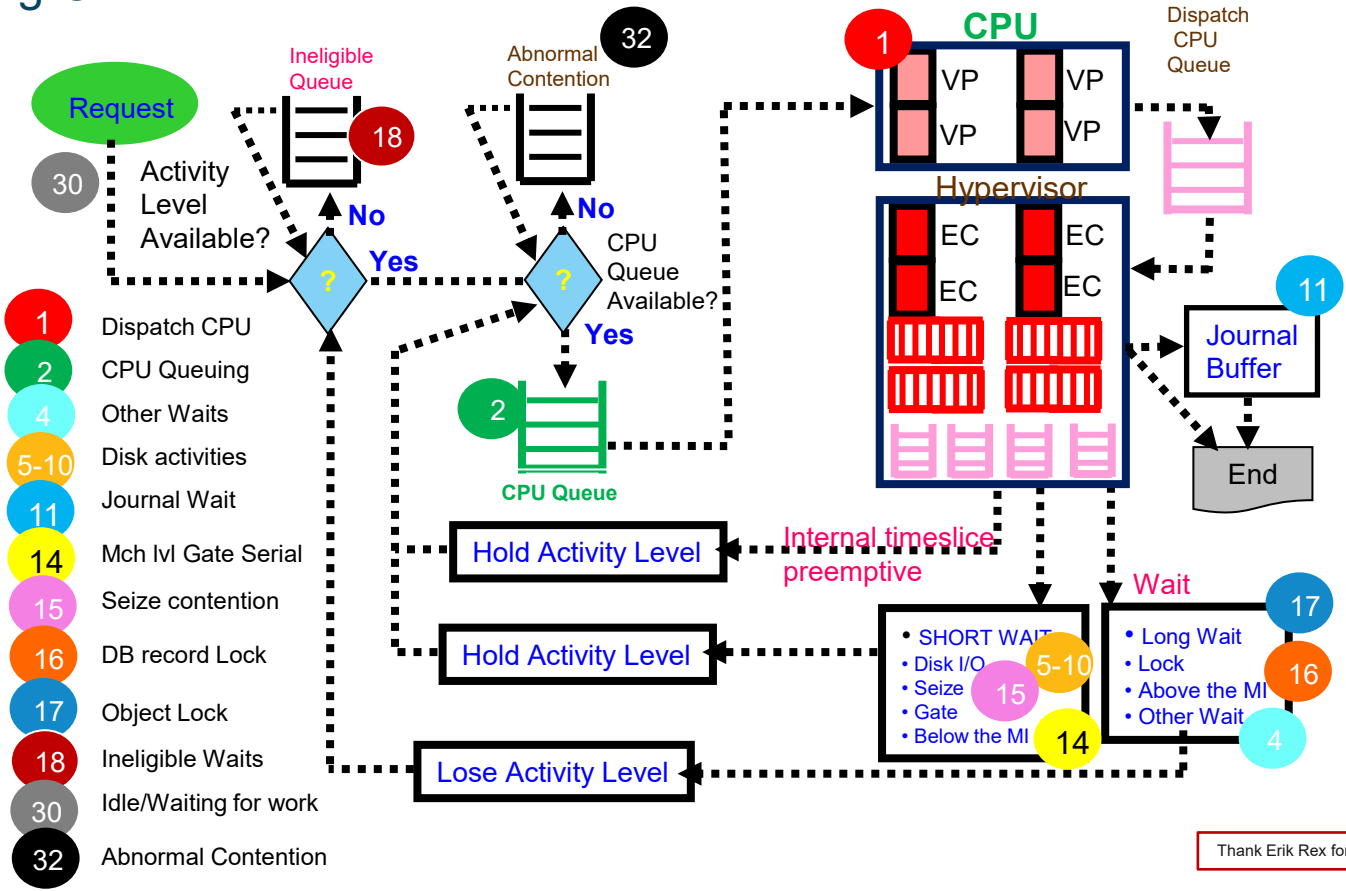
The number of times the thread or LIC task experienced a state covered by the wait bucket

TIME

The elapsed wall-clock time the thread or LIC task spent in a state covered by the wait bucket

Thank iDoctor team for the slide

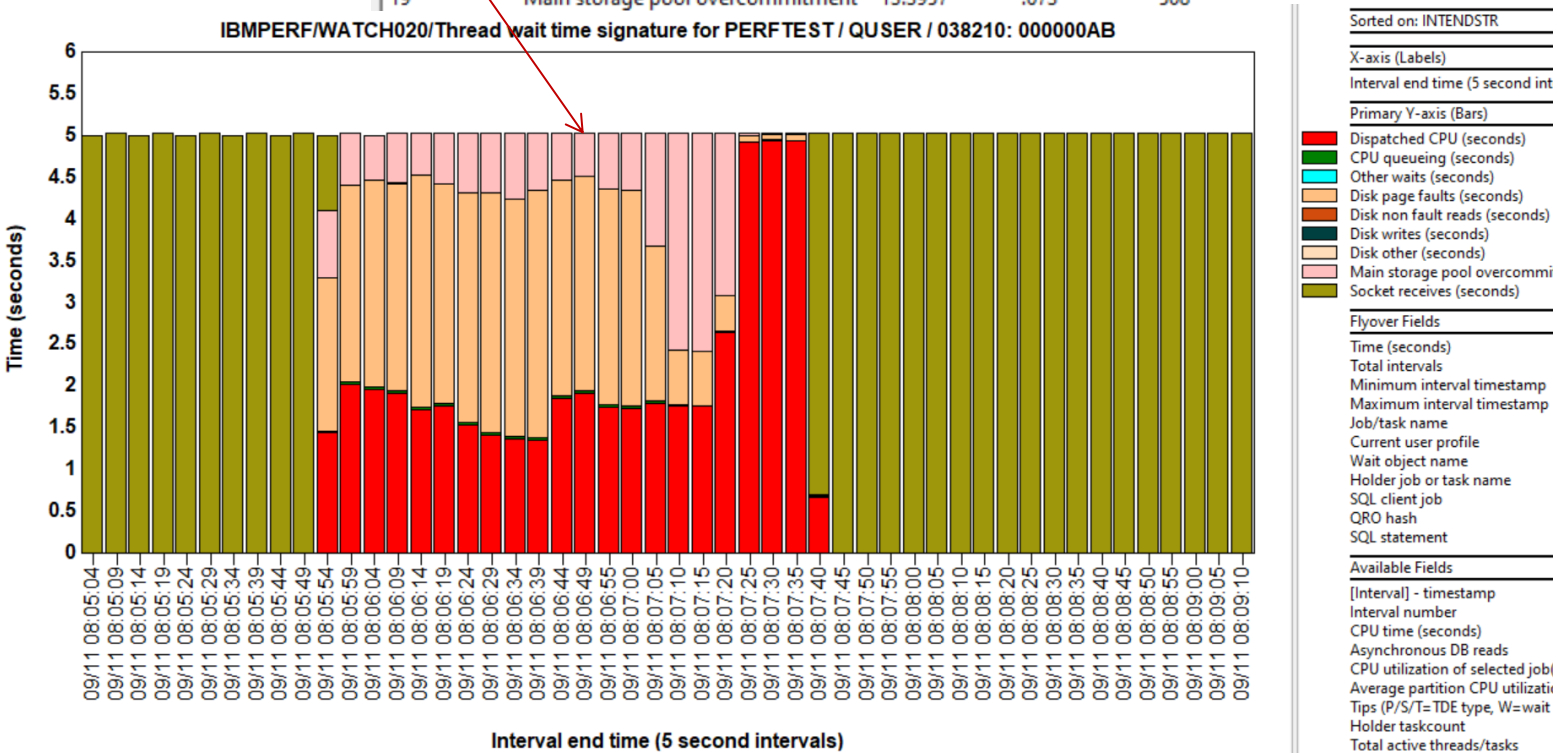
Processing Overview



Thank Erik Rex for the slide - RIP

Wait Accounting

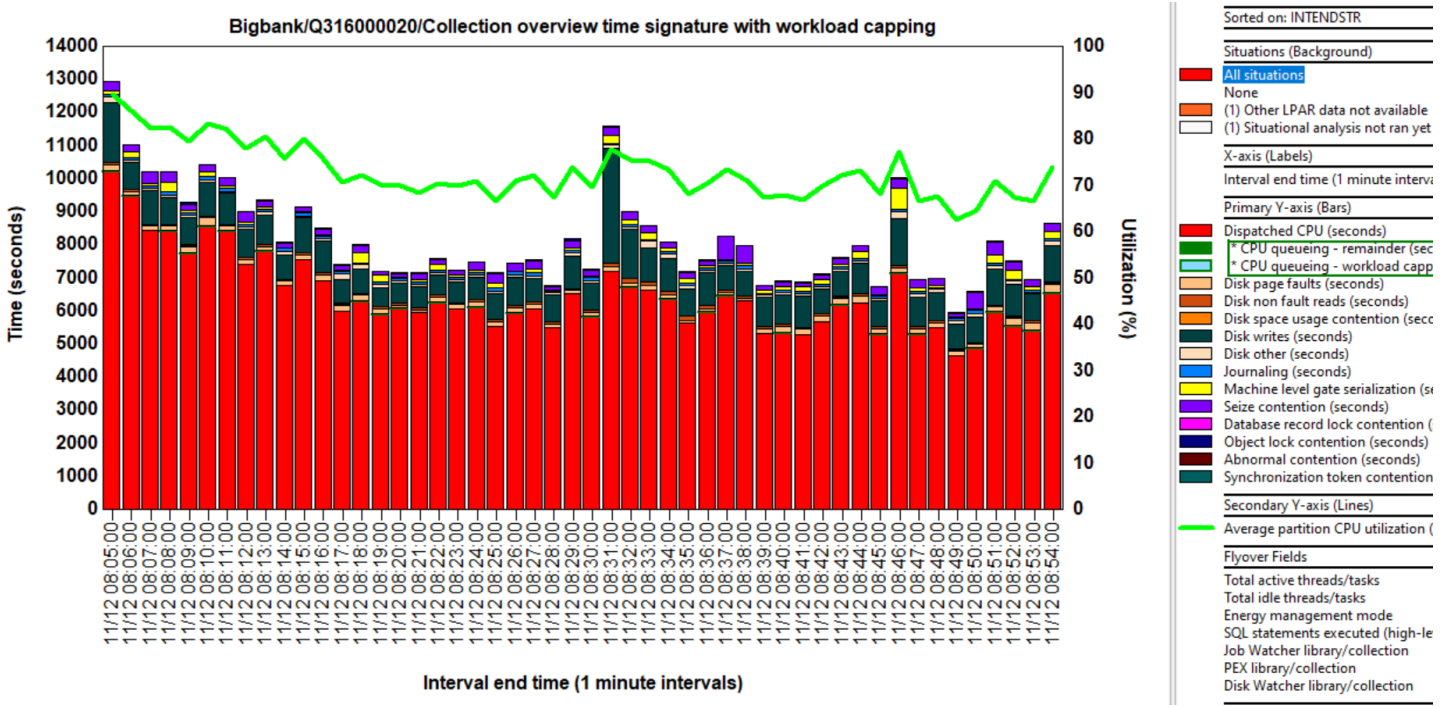
Wait bucket number	Wait bucket description	Percent of total time	Time (seconds)	Total occurrences
1	Dispatched CPU	34.7082	1.745	10,608
2	CPU queueing	.6153	.030	10,608
5	Disk page faults	51.2753	2.579	10,099
10	Disk other	.0055	0	1
19	Main storage pool overcommitment	13.3957	.673	508



