Upgrading to ASE 16 SP02

Customer

Jeff Tallman jeff.tallman@sap.com
SAP ASE Product Management

Agenda

- **Threaded Kernel**
- **Procedure Cache**
- **Query Optimization**
- **Other Key Affected Areas**
  - ULC/PLC Queue
  - Parallel Query/Utilities
- **Sizing and OS tuning**
  - T-Shirt Sizing

Disclaimer

·This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.

Migrating to ASE 16 - The big challenges

- **Process Kernel Threaded Kernel**
  - A lot of misunderstanding….and a lot of bad gouge on the internet (should be no surprise)
  - Some have had (very) legitimate issues - some OS - some ASE
  - Others didn't, but thought they did as they didn't know how to isolate the real problem causes

- **Procedure Cache sizing**
  - Changes/enhancements in QP have caused procedure cache requirements to creep up
  - This really started in 15.0, but then 15.7 was another bit of jump
  - Bad sizing often lead to spinlock contention and *excessive* use of dbcc proc_cache(free_unused)
    - ✓Most of the time when engineering suggested this to TS for customers, it was only intended to be a short term workaround until the proc cache sizing or other issue could be isolated - never intended to run for months….

- **QP Changes**
  - Believe it or not the default still behaves as if 15.0.3 ESD #1 ….even in 16 SP02
  - However, as problems get fixed, sometimes QP changes happen
    - ✓We really do try to make sure those changes are in different opt_levels….but sometimes you just can't

- **…and then there is the oft overlook list of things you shouldn't do anymore**

Migration Level of Effort

·ASE 15.7 sp100+ ASE 16

- Minimal as ASE 15.7 sp100 was code-branch
  - ✓More like an IR than a major upgrade (e.g. 15.0.2 9 15.0.3 or 15.7 ESD #4 to 15.7 sp110)
- Biggest hurdle is learning to run ASE the way it should have been under 15.7
  - ✓Threaded kernel, procedure cache/statement cache sizing, etc.

·ASE 15.7 sp 5x/6x ASE 16

- Minor - same hurdle with threaded kernel, procedure cache sizes, etc.
- Similar to LOE needed to move from 15.0.3 15.5

·ASE 15.0 or 15.5 ASE 16

- Considerable - changes in 15.7 were huge, while skipping over it, you still need to consider impact of those changes
  - ✓Note that you may have to upgrade to 15.7 *first* - and then to 16.0…and it may be advisable even if not required

·Biggest impacts

- Changes to operational procedures due to new and improved maintenance commands
- Changes in engine/memory allocations due to threaded kernel and HW memory size increases

Migration Guidance

- **You still should do the same things as any other major upgrade**

1. Get at least 1 weeks worth of baseline metrics from MDA
2. See #1
3. Search and find potential problem areas (see things you should deprecate at the end)
4. Delete and rerun update index statistics
5. Test your application
6. Test your application
7. Test your application

- **Thankfully, Workload Analyzer exists….**
  - Capture 15.7 sp137+
  - Replay on ASE 16SP02+

Migrating Page Sizes

This is *not* an upgrade issue

- You can move from ASE 15.7 to 16.0 with 2K pages
- While a 4K page may be more optimal for IO block xfers for some hardware…..
  - ✓ ….it may not be the most optimal for your application
  - ✓ Most IO bottlenecks are other easier resolved issues
    - E.g. filesystem tuning, fixing bad queries, etc.

There is less pressure now due to 2x DB size

- Prior to 15.7 ESD#2
  - ✓ pageid was a signed int
  - ✓ Result was limited to 2B pages per database
- After 15.7 ESD#2
  - ✓ logical pageid is a unsigned int
  - ✓ New limit per database is 4B pages

Methods:

- Sybmigrate 3 UI deprecated due to Sybase Central??
  - ✓ Utility/cmd line still present in 16sp02
  - ✓ Used CIS underneath with CIS ArrayInserts to move data
  - ✓ Had issues with timestamps (fixed in 15.7 due to updateable timestamps??)
- Manual bcp
  - ✓ Can be more optimal than sybmigrate - handles all datatypes better
  - ✓ May take several days - especially to rebuild indexes
- SRS Direct Load (the winner in my opinion)

Page Size	<15.7	>15.7
2K	4TB	8TB
4K	8TB	16TB
8K	16TB	32TB
16K	32TB	64TB

Is this really a page size issue or is it more of a data volume issue???

(Hint - it could be both….but most of the time, it is the latter)

Migrating Page Sizes: There are no silver bullets

- **Remember, index leaf pages have page id + row id pointers….**
  - ….and these are a much bigger issue than the normal data/index page or text chain page linkage pointers.

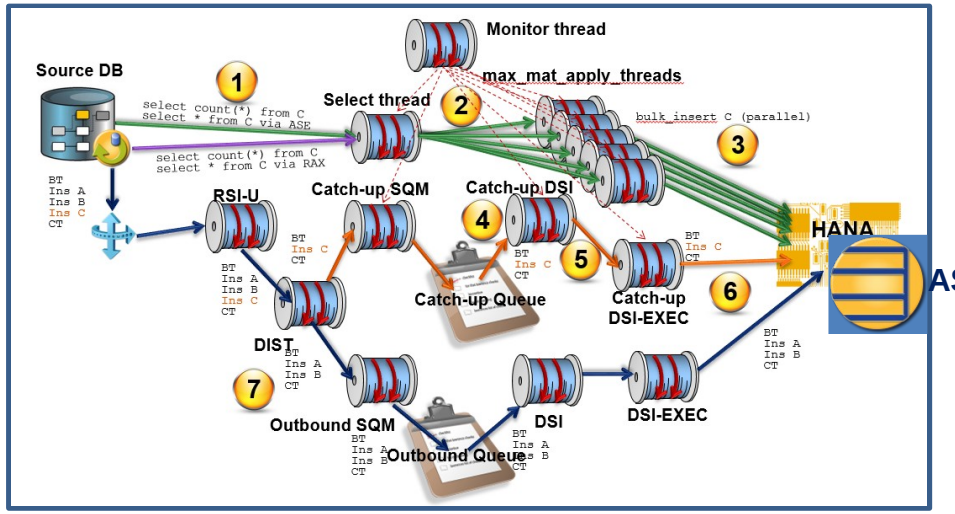
- **The commonly requested ‘solutions’ don’t work as a result**
  - Increase logical page id to unsigned bigint
    - ✓Requires expanding the page header of every page
    - ✓If page header expands into page, it may cause rows to be moved (forwarded?) - see above
  - Dump at one size/load at another (larger only….smaller is impossible?)
    - ✓Dump/load is a complete copy of the disk image
    - ✓You would need nX the page size to restore in a larger page size
    - ✓You then would have really sparse pages (but you would have to touch every page to update page space bits… and we don’t do that on a load as it goes straight to disk vs. through ASE).
    - ✓You would then have to run a reorg rebuild on EVERY table
    - ✓Then run shrink db to get space back….if you were lucky that space was freed on entire device segments.
    - ✓It is a lot of down time (dump, load, reorg, …)
  - Best option is a multi-page size server w/ alter table page size
    - ✓Requested as FR/but no real priority
    - ✓Biggest issue is impact on cache/buffer sizes - need to support multiple cache sizes per buffer size
    - ✓Would still mean doing individual table reorgs (online reorg perhaps)
    - ✓This is best solution, but it (like many other enhancements) are waiting for resource assignments