Wait Accounting

All jobs accumulate

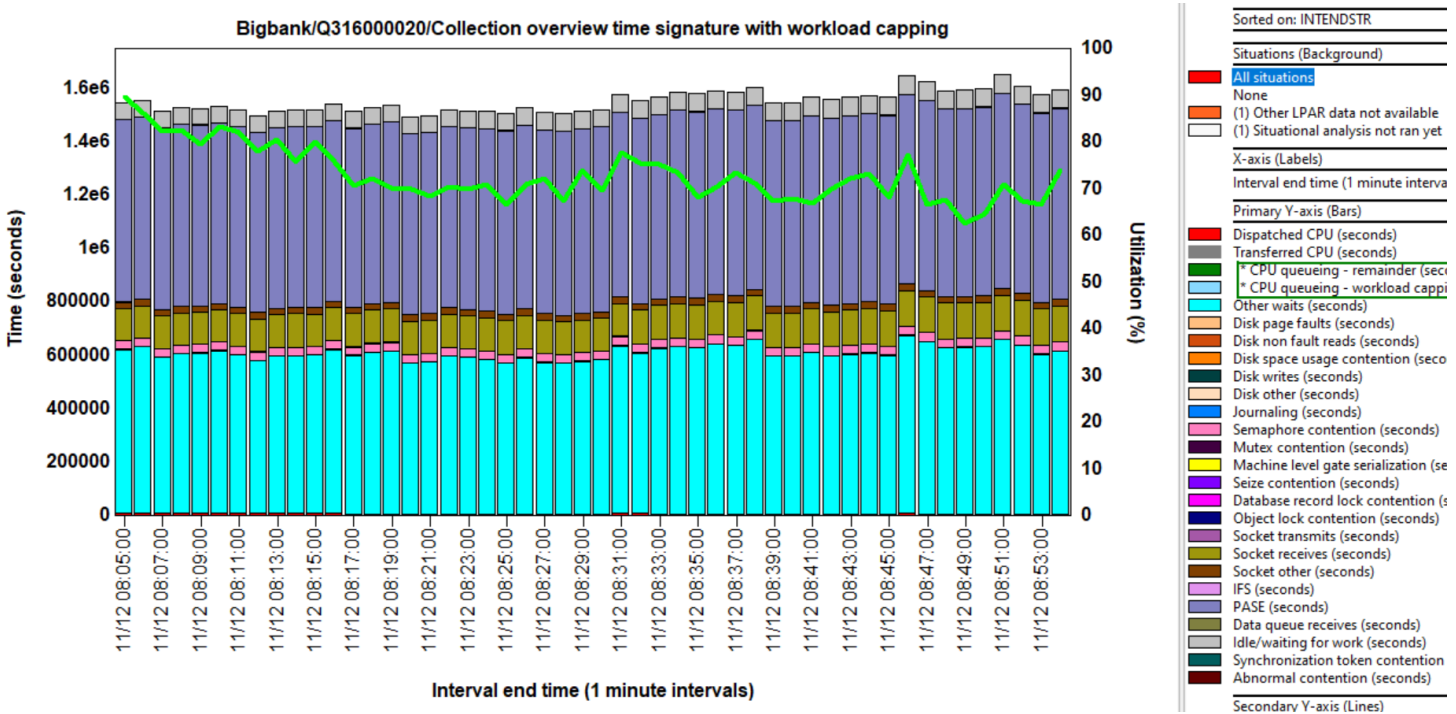
Showing important waits only



Wait Accounting

Idle waits covers most time

So, not shown



Upgrade to Power9/10 or Power11 makes your jobs faster?

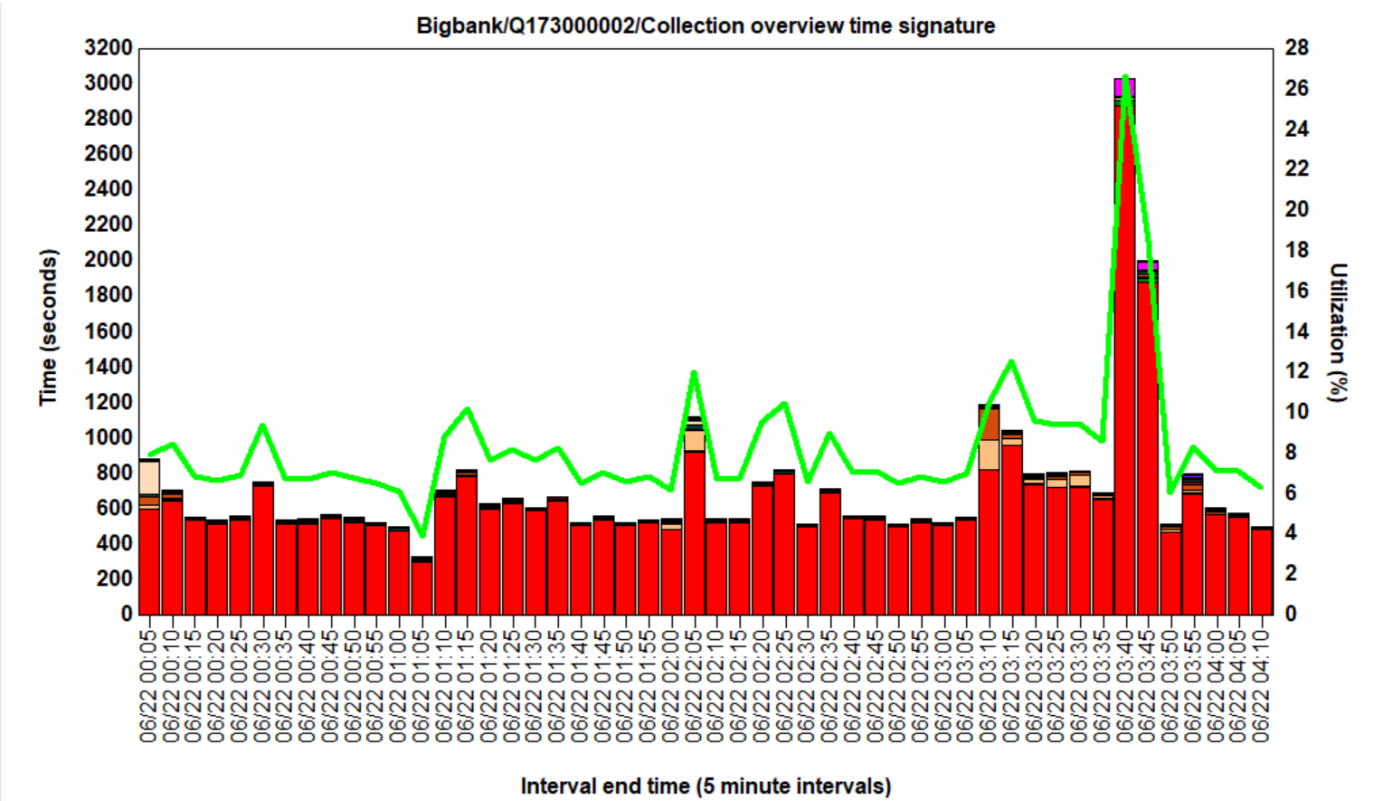
#Upgrades always makes your jobs faster

Sometimes we get complaints that upgrade to Power9 or Power10 do not give the expected better performance.
Sometimes upgrades and expectations are not calibrated
It happens that performance data has not been analyzed before upgrades, so advantages are simple assumed

Lets take an example

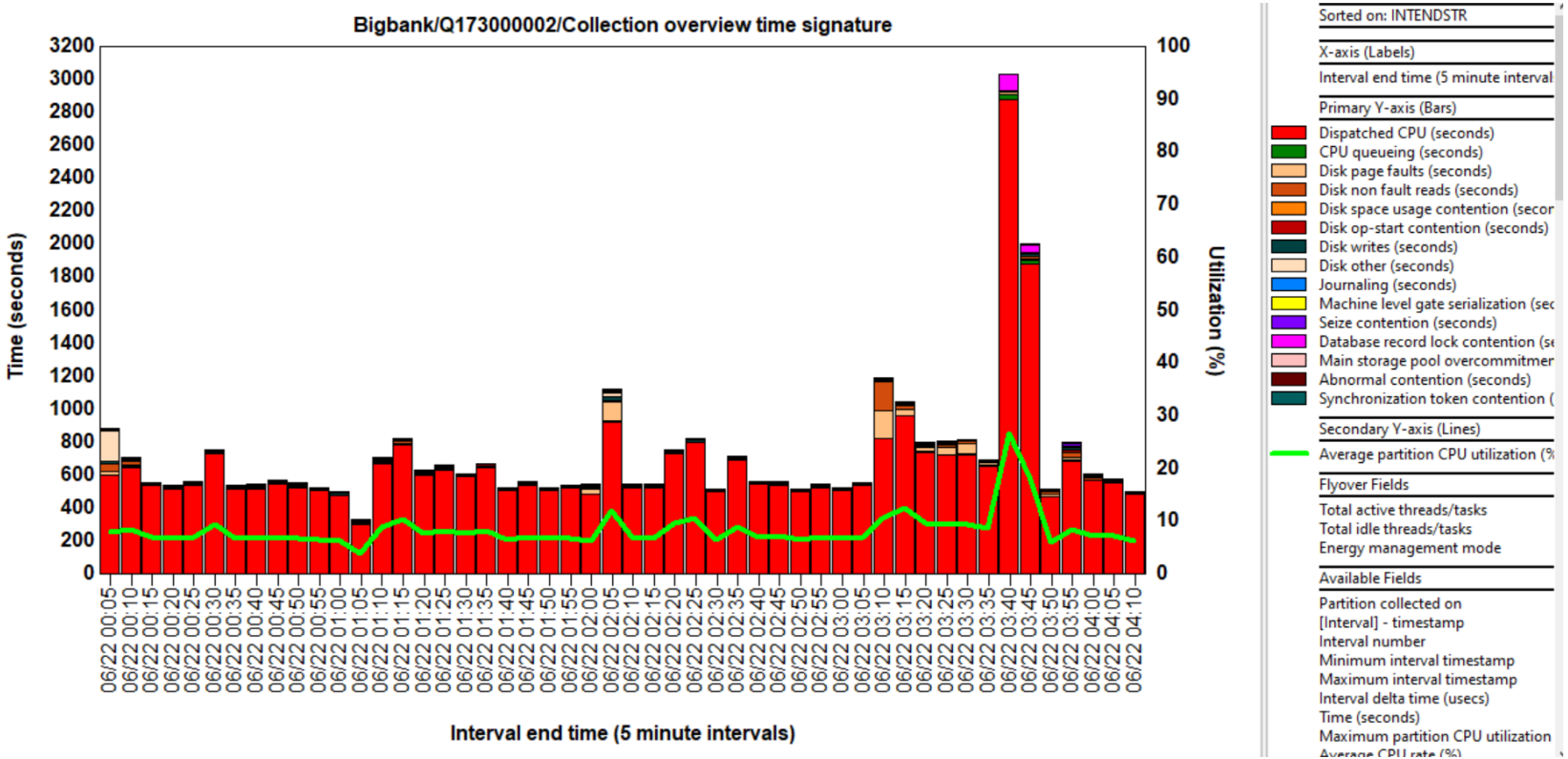
Complaint example where upgrade is not causing faster EOD