Migrating Page Sizes: A technique that works using SRS 15.7.1

- **Use SRS 15.7.1sp204+ with direct load materialization**
  - Zero-down time
    - ✓ SRS bulk loads current data and uses a catchup queue to hold inflight txns - applies when bulkload finished
    - ✓ Can use parallel selects/apply threads as well as multiple tables concurrently
    - ✓ ~100M rows/hr xfer rate
    - ✓ With no application down-time, it can be done online
  - Direct load supported
    - ✓ ASE ASE
    - ✓ ASE, DB2, MSSQL, ORA HANA

```
create subscription sub_name
for {table_repdef | func_repdef | publication pub |
database replication definition db_repdef }
[ with primary at server_name.db ]
with replicate at data_server.database
[where {column_name | @param_name}
{< | > | >= | <= | = | &} value
[and {column_name | @param_name}
{< | > | >= | <= | = | &} value]...]
[without holdlock [direct_load [init replicate table with
{create | create_or_truncate | truncate | recreate}]
[user username password pass][num_of_selects selects]
[hold_resource_on_error]]| incrementally | without
materialization]
[subscribe to truncate table]
[for new articles]
```

- **Notes/Considerations**
  - Just needs to be done over a period of days/weeks vs. a weekend
    - ✓ It can do the largest but I recommend using bcp on static portion of large tables and then create indexes before using direct load
    - ✓ For smaller databases, you can do the entire database with a database subscription with direct load
    - ✓ Needs a bit of planning due to RI constraints, etc. but could be scripted
  - Used by a large healthcare company to move a 6TB system from 2K to 8K



A quick guide to implementation

- **Copy Schema**
  - All tables, indexes, procs, triggers, etc.
  - Consider changing APL tables to datarows if using parallel materialization

- **Enable bulkcopy/truncate log on checkpoint in target DB**

- **Tune SRS 15.7.1 sp204+ (preferably 30#+) for Direct Load (lots of threads/memory)**

- **Create a job queue of tables**
  - Submit only N subscriptions at a time - using parallel shell scripts or whatever you wish
    - ✓ Although this can be controlled via num_concurrent_subs in SRS, better if you do it due to logic (below) and load balancing
  - As each one completes, check rowcounts, submit next subscription and run update index stats on prev table
  - If it fails, truncate target, drop subscription and retry (or use DA to sync rows)

- **For really big tables (>1B rows?)**
  - Drop all indexes at target
  - BCP static portion of table
  - Recreate all indexes
  - Modify rs_select fstring on repdef to select non-static portion
  - Use direct load subscription as above

```
create table migrate_job (
   tablename      varchar(255),
  num_rows       bigint,
   SPID           int    null,
   status         varchar(15),
   cur_task       varchar(15),
   task_started   datetime,
   sub_complete   datetime,
   upd_stats      datetime
) lock datarows
```

Thread Kernel

·Why it is critical and how to tune it

CPU Threading

We all know why…..

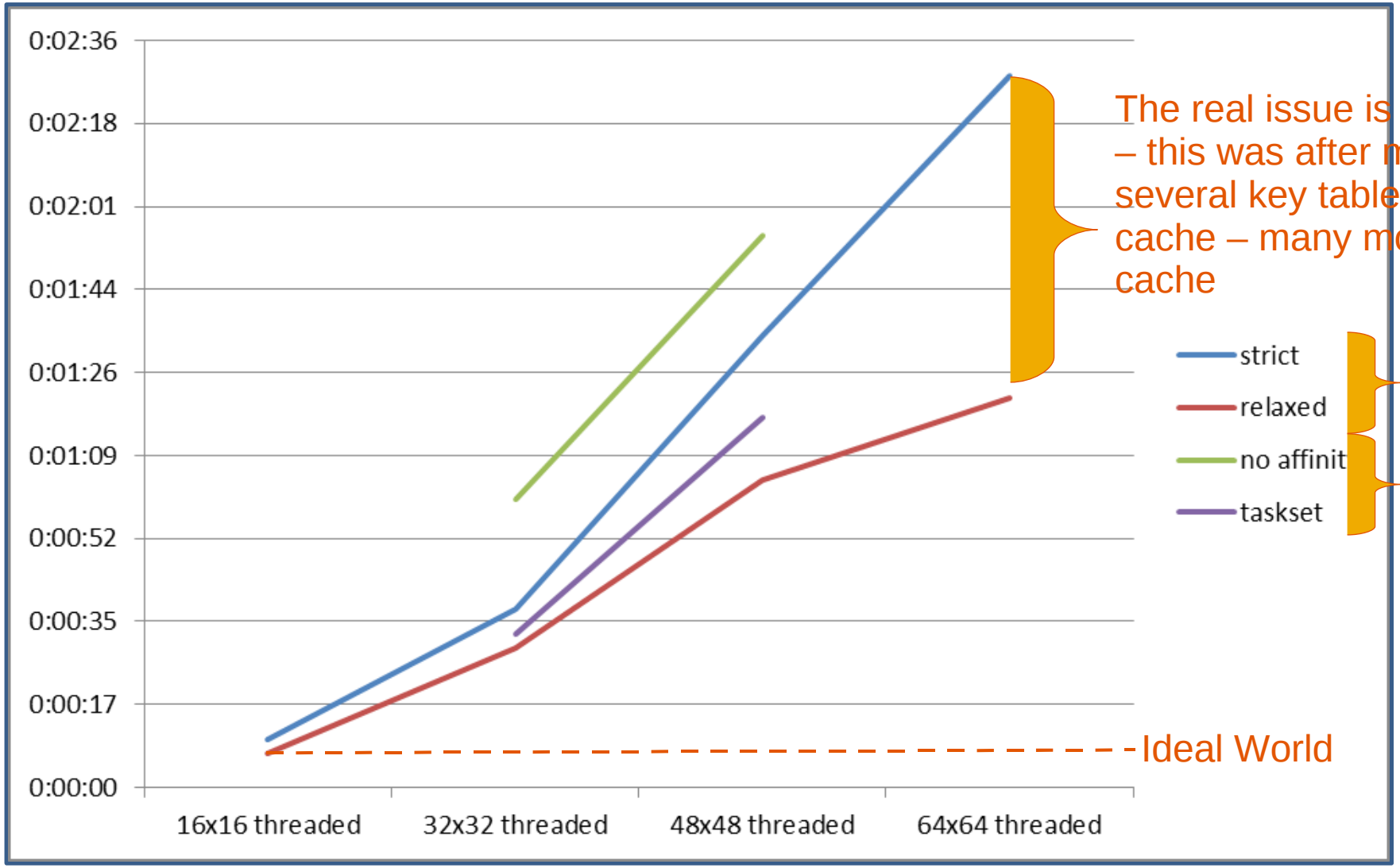
- A lot of empty CPU cycles waiting for main memory fetches, etc.
  - ✓Hence the ever larger increases in L1/L2 caches
  - ✓Exploits empty cycles by running other tasks
- Exposed to OS as if another virtual CPU
  - ✓Doesn't scale 100% - expect small gains – but a lot depends on app
  - ✓Intel x86/64 – limit engines to cores + 50%; AIX SMT-4 – limit engines to logical processors in LPAR

The real gain is in OS scheduling + CPU dispatch

- CPU's (HW chips) try to dispatch HW threads on *same* core at same time
  - ✓Not a good idea for apps with a lot of threads
  - ✓Works best for different apps/unrelated processes
- OS tries to spread a single process's OS (POSIX) threads *across* the cores
- As a result, ASE QP engines in threaded kernel will run better (across the cores)
  - ✓To do equivalent distribution in process kernel, you would have to use CPU binding (e.g. taskset in Linux)
  - ✓You may need to tune disk/network tasks in threaded kernel plus size blocking pool

Common Argument: Process vs. Threaded Kernel (ASE 15.7)

Query response times - ideal is flat line, but lower is better



The real issue is cache contention – this was after moving one of several key tables to a relaxed cache – many more still in default cache

Threaded Kernel

Process Kernel
(w/ cache change)

Transitioning to the Threaded Kernel

Now you know why

- Better distribution of workload across the cores vs. with process kernel

Along the way, we decided to fix some problems

- Network handling (eliminate need for network engines)
  - ✓More specifically, eliminating the sleep requirement to allow it to be woken up on network engine
- IO handling (eliminate need for each engine to poll for IO completion)
- Fix RepAgent latency issue
  - ✓Process kernel uses deferred async event queue for processing RPC, CIS and RepAgent communications
  - ✓When engines are busy, it deprioritized processing RPC/CIS results or RepAgent packet acks from SRS.
    - ….to the point it would “stall” for multiple minutes
  - ✓Threaded kernel RepAgent throughput is up to 10x faster than process kernel as a result

Don’t think you are “safe” running process kernel

- Much like many performance features don’t work with ‘compatibility mode’….
- …many new options don’t work with process kernel – especially 16sp02 MemScale& Always-On
- …it *may* not be there in some future release

You may need to do more OS tuning than you used to

·Process Kernel

- We distributed resources across the engines as each engine as a separate process
- Avoided the need to do OS tuning at a 'process level' for ASE
  - ✓But needed for products such as SRS & IQ

·Threaded Kernel

- More susceptible to OS process limits as there is a single process
- Things to tune
  - ✓Kernel
  - ✓/etc/security/limits.conf (or similar) & ulimit
  - ✓I/O subsystem & file system
  - ✓hardware interfaces (ifconfig)
- As this is OS specific, beyond today's session

ulimit

User limits - limit the use of system-wide resources.

Syntax

ulimit [**-acdfHlmnpsStuv**] [**limit**]

Options

-S Change and report the soft limit associated with a resource.
-H Change and report the hard limit associated with a resource.

-a All current limits are reported.
-c The maximum size of core files created.
-d The maximum size of a process's data segment.
-f The maximum size of files created by the shell(default option)
-l The maximum size that can be locked into memory.
-m The maximum resident set size.
-n The maximum number of open file descriptors.
-p The pipe buffer size.
-s The maximum stack size.
-t The maximum amount of cpu time in seconds.
-u The maximum number of processes available to a single user.
-v The maximum amount of virtual memory available to the proces

How many engines do you need

- **A good starting point is 15.5 process engines - 2**
  - E.g. if running 12 engines in 15.5, start with 10 in 15.7
  - Why?? Due to network and disk IO controllers
  - You may need even less
    - ✓Estimates were that engines were spending 20% of their time doing IO
    - ✓So you might need between 2 and (# of engines/5) fewer engines

- **What happens if you don't**
  - Effect would be similar to running 14+ (or 20% more) engines in process kernel
  - Impact is that you could increase any existing spinlock contention

- **Engine Runnable Process Search Count vs. Thread Idle Timeout**
  - Default 15.x rpsc was 2000 – that was 2000 loops through the scheduler
    - ✓if idle and no outstanding disk or network IO, engine yielded CPU
  - Thread pool idle timeout defaults to 100µs. Yes…microseconds.
    - ✓Might be a bit short for some apps – a better default might be 250, start there instead for default thread pool
    - ✓Values above 500 are likely really suspect (yes, it can improve app response time - but it mostly just churns CPU)

syb_default_pool (and any user defined thread pools)

Query Processing Engines

- Total number of engines in all pools is limited by sp_configure ‘max online engines’
- As a result, it is best to set ‘max online engines’ to the max the host will support given the core count and any cpu threading within the license constraints
  - ✓E.g. on x86 with HT enabled set to #cores + 50%
  - ✓E.g. on IBM P7 with SMT=4 3 set to cores * SMT
- Control actual number of online engines via the thread pool size
  - ✓So….it is perfectly permissible (if not advisable) to have ‘max online engines’ set to 16 on a 16 core box (HT disabled) with syb_default_pool set to 8 because 8 engines is all we need.

User defined thread pools

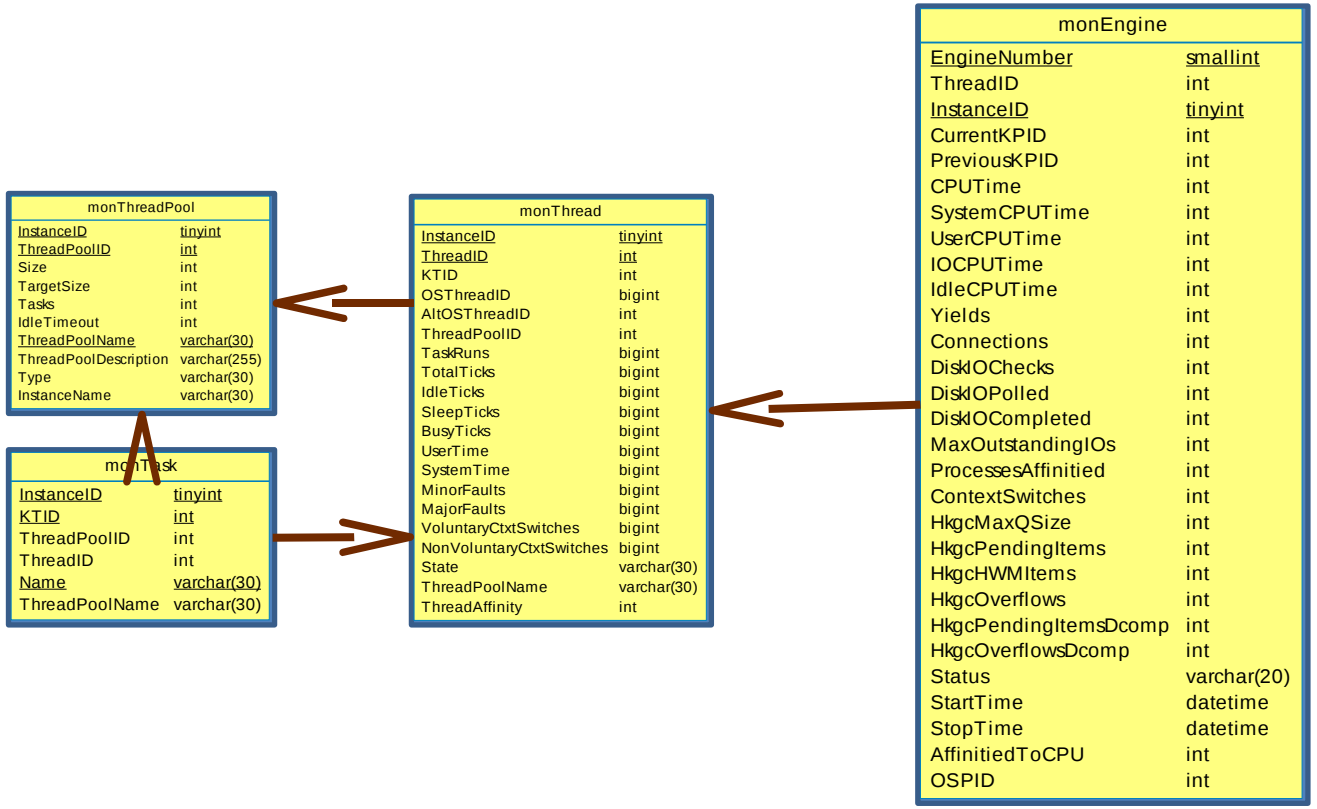
- Recommended - e.g. create one for batch jobs, DBA’s, etc.
  - ✓Set the idle timeout to a low value (e.g. 50) for less critical tasks such as DBA’s or batch jobs
  - ✓That way when no one is on, the threads don’t hog CPU
- These still are QP engines

syb_default_pool & engines

- **monEngine**
  - A list of all threads that are query processing engines
    - ✓Syb_default_pool
    - ✓User defined thread pools

- **Some legacy counters for process kernel**
  - DiskIOChecks/Polled
  - ContextSwitches
  - Connections