Power 9



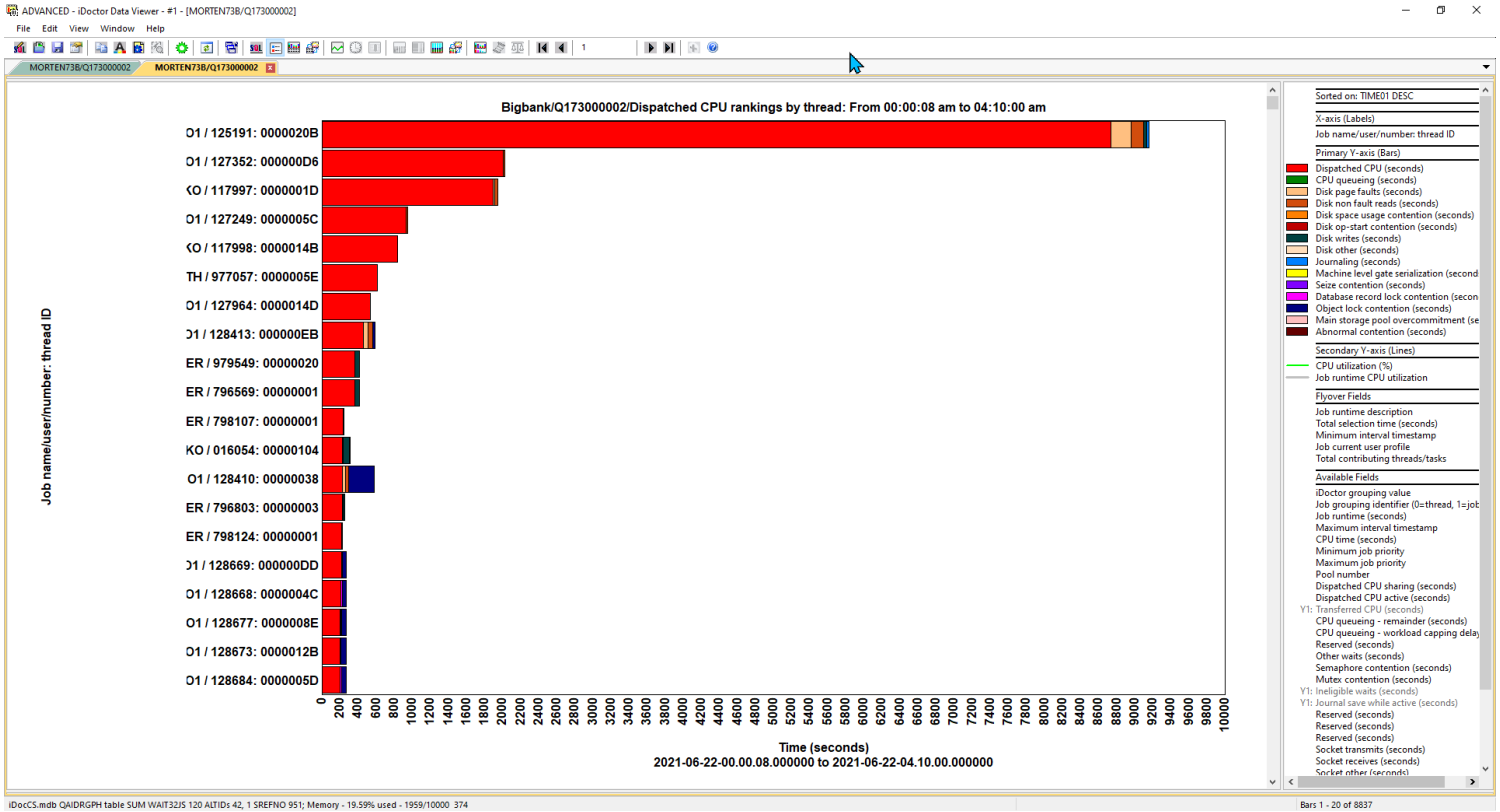
Complaint example where upgrade is not causing faster EOD

Adjust CPU utilizatior



Complaint example where upgrade is not causing faster EOD

Jobs running the 4 hours period. Only one job running most of the time.



#Upgrades always makes your jobs faster

The learning is often to suggest to run multiple jobs at the same time.
If not already implement, it should be considered, so the applications can scale with more workload.
The throughput on Power11 is higher that on Power10 and again higher than on Power9

One single thread do not show any big difference in performance when its only about CPU

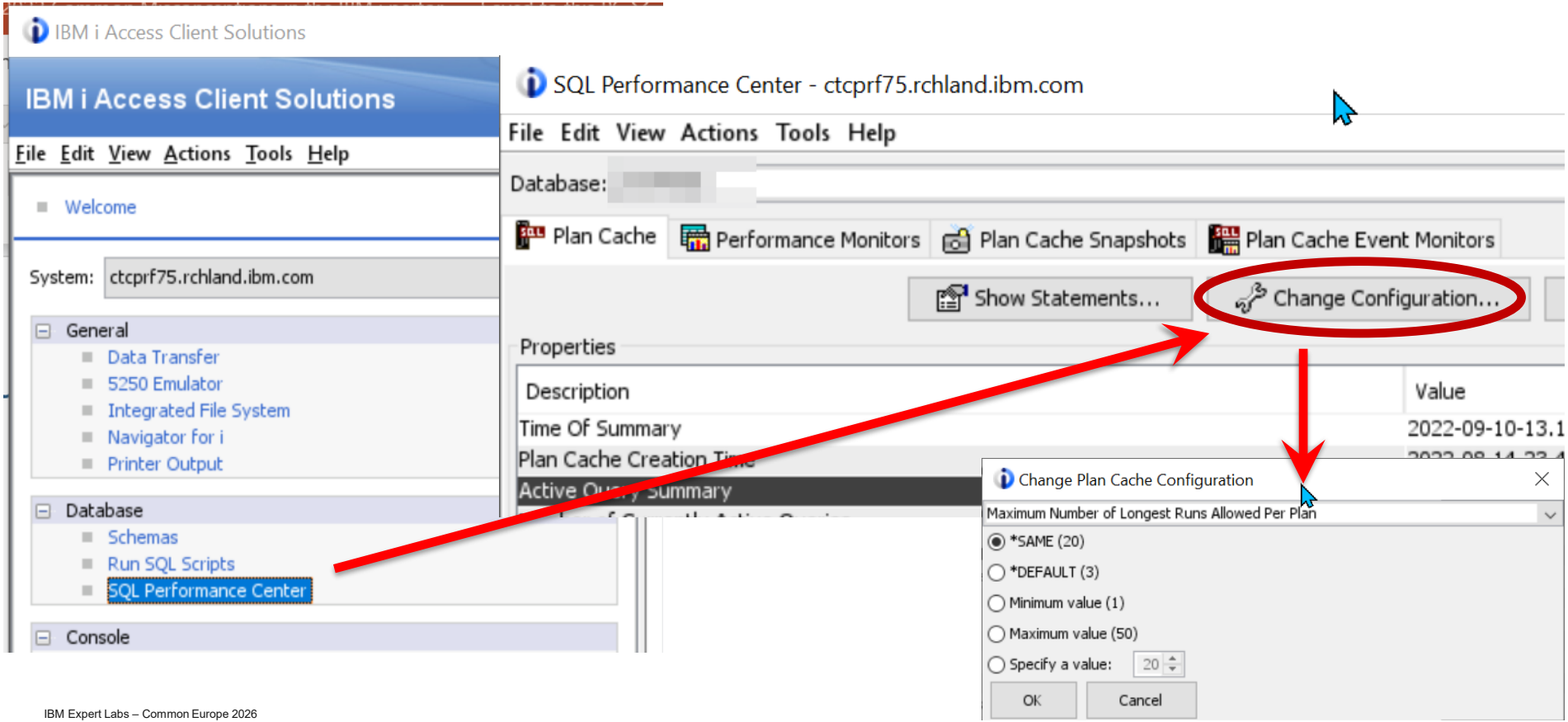
Introduction to viewing SQL requests

Introduction to viewing SQL requests

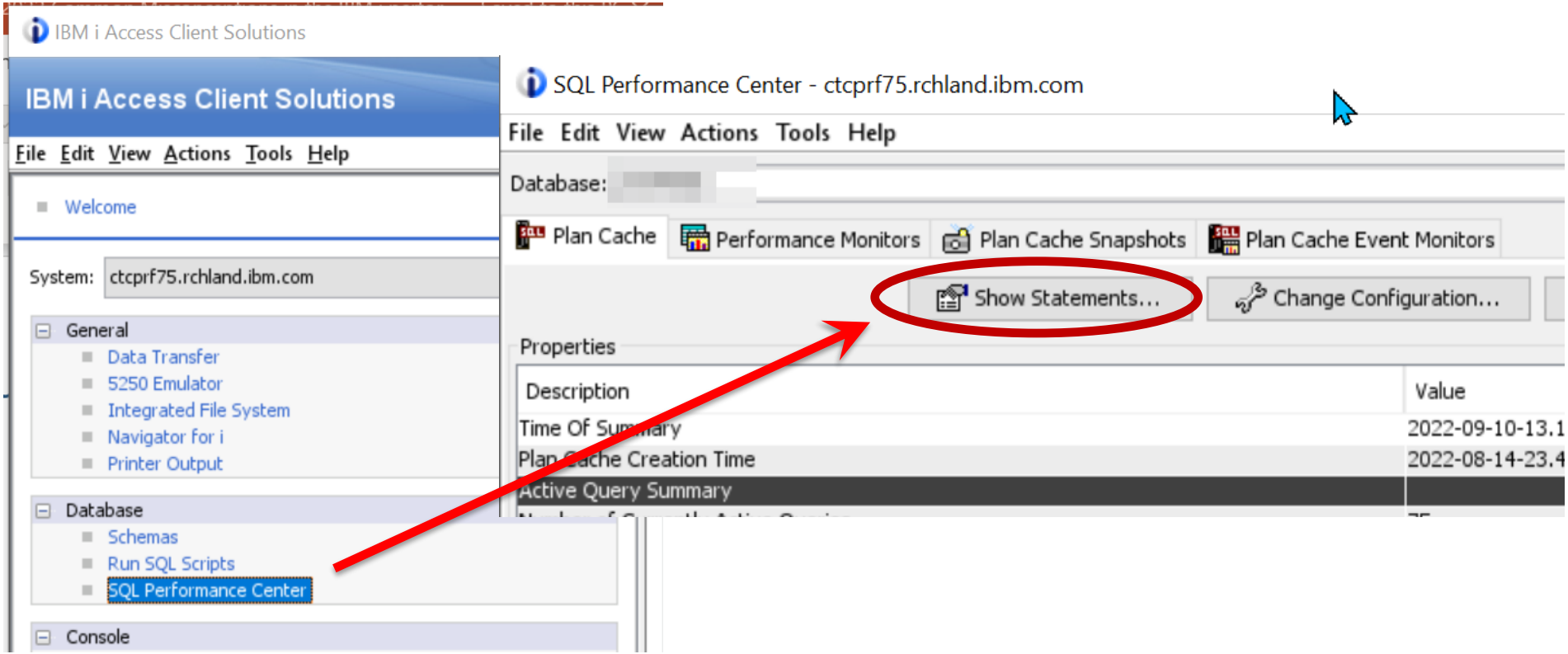
- IBM i has a Plan Cache that might contain the access plans for queries optimized and previously executed by SQE
- In 7.4/7.5/7.6 most queries goes the SQE route
- The Plan Cache contains run time statistic for already executed queries
- The Plan Cache is always active, so no need to start up any collection
- As it's a cache, its cleared by IPL and constantly maintained and cleaned up
- You get insight into the Plan Cache via ACS (Access Client Solution)

https://ibm.biz/IBMi_ACS

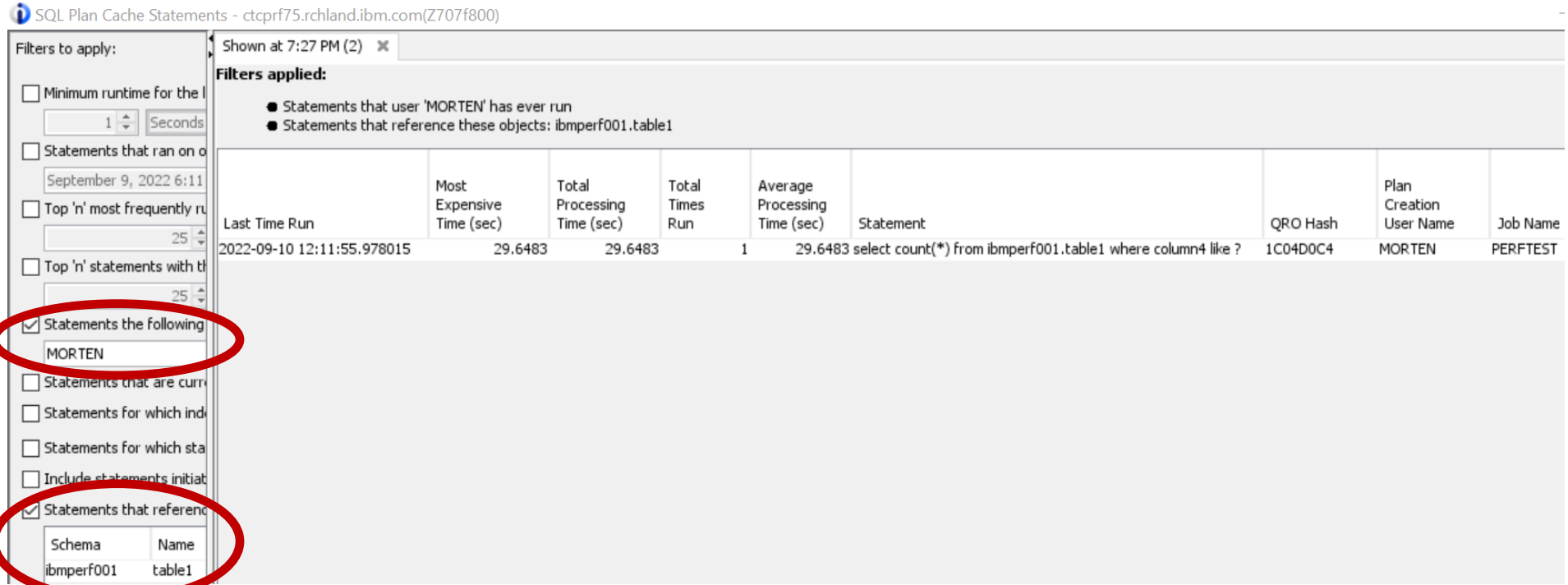
Introduction to viewing SQL requests – Number of longest runtime per plan



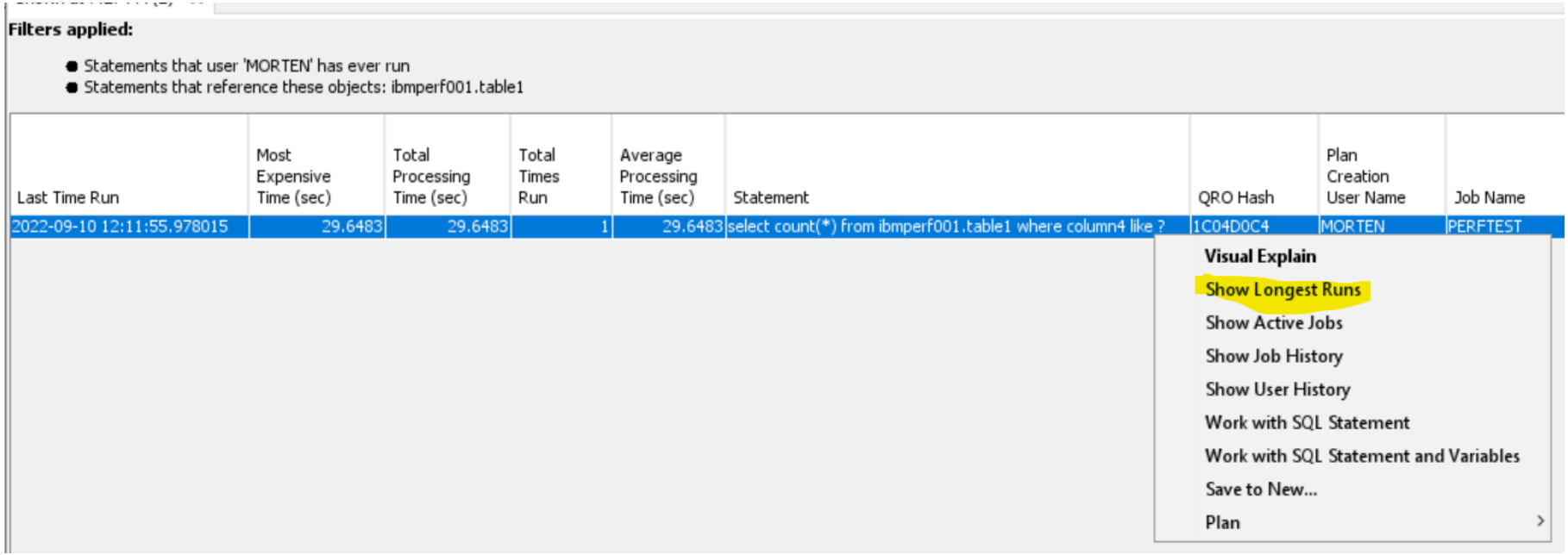
Introduction to viewing SQL requests



Introduction to viewing SQL requests – Good filtering



Introduction to viewing SQL requests – Show Longest Runs



The Show Longest Runs gives access to very detailed information about single executions using the plan

Example of CPU versus I/O's

Example of running SQL request in different memory settings

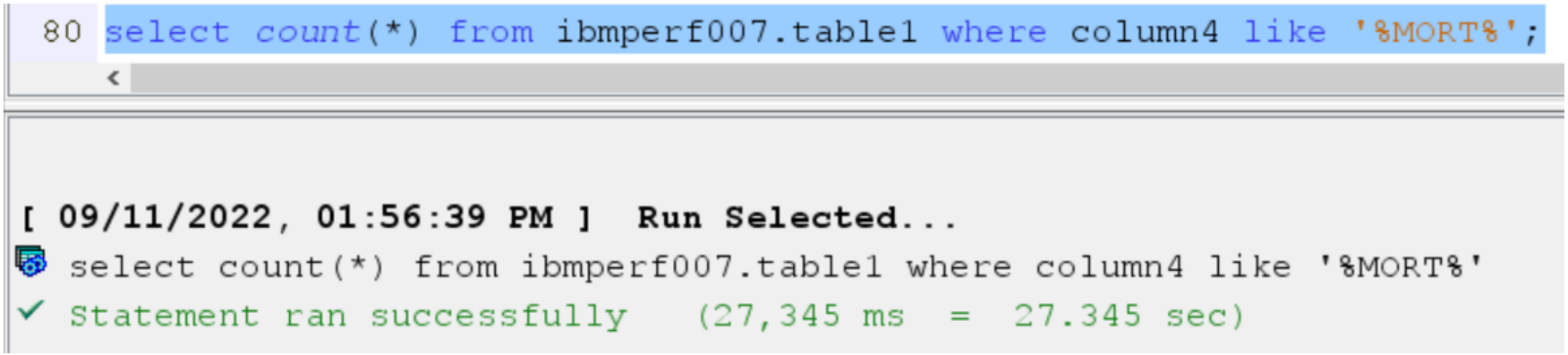
In the following example we are running a relative heavy SQL request that will need to go through 10.000.000 rows in a table (size 10GB):

```
select count(*) from ibmperf001.table1 where column4 like '%MORT%';
```

The column4 is defined at CHAR 1000
This is not a best practice, but it is often observed

The purpose of the example is to show a job accessing some amount of data doing I/O's and the example is not an example of best practice for constructing a SQL request.

Example of running SQL request in different memory settings in ACS

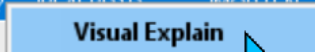


Run time from ACS involves laptop time, network time and any time before the request arrives at the DB.

The first run is in a memory pool having a size of 50MB

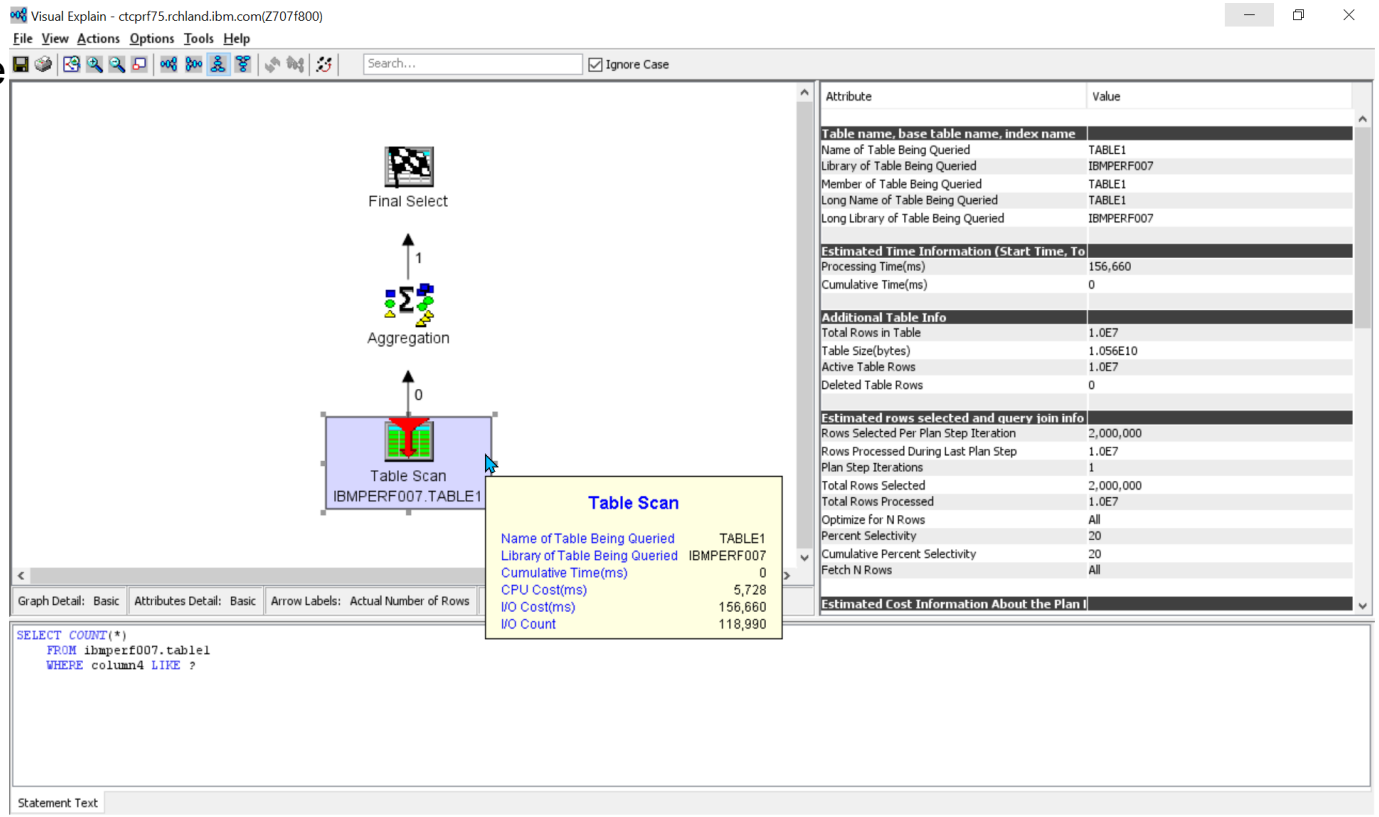
Example of running SQL request in different memory settings

Last Time Run	Most Expensive Time (sec)	Total Processing Time (sec)	Total Times Run	Average Processing Time (sec)	Statement	QRO Hash	Plan Creation User Nam
2022-09-11 06:56:44.990600	26.7802	26.7802	1	26.7802	select count(*) from ibmperf007.table1 where column4 like ?		
2022-09-11 06:53:57.135032	0.0147	0.0177	3	0.0059	INSERT INTO QTEMP/PROCVER (SELECT CAST(REPEAT(?,		



Example of running SQL request in different memory settings

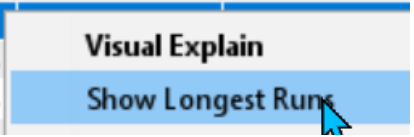
Numbers in flyover are estimated numbers



Example of running SQL request in different memory settings

Actual numbers from Show Longest Runs

	Average Processing Time (sec)	Statement	QRO Hash	Plan Creation User Name
1	26.7802	select count(*) from ibmperf007.table1 where column4 like ?		
3	0.0059	INSERT INTO QTEMP/PROCVER (SELECT CAST(REPEAT(?, ...		
1	0.0086	select REPEAT(?, CALLLVL) CONCAT CALLLVL AS CALLLVL, LI..		