- **Some counters are derived**
  - CPU, IO, System, User Time
  - But they are still key - e.g. high IO time refers to time spent submitting IOs to the OS - which is a key factor in finding out if IO subsystem is slow



syb_blocking_pool

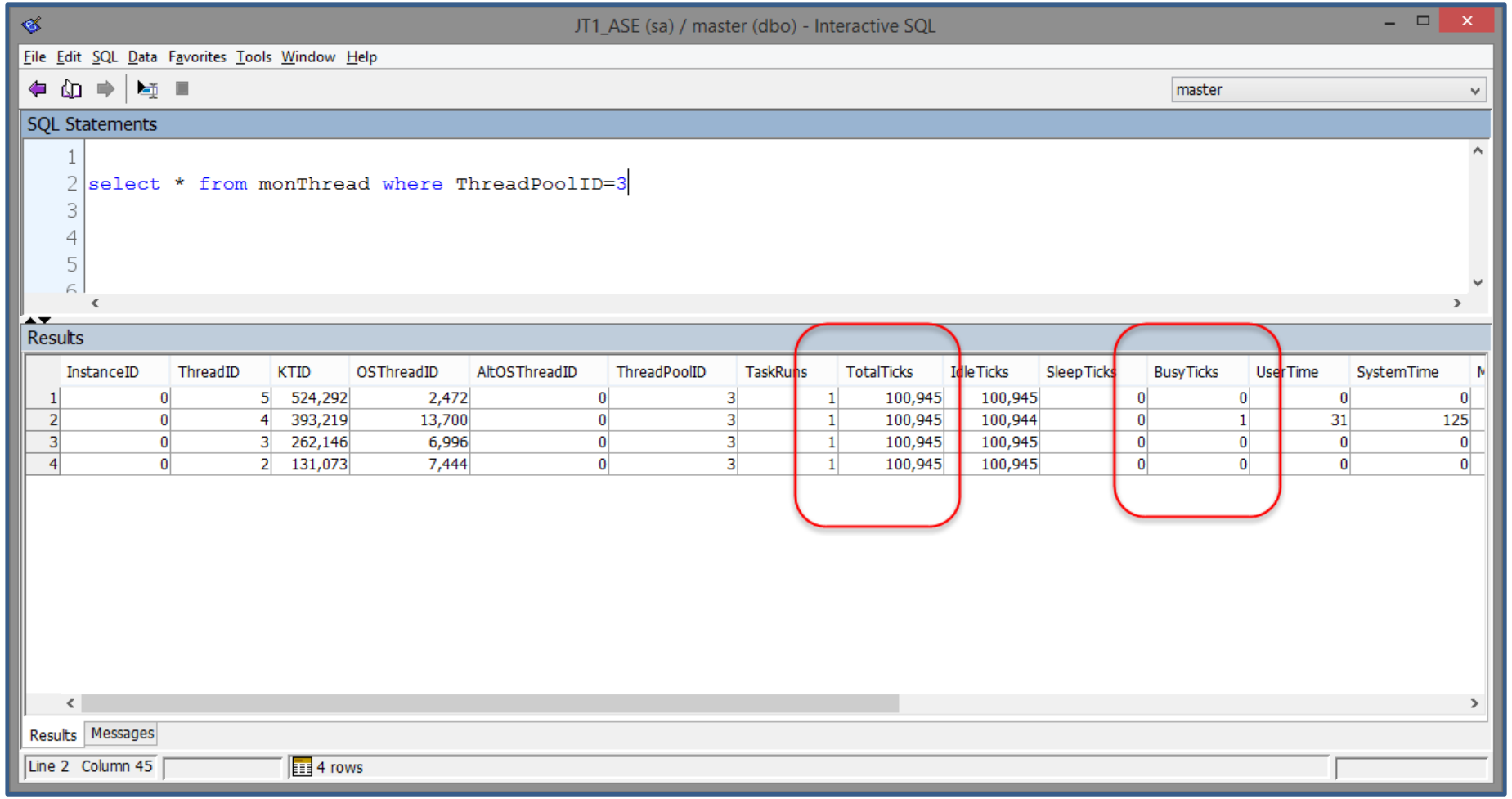
- **Used by non-disk/network tasks**
  - Essentially it will be mostly used by SPID's that run on QP engines

- **It has NOTHING to do with query execution blocking (ala locks)**
  - Not matter which genius on the internet says it does.
  - It refers to when ASE needs to execute a blocking OS call

- **Think of it this way….**
  - SPID execs a query that needs to make a blocking OS call that could take awhile
    - ✓ E.g. the first RPC or CIS call that needs to do a DNS lookup or similar…or a non-thread safe lib call that must be blocking
    - ✓ Memory allocations also can take a long time - e.g a DBA increasing total memory
  - If it stayed on the engine thread, nothing productive would be done
  - Instead, the task is put on one of the threads from syb_blocking_pool until the blocking OS call completes
  - Engine could work on something else in meantime

- **Sizing**
  - Default of 4 is more than likely enough
  - You can measure by looking an monThread and comparing BusyTicks to TotalTicks
    - ✓ If all threads show >60%, then maybe increase - but also find out why you are making so many blocking OS calls - e.g. JAVA in ASE perhaps???
  -

monThread and syb_blocking_pool

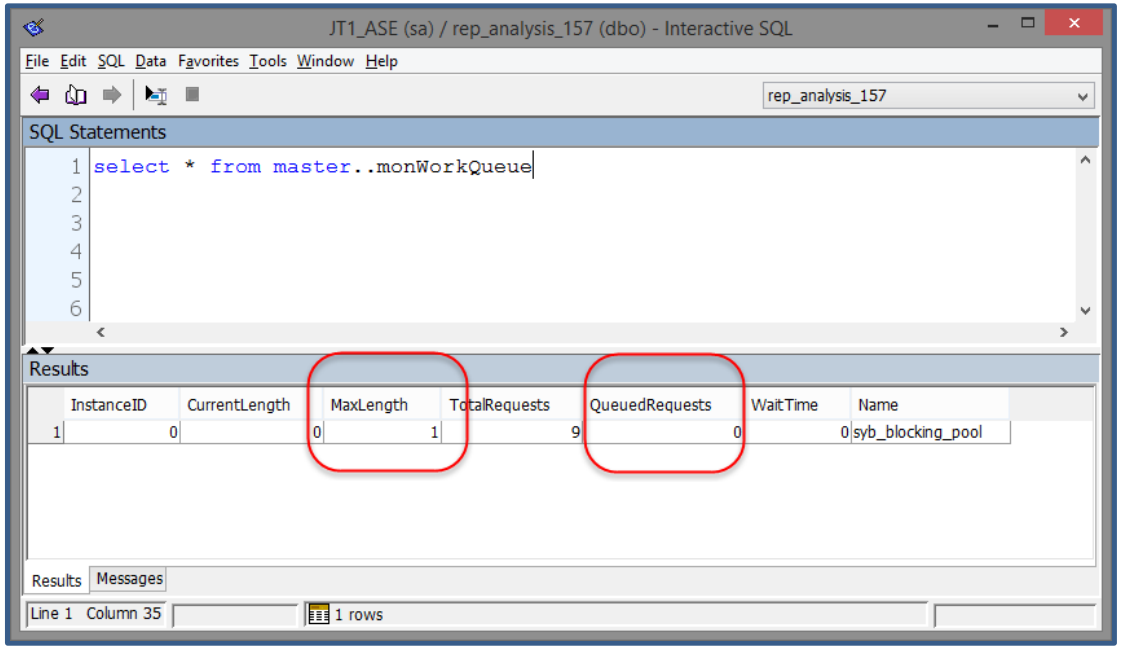


Syb_blocking_pool & monWorkQueue

- **Alternative to measuring CPU**

- **Watch monWorkQueue**
  - MaxLength
  - QueuedRequests
    - ✓Consider increasing when QueuedRequests greater than 3x the number of threads in syb_blocking_pool
  - WaitTime
    - ✓WaitTime / QueuedRequests = msPerWait

monWorkQueue	
InstanceID	tinyint
CurrentLength	int
MaxLength	int
TotalRequests	int
QueuedRequests	int
WaitTime	int
Name	varchar(30)