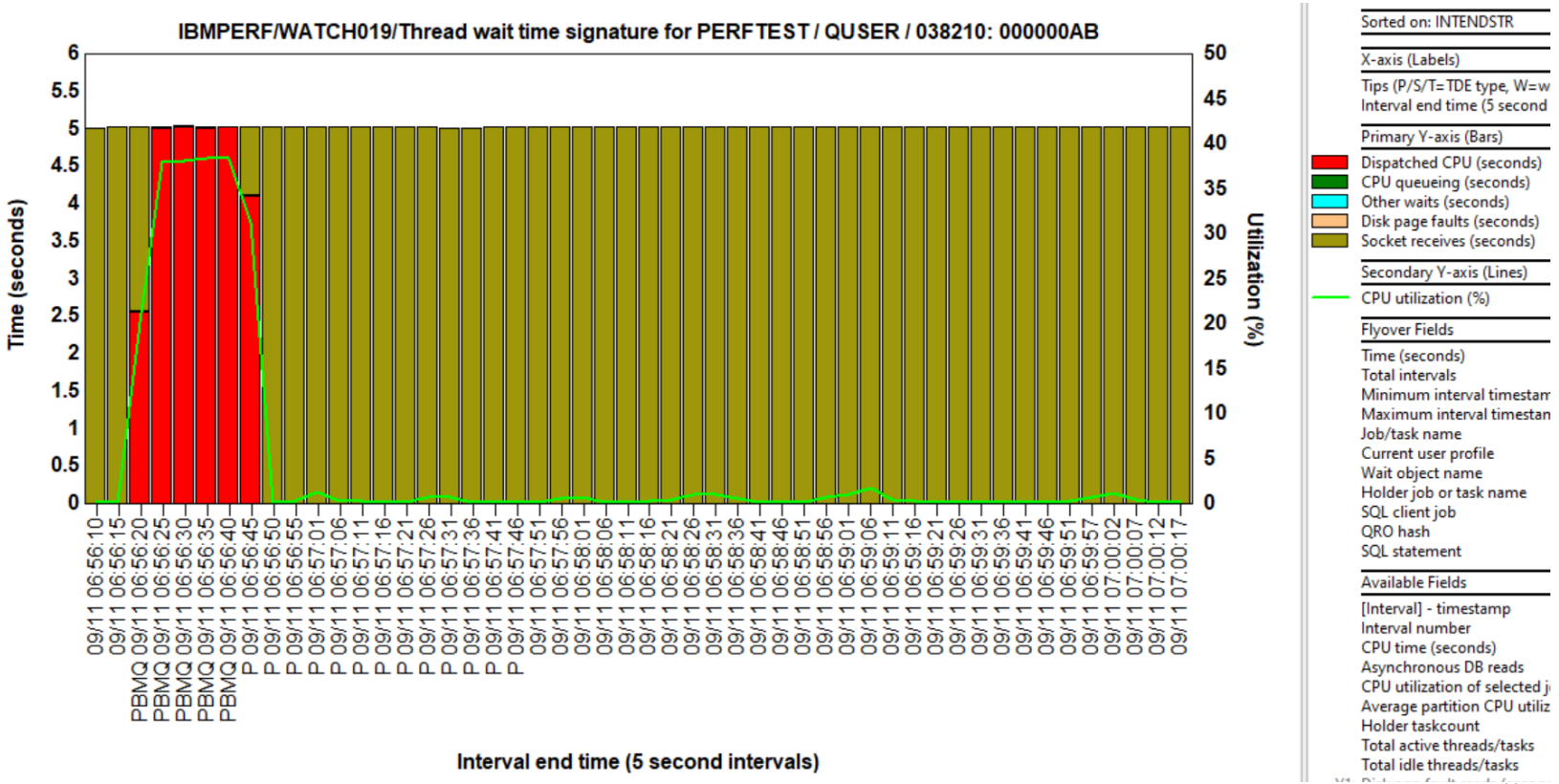

```
SELECT COUNT(*)
    FROM ibmperf007.table1
    WHERE column4 LIKE ?
```

Time Run	Processing Time (sec)	Records Selected	User Name	Job Name	Job User	Job Number	CPU Time (sec)	Synchronous Database Reads	Asynchronous Database Reads	Synchronous Non-database Reads	Asynchronous Non-database Reads	Page Faults	Cached Result Used
2022-09-11 06:56:44.990600	26.7802	1	MORTEN	PERFTEST	QUSER	038210	7.7799	5	88466	2	0	7	No

Mainly asynchronous reads

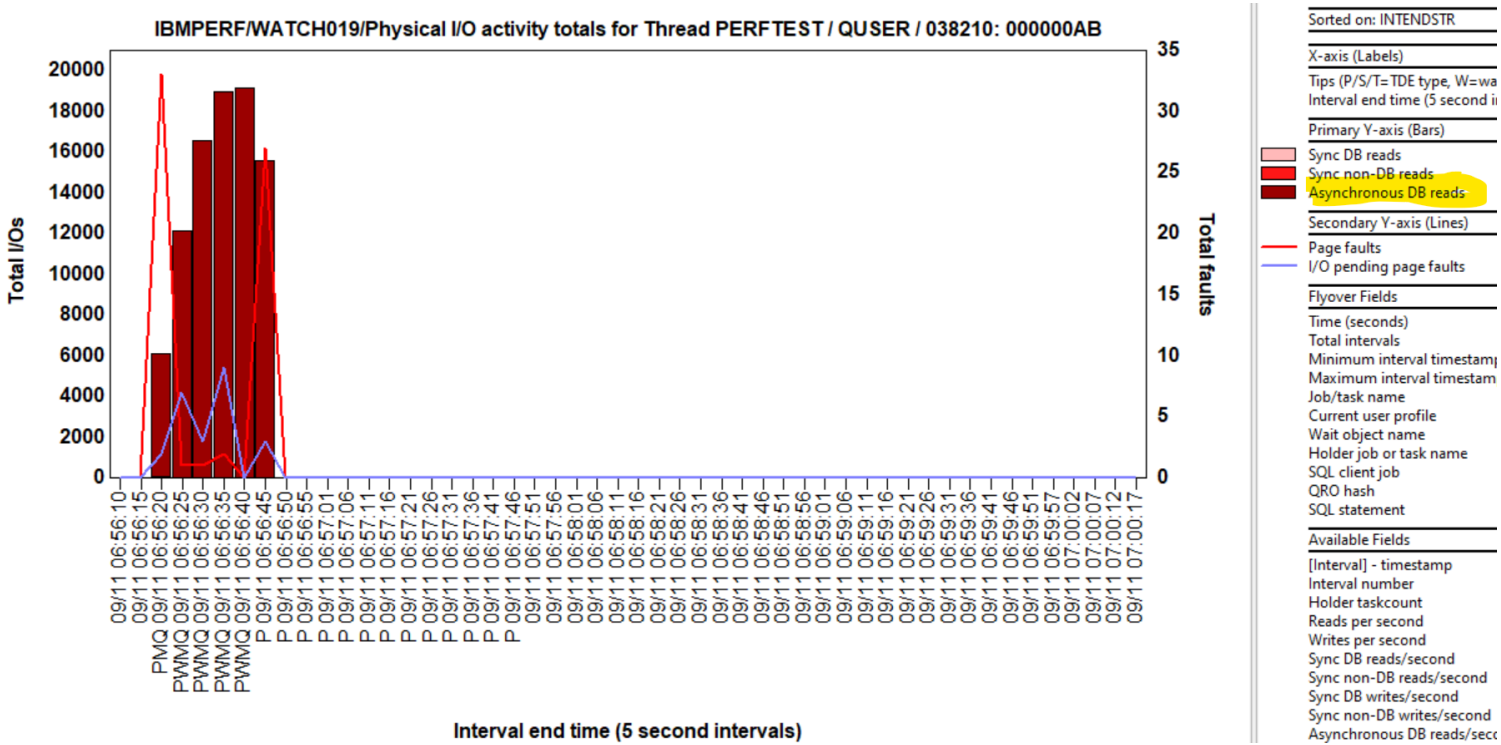
→ This leads to the jobs wait time signature under wait accounting

Example of running SQL request in different memory settings



Example of running SQL – Lots of I/O's – Asynchronous DB reads

In addition to wait accounting information, several I/O stats are available

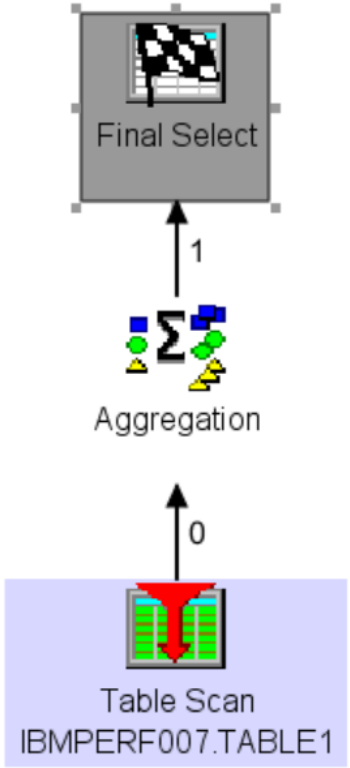


Example of running SQL – Fair share of memory

Back to Visual Explain

Average Active Used is typical defined as highest number of active users in the memory pool and 5

Memory pool size 50MB. The fair share is 10MB



Attribute	Value
Environment Information for SQL Statement	
Memory Pool Size	5.242E7
Memory Pool ID	6
Share of Memory Available(bytes)	1.048E7
Average Active Used	5
Maximum Active Threads Allowed in Pool	100
Average Active In The Pool	1
Number of Processors	4 (32)
Workload Group Specified	No
Processor Units	1
Max Environmental Parallel Degree	1
Date format	ISO
Date separator	-
Time format	ISO
Time separator	:
Decimal point	.
Sort Sequence Library	None
Sort Sequence Table	None
Language ID	ENU
Parallel degree setting	*NONE
Maximum number of tasks	Not Available
Rollback Hold Option	Cursor Position May B
Concurrent Access Resolution Specified	WAIT FOR OUTCOME
File Wait Time	30
Allow Temporary Index	Yes

Example of running SQL request in different memory settings

Testing is difficult because the DB, the Power system and the storage is very clever

When testing you can clear your memory pool and reconnect

Repeat the test and check that you have similar result.

The next run makes use of the same plan and can be viewed under Longest Runs

```
SELECT COUNT(*)
    FROM ibmperf007.table1
    WHERE column4 LIKE ?
```

Time Run	Processing Time (sec)	Records Selected	User Name	Job Name	Job User	Job Number	CPU Time (sec)	Synchronous Database Reads	Asynchronous Database Reads	Synchronous Non-database Reads	Asynchronous Non-database Reads	Page Faults	Cached Result Used
2022-09-11 07:50:40.158790	26.8421	1	MORTEN	PERFTEST	QUSER	038210	7.7822	37	88537	3	0	40	No
2022-09-11 06:56:44.990600	26.7802	1	MORTEN	PERFTEST	QUSER	038210	7.7799	5	88466	2	0	7	No

Example of running SQL request in different memory settings

Next step: Change the memory pool size from 50MB to 1GB

Not any important difference in Processing Time. A new plan is created, and the result can be viewed under Longest Runs:

```
SELECT COUNT(*)
    FROM ibmperf007.table1
    WHERE column4 LIKE ?
```

Time Run	Processing Time (sec)	Records Selected	User Name	Job Name	Job User	Job Number	CPU Time (sec)	Synchronous Database Reads	Asynchronous Database Reads	Synchronous Non-database Reads	Asynchronous Non-database Reads	Page Faults	Cached Result Used
2022-09-11 07:54:45.088140	29.3349	1	MORTEN	PERFTEST	QUSER	038210	8.5680	1	80999	0	0	1	No

The number of asynchronous reads is 8,000 less.

A new plan is typically created when the fair share of memory changes with 50%

Example of running SQL – Very small partition of memory

Next step - Change memory pool to 1MB → Fair share is 0.2MB

```
[ 09/18/2022, 04:40:55 PM ]  Run Selected...
select count(*) from ibmperf008.table1 where column4 like '%MORT%'
Statement ran successfully   (111,571 ms  =  1.86 min)
```

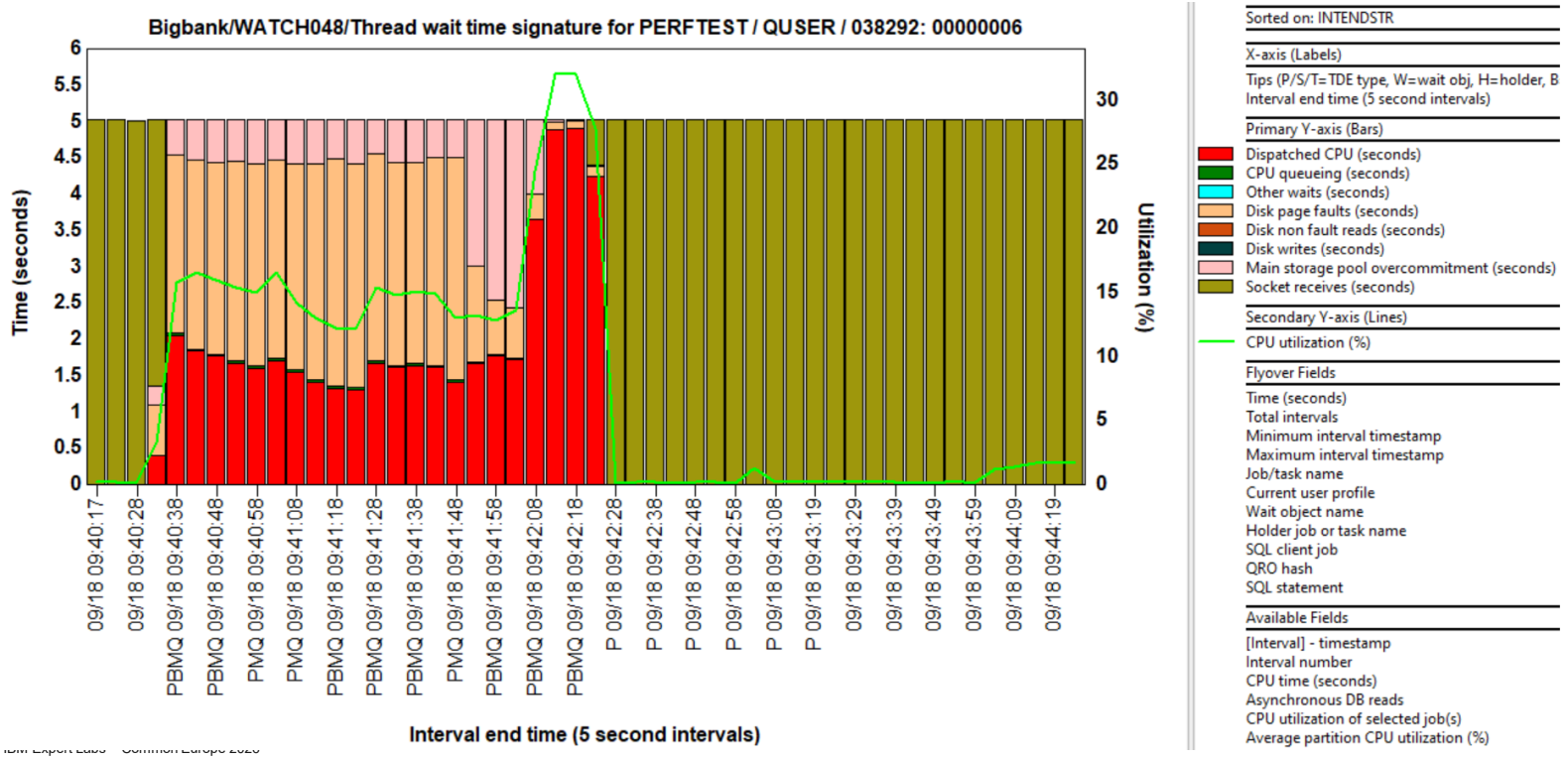
```
SELECT COUNT(*)
    FROM ibmperf008.table1
    WHERE column4 LIKE ?
```

Time Run	Processing Time (sec)	Records Selected	User Name	Job Name	Job User	Job Number	CPU Time (sec)	Synchronous Database Reads	Asynchronous Database Reads	Synchronous Non-database Reads	Asynchronous Non-database Reads	Page Faults	Cached Result Used
2022-09-18 09:42:23.043298	111.0691	1	MORTEN	PERFTEST	QUSER	038292	14.8493	106288	755727	30167	4250	136455	No

Very different activity: Many more asynchronous reads and many synchronous reads

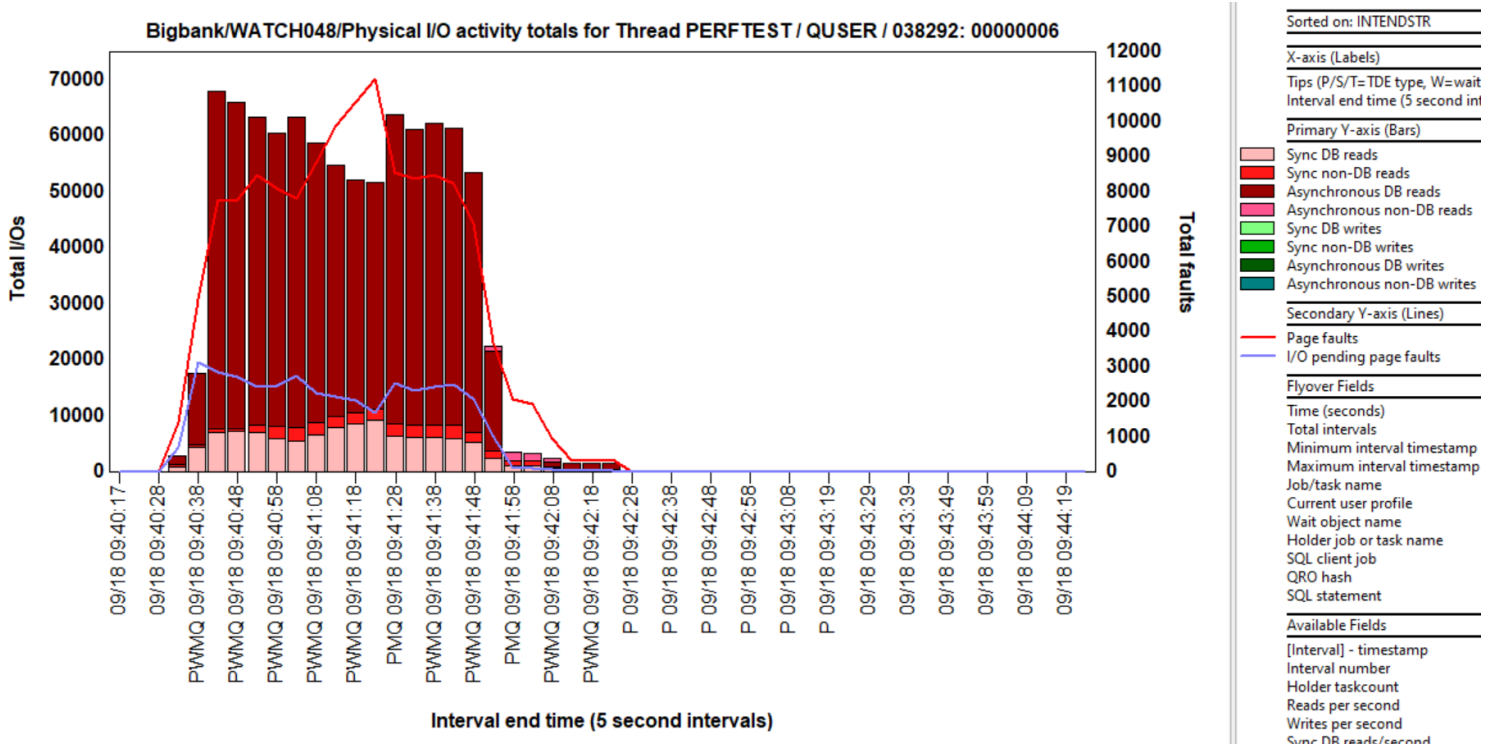
Example of running SQL - Very small partition of memory

Clearly very little memory available



Example of running SQL - Very small partition of memory

Heavily I/O activity

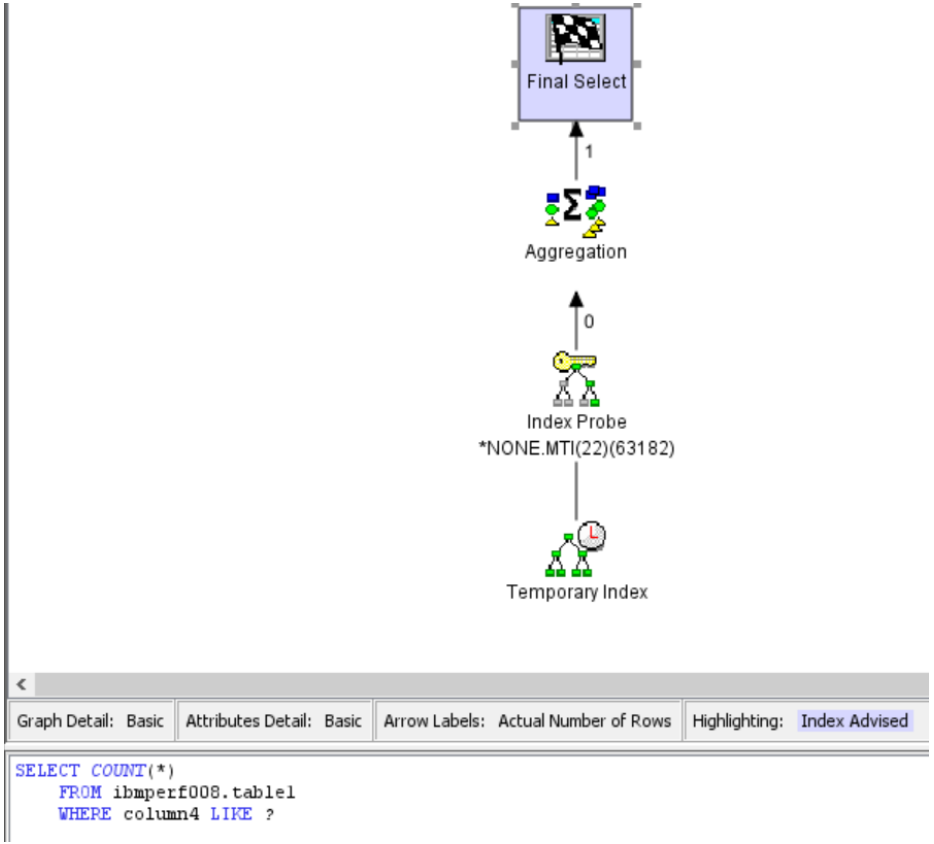


Example of running SQL - Very small portition of memory

More efficient to have an index, so the DB opted for that.