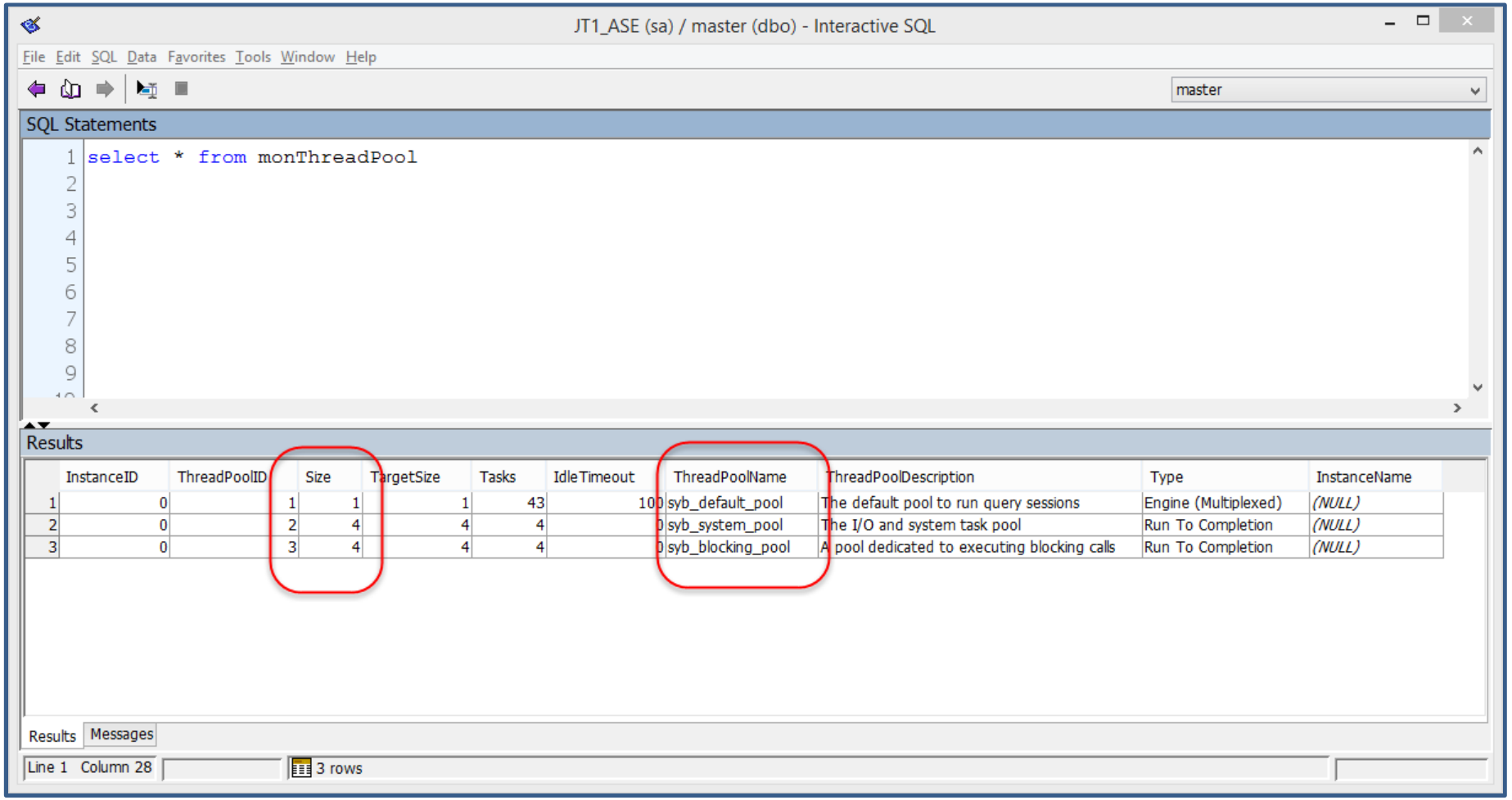


syb_system_pool

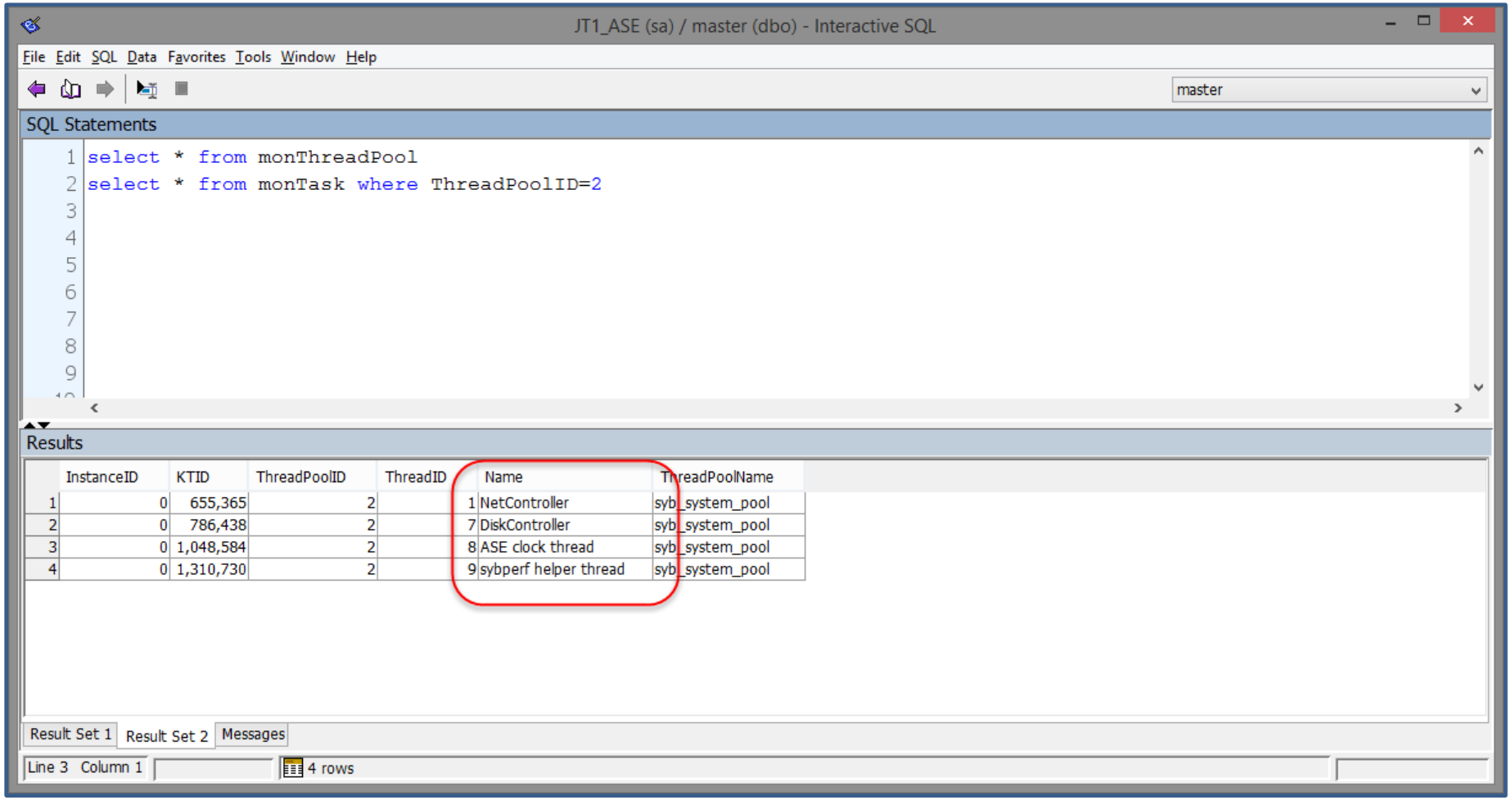
- **Not in config file**
  - Because size is determined dynamically based on configuration

- **Minimum size is 3….but most sites likely have 4+**
  - OS dependent & feature dependent
    - ✓E.g. enabling NV Cache adds one per each lazy cleaner
  - Windows
    - ✓DiskController
    - ✓NetController
    - ✓ASE clock thread
    - ✓SQL Perfmon integration helper
  - Unix/Linux
    - ✓One for each diskcontroller
    - ✓One for each netcontroller
    - ✓One for each Signal Handler
    - ✓One for CtLib Controller

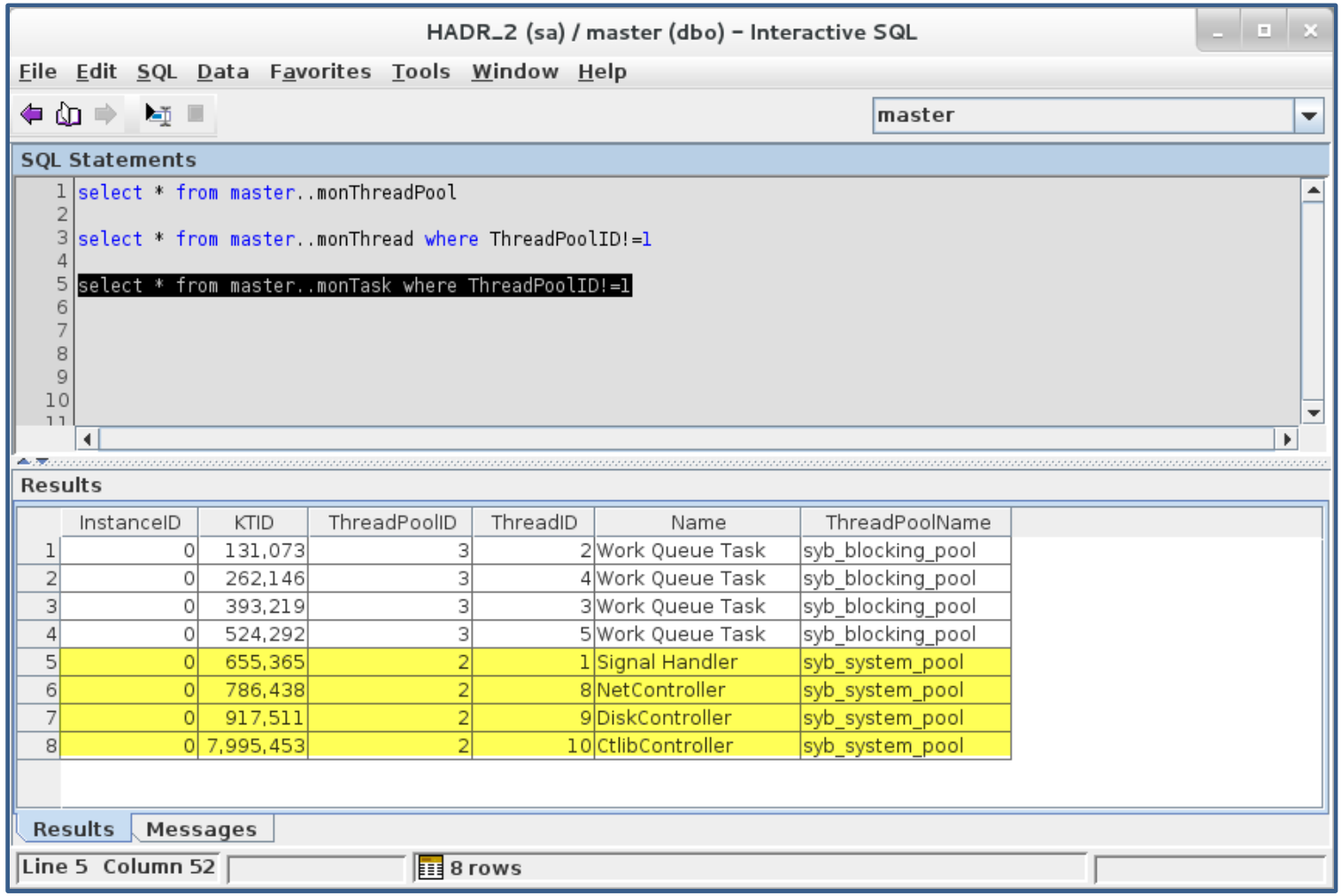
monThreadPool (Windows x64)



monTask for syb_system_pool (Windows x64)



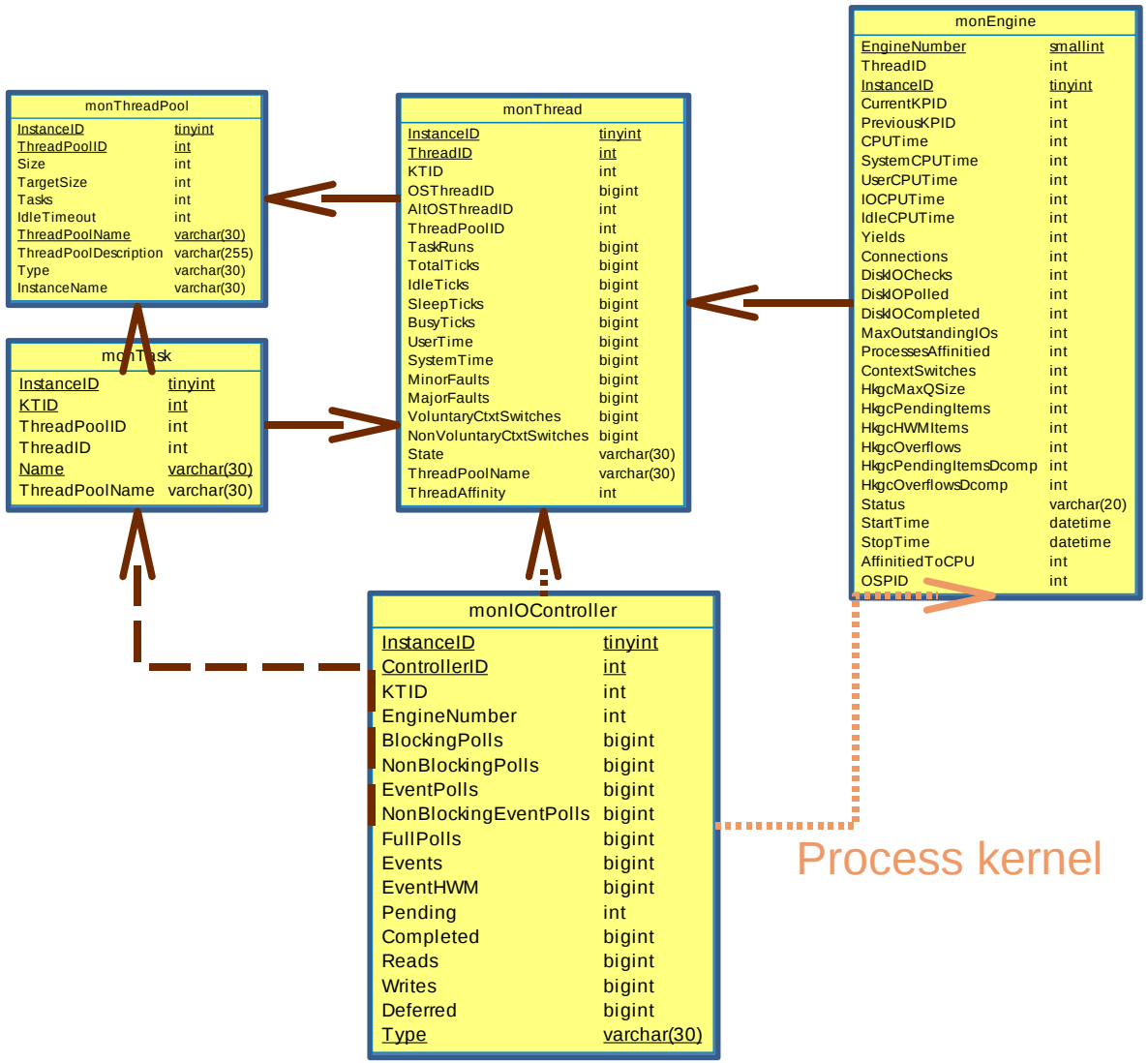
monTask for syb_system_pool (Linux x64)