Index created is a Maintained Temporary Index (MTI)

Environment Information for SQL Statement	
Memory Pool Size	1,048,576
Memory Pool ID	6
Share of Memory Available(bytes)	209,715
Average Active Used	5
Maximum Active Threads Allowed in Pool	100
Average Active In The Pool	1
Number of Processors	4 (32)



Example of running SQL - Rerun the SQL to verify it using the MTI

Here the memory pool is cleared, and the job has re-connected like in all other situations

```
[ 09/18/2022, 04:59:39 PM ]  Run Selected...
select count(*) from ibmperf008.table1 where column4 like '%MORT%'
Statement ran successfully   (18,247 ms  =  18.247 sec)
```

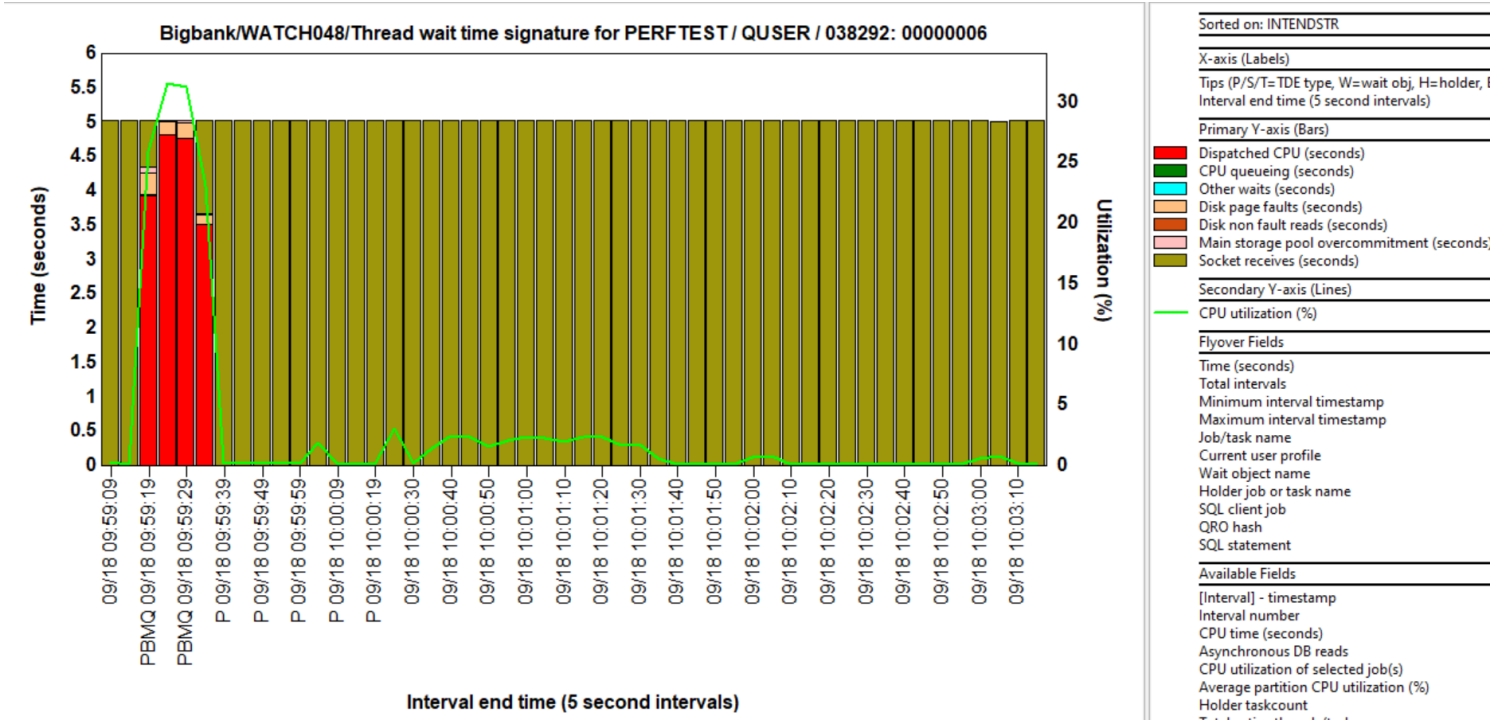
```
SELECT COUNT(*)
    FROM ibmperf008.table1
    WHERE column4 LIKE ?
```

Time Run	Processing Time (sec)	Records Selected	User Name	Job Name	Job User	Job Number	CPU Time (sec)	Synchronous Database Reads	Asynchronous Database Reads	Synchronous Non-database Reads	Asynchronous Non-database Reads	Page Faults	Cached Result Used
2022-09-18 09:42:23.043298	111.0691	1	MORTEN	PERFTEST	QUSER	038292	14.8493	106288	755727	30167	4250	136455	No
2022-09-18 09:59:33.314621	17.8079	1	MORTEN	PERFTEST	QUSER	038292	5.5257	920	4202	238	0	1158	No

Same plan used, but the MTI exists, so much less I/O's

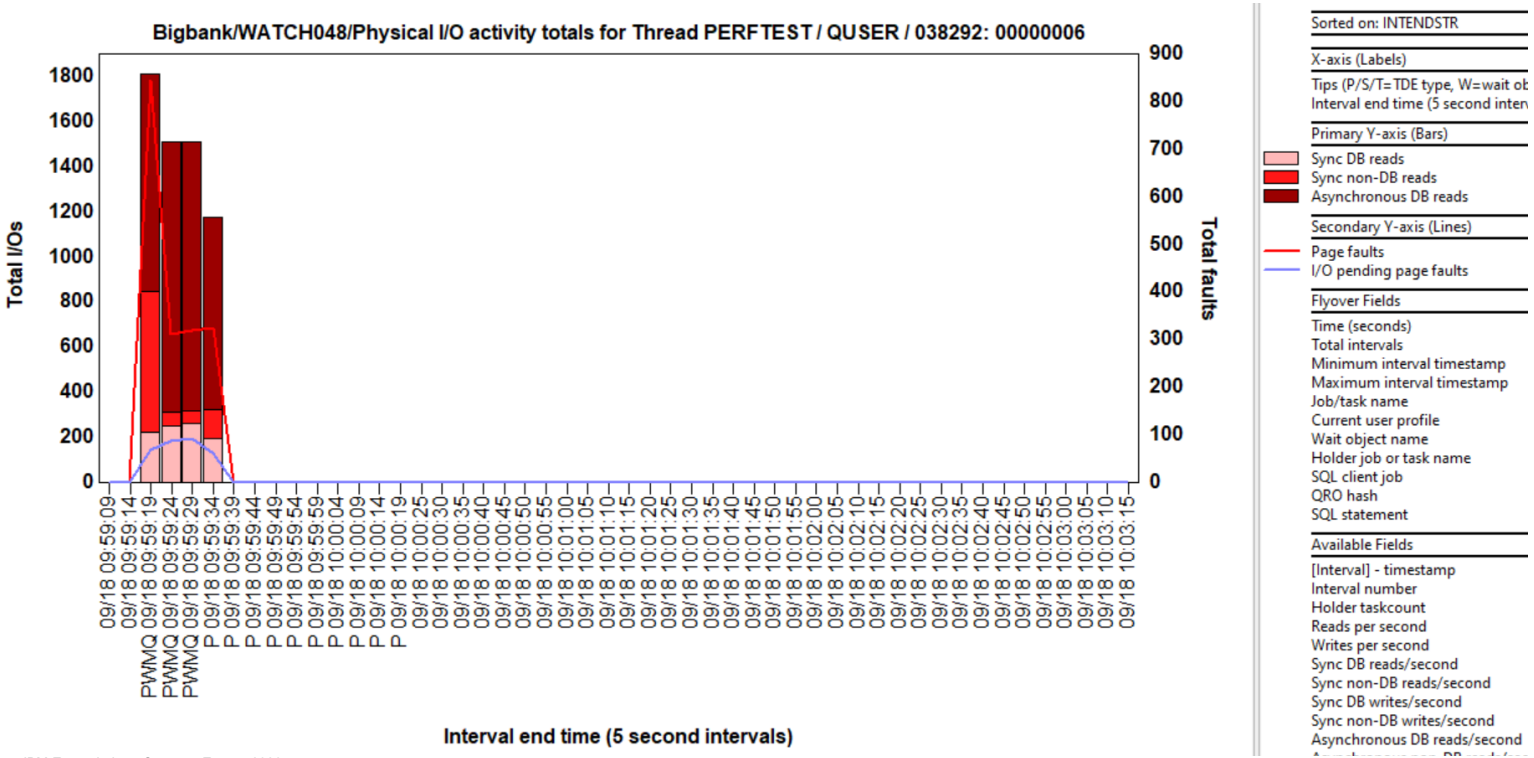
Example of running SQL - Very small partition of memory

The time signature shows most wait time is in CPU



Example of running SQL - Very small partition of memory

The I/O's shows some synchronous DB and non-DB reads



Example of running SQL – Permanent index and more memory

In case the query was important, the best way to keep constant good performance is creating a permanent index:

CREATE INDEX IBMPERF008.TABLE1_IDX ON IBMPERF008.TABLE1 (COLUMN4)

This causes the query to run faster:

```
[ 09/18/2022, 06:38:55 PM ]  Run Selected...
select count(*) from ibmperf008.table1 where column4 like '%MORT%'
Statement ran successfully   (20,983 ms  =  20.983 sec)
```

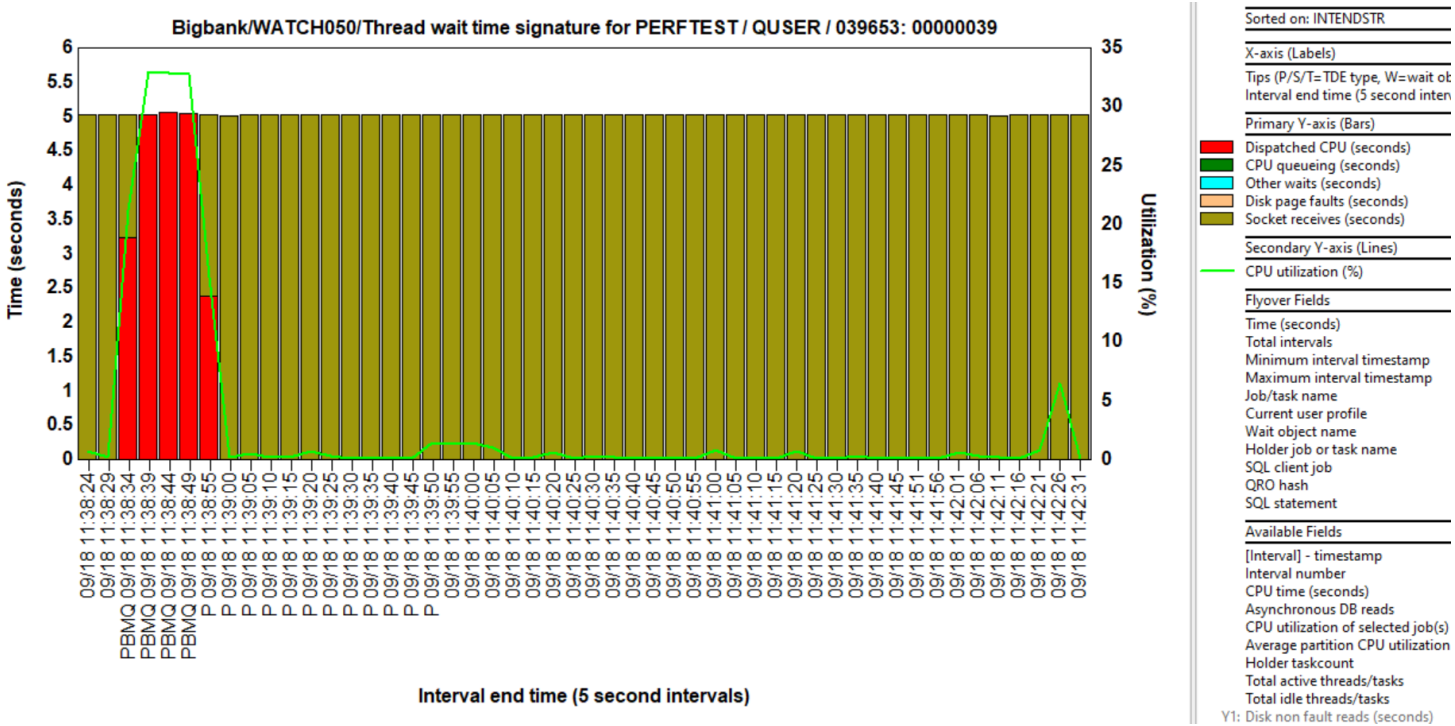
Mainly asynchronous reads when enough memory available (50MB):

```
SELECT COUNT(*)
    FROM ibmperf008.table1
    WHERE column4 LIKE ?
```

Time Run	Processing Time (sec)	Records Selected	User Name	Job Name	Job User	Job Number	CPU Time (sec)	Synchronous Database Reads	Asynchronous Database Reads	Synchronous Non-database Reads	Asynchronous Non-database Reads	Page Faults	Cached Result Used
2022-09-18 11:38:52.362075	20.7347	1	MORTEN	PERFTEST	QUSER	039653	6.7427	3	2439	0	0	3	No