	InstanceID	KTID	ThreadPoolID	ThreadID	Name	ThreadPoolName
1	0	131,073	3	2	Work Queue Task	syb_blocking_pool
2	0	262,146	3	4	Work Queue Task	syb_blocking_pool
3	0	393,219	3	3	Work Queue Task	syb_blocking_pool
4	0	524,292	3	5	Work Queue Task	syb_blocking_pool
5	0	655,365	2	1	Signal Handler	syb_system_pool
6	0	786,438	2	8	NetController	syb_system_pool
7	0	917,511	2	9	DiskController	syb_system_pool
8	0	7,995,453	2	10	CtlibController	syb_system_pool

Engines & IO Processing

- **Engines *still* do the IO**
  - If the query needs to do a disk read - the engine still submits the read request (via aio_read)
- **Engines don't check for IO completion**
  - That amount of polling overhead is now done by dedicated threads
- **IOControllers**
  - Poll (Check) for IO completion
  - If IO is complete, it wakes up SPID and puts it on the run queue
  - monIOController is a list that tracks IO processing statistics
    - ✓The thread that actually does it is in monThread (syb_system_pool)



IO Controllers (1)

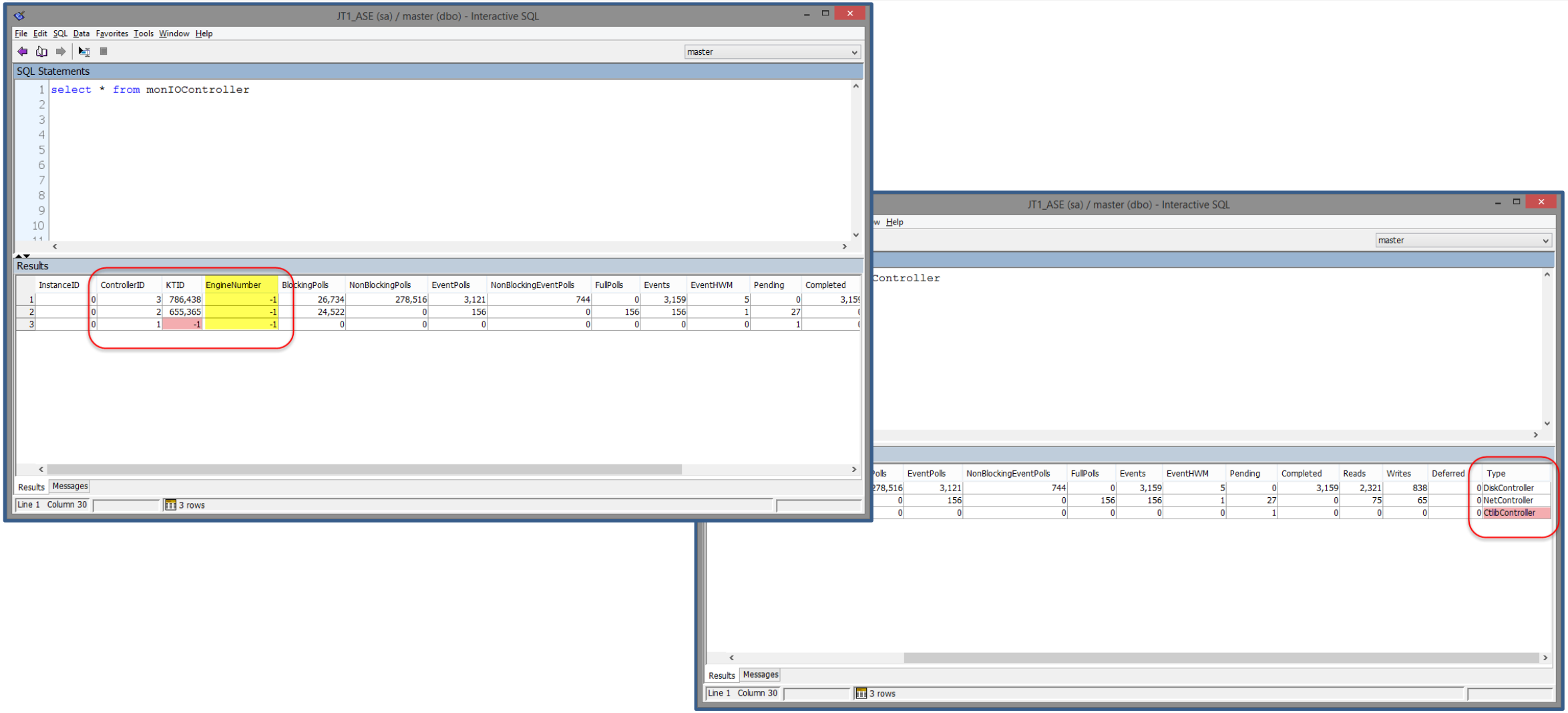
- **3 IO Controller Types**

- **Network**
  - sp_configure ‘number of network tasks’ default is 1

- **Disk IO**
  - sp_configure ‘number of disk tasks’ default is 1

- **Ctlib**
  - single controller (sorry) - but then it uses network….soooo
  - Remember, monIOController is merely the IO statistics - e.g. the IO event queue processing
  - So, the Ctlib IOController is essentially a replacement for the process kernel’s problematic deferred async event queue which was used by CIS, RepAgent, etc.
  - As a result of being more of an event queue, it doesn’t have a separate task/thread as do disk/network controllers
    - ✓The actual network processing will be done by the network IO Controller

monIOController



Tuning Disk Controllers

Disk Controllers

- A common misconception is that these do the IO's. They don't.
- Each engine still submits IO request to OS
  - ✓Remember disk IO is DMA – so request is read/write from disk to this memory address
  - ✓It puts SPID on sleep queue with link to IO request
- Disk IO controller merely *POLLS* for IO completion
  - ✓If completed, it also moves SPID from sleep to run queue (runnable)

Tuning

- Probably don't need more than one as only needs to poll for IO completion
  - ✓Still – watch the CPU busy% - if >60% or a lot of FullPolls, increase by 1
  - ✓OS IO tuning is going to be the biggest gain – the faster the OS handles the IO's, the shorter the number of outstanding IOs that disk controller has to poll for
  - ✓A slow IO subsystem could as a consequence drive disk IO CPU usage higher – if high 'Pending' look at IO subsystem tuning.
- Strictly use CPU ticks for this task to add disk tasks - don't look at monEngine IOBusy
  - ✓monEngine IOBusy measures how much CPU the ENGINE thread spent submitting IO reads & writes
  - ✓For disk controller, we are looking at how much CPU is spent POLLING for IO completion
  - ✓A slow IO subsystem could increase CPU for both….