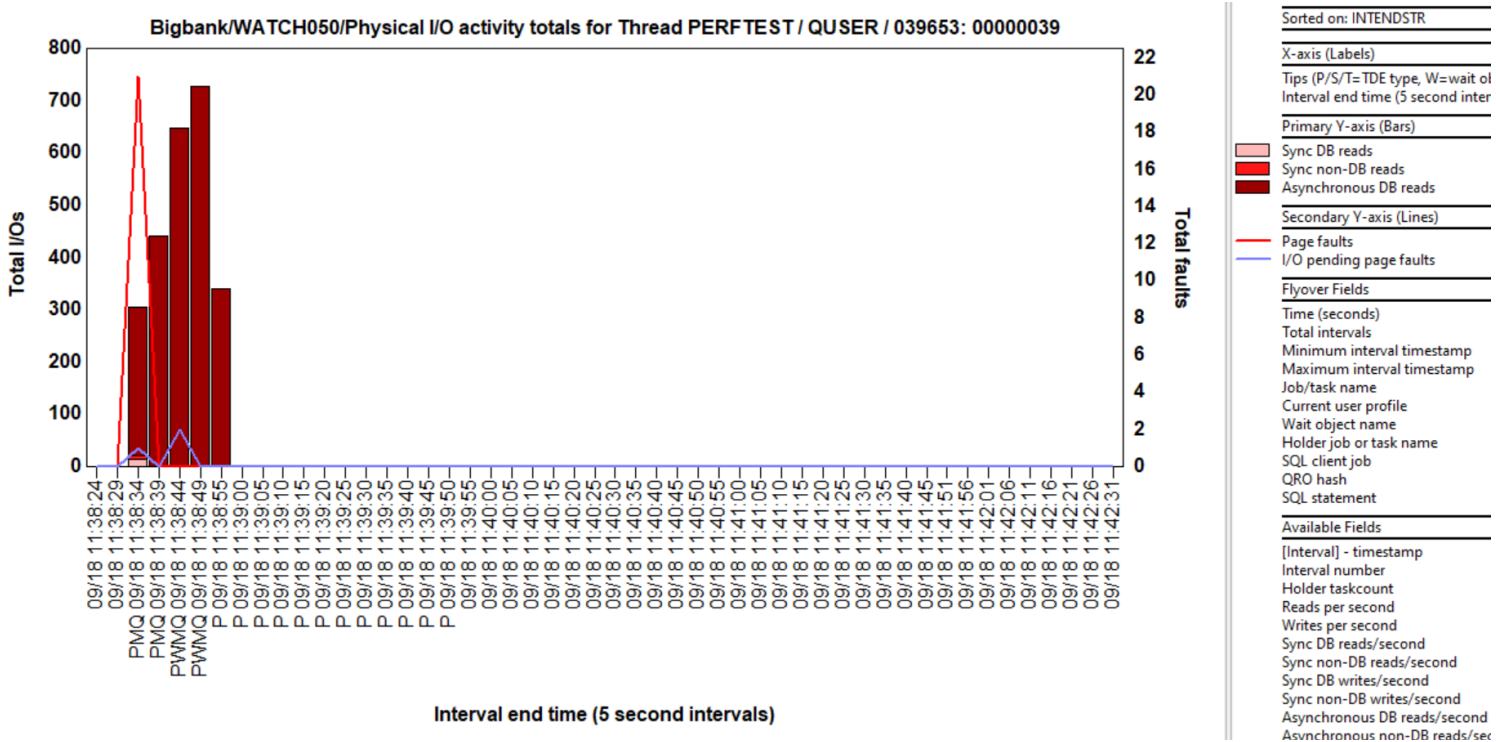
Example of running SQL – Permanent index

The time signature for the job confirms a better scenatio



Example of running SQL – Permanent index

I/O’s shows mainly asynchronous reads

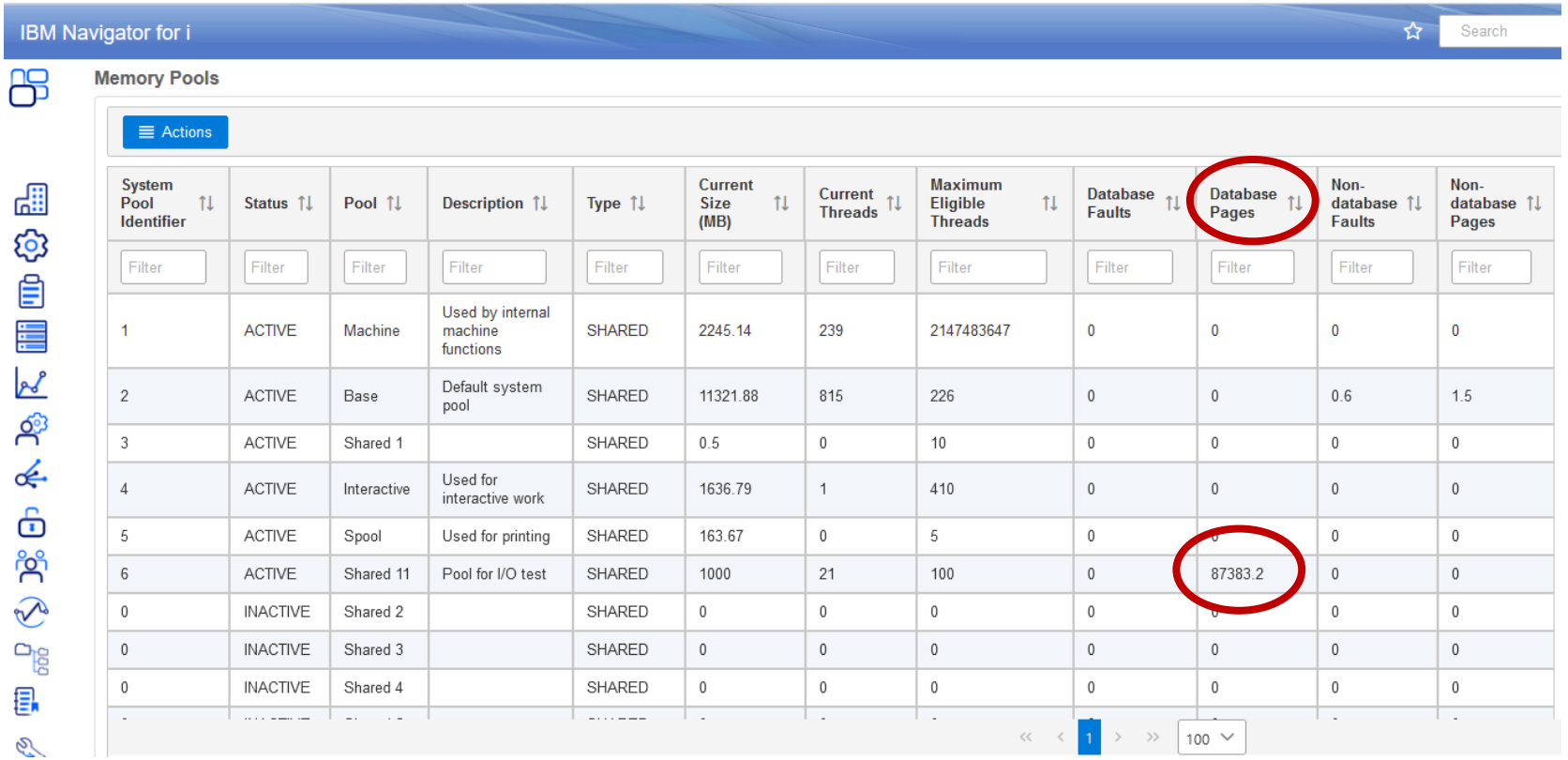


Example of running SQL – What other tools

WRKSYSSTS would show screen like the following with doing asynchronous DB reads

Example of running SQL – What other tools → New Navigator

COOL!



System Pool Identifier	Status	Pool	Description	Type	Current Size (MB)	Current Threads	Maximum Eligible Threads	Database Faults	Database Pages	Non-database Faults	Non-database Pages
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	ACTIVE	Machine	Used by internal machine functions	SHARED	2245.14	239	2147483647	0	0	0	0
2	ACTIVE	Base	Default system pool	SHARED	11321.88	815	226	0	0	0.6	1.5
3	ACTIVE	Shared 1		SHARED	0.5	0	10	0	0	0	0
4	ACTIVE	Interactive	Used for interactive work	SHARED	1636.79	1	410	0	0	0	0
5	ACTIVE	Spool	Used for printing	SHARED	163.67	0	5	0	0	0	0
6	ACTIVE	Shared 11	Pool for I/O test	SHARED	1000	21	100	0	87383.2	0	0
0	INACTIVE	Shared 2		SHARED	0	0	0	0	0	0	0
0	INACTIVE	Shared 3		SHARED	0	0	0	0	0	0	0
0	INACTIVE	Shared 4		SHARED	0	0	0	0	0	0	0

Example of running SQL – SETOBJACC?

Would SETOBJACC speed up the query here?

CHGSHRPOOL POOL(*SHRPOOL60) SIZE(10000 *MB) ACTLVL(*DATA)
TEXT('Pool for data') MINPCT(xx) MAXPCT(xx)

SETOBJACC OBJ(IBMPERF008/TABLE1_IDX) OBJTYPE(*FILE)
POOL(*SHRPOOL60) MBRDATA(*BOTH)

Answer: Very likely NOT

Example of running SQL - Learning

- The DB is extremely efficient calculating and using the best plan
- Its possible to access data asynchronous just spending CPU while running some heavy queries
- Indexes can be used to reduce I/O load and CPU usage and causing queries running faster
- Important to have enough memory, also when no faulting is taking place.

Take away

Take away

- CPU utilization is very relative
- You can do lots of I/O's without waiting
- Memory monitoring and sizing is important
- iDoctor and ACS are excellent tools and even better when used together

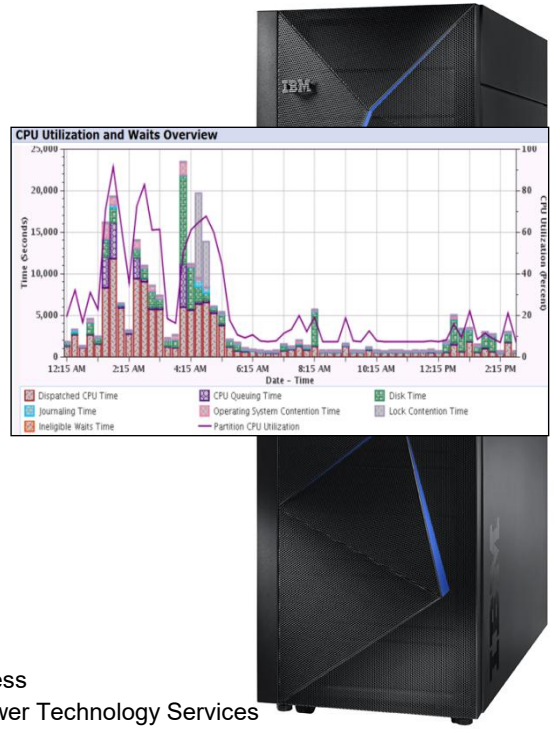
IBM i Performance: References

- IBM i Performance – IBM Documentation Site
- IBM i on Power Performance FAQ
- IBM Db2 Mirror for i: Performance Considerations
- IBM i on Power Systems Workshops
- VIOS Performance Advisor
- IBM Power Virtualization Best Practices Guide
- IBM Power Systems Performance Guide: Implementing and Optimizing
- IBM PowerVM Best Practices Redbook
- IBM PowerVM Virtualization Managing and Monitoring

IBM i Performance Analysis Workshop

Learn the science and art of performance analysis, methodology and problem solving

Managing and analyzing the data can be quite complex. During this workshop, the IBM Systems Lab Services IBM i team will share useful techniques for analyzing performance data on key IBM i resources and will cover strategies for solving performance problems. It will aid in building a future foundation of performance methodology you can apply in your environment.



Overview:

- Topics covered include:
  - Key performance analysis concepts
  - Performance tools
  - Performance data collectors (Collection Services, Job Watcher, Disk Watcher, and Performance Explorer)
  - Wait accounting
- Core methodology and analysis of:
  - Locks
  - Memory
  - I/O subsystem
  - CPU
- Concept reinforcement through case studies and lab exercises
- Discussions on theory, problem solving, prevention and best practices

Workshop details:

- Intermediate IBM i skill level
- 3-4 day workshop, public or private (on-site)
  - For public workshop availability and enrollment: IBM i Performance Analysis Workshop
  - For additional information, including private workshops, please contact Eric Barsness at ericbar@us.ibm.com or Stacy Benfield at stacylb@us.ibm.com, members of Power Technology Services