Network IO Controller Tuning

Network IO Controller Tuning

- Most OS's these days support Receive Side Scaling (including Solaris 11 finally)
  - ✓This means that more than one socket can process network interrupts
- Network IO Controller is merely polling for IO completion - therefore only when there is a lot of network traffic is it likely that due to RSS the single IO Controller will start to lag

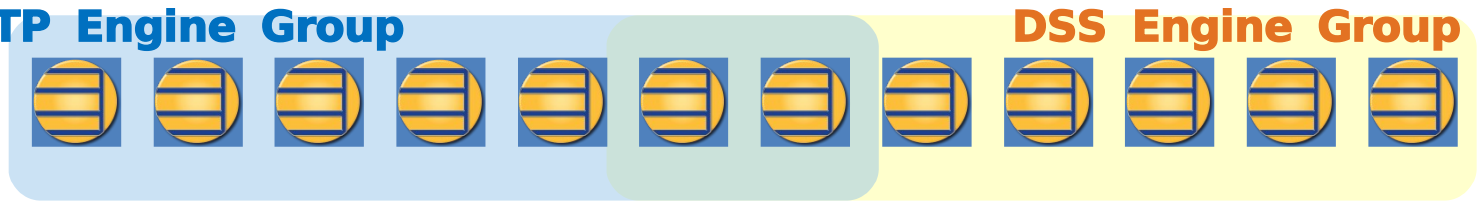
How to tune ASE network controllers

- Start with 1 and increase as necessary
  - ✓Most likely you will max out at about 1 per socket (good starting config as sweet spot)
- Docs say monitor monTread for CPU usage of network controller thread
  - ✓If busy >60%, consider adding one
- Use monIOController - if regularly see FullPolls (non-Windows) then increase
  - ✓Windows will always show FullPolls due to TCP stack is single event
- If using thread pools, you may want to monitor monProcessWaits for WaitEventID=251 (send sleep essentially) for apps in non-default pool and add network controllers if more than ~5ms/IO
  - ✓This is my personal favorite as it has shown benefits even when monThread wasn't that busy.

Engine Groups (Process Kernel)

- **Creating Engine Groups**
  - All engines are allocated to a single global engine pool
  - Create engine group and assign engines from existing max online engines
  - Use dynamic listeners to force users to have network IO on same engines as are in engine groups

- **Issues**
  - Only a single set of run queues regardless of number of engine groups
  - By default, an application could run on ANYENGINE…app binding required
  - Engine groups could overlap



Run Queues
(by base EC class)

SPID	EC	Engine Group
101	1	OLTP
25	1	DSS
36	1	OLTP
42	1	DSS
58	1	ANYENGINE

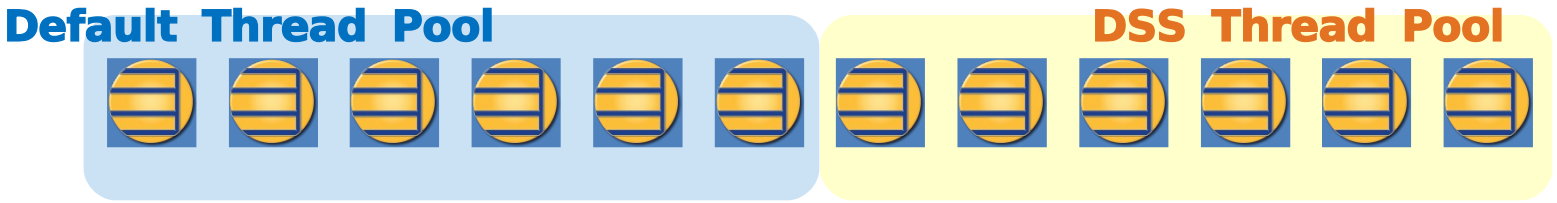
SPID	EC	Engine Group
13	2	OLTP
202	2	DSS
35	2	ANYENGINE
48	2	DSS
56	2	OLTP

SPID	EC	Engine Group
145	3	OLTP
274	3	ANYENGINE
352	3	OLTP
480	3	DSS
57	3	ANYENGINE

Thread Pools (Threaded Kernel)

- **Creating Thread Pools instead of Engine Groups**
  - Initially, all engines are in the default pool (after upgrade)
    - ✓ Use alter thread pool to reduce threads to desired number of engines (engine group size)
  - Create new/additional thread pool for non-default engine groups
  - Each thread pool has it's own run queues

- **Issues**
  - No single spanning group such as ANYENGINE (e.g. for dba's)
  - No ability to use LASTONLINE as isolation engine (e.g. RepAgents)
  - No overlap allowed for ala engine groups



SPID	EC
101	1
25	1
36	1
42	1
58	1

SPID	EC
13	2
202	2
35	2
48	2
56	2

SPID	EC
145	3
274	3
352	3
480	3
57	3

SPID	EC
105	1
27	1
37	1
43	1
59	1

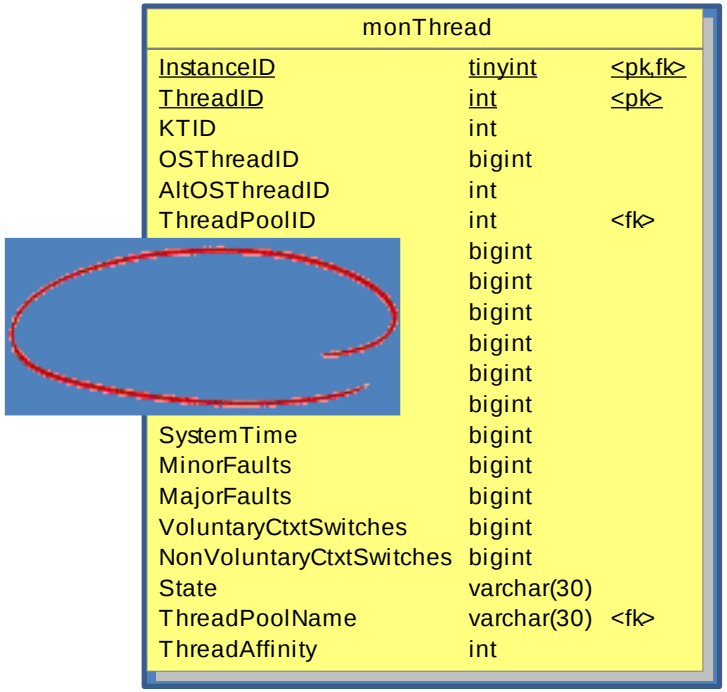
SPID	EC
14	2
203	2
36	2
49	2
58	2

SPID	EC
146	3
275	3
354	3
481	3
51	3

Monitoring ASE 15.7+ in Threaded Kernel

- **monThread is key table**
  - Contains all threads – engines as well as IO controllers, etc.

- **Key Metrics:**
  - CPU% = BusyTicks/TotalTicks
  - Time is an approximated derived value based on sampling of a tick length
  - SleepTicks – not on CPU
  - IdleSpin% = IdleTicks/TotalTicks
    - ✓If higher than 30%, consider reducing IdleTimeout
  - Idle% = (IdleTicks + SleepTicks)/TotalTicks
    - ✓If multiple engines show >100%, reduce thread pool size
  - NonVoluntaryCtxtSwitches □CPU oversubscribed
  - MajorFaults □ Memory oversubscribed



Procedure Cache

·Spinlocks, changes, etc.

Procedure/Statement Cache – How it Really is Sized

- **Two parameters that control it via sp_configure**
  - procedure cache size □ size of procedure cache less statement cache extension
  - statement cache size **k** in reality does 2 things
    - ✓Increase the total procedure cache to (procedure cache size + statement cache size)
    - ✓Limits the memory used by statement cache to (statement cache size)

- **Each statement consists of:**
  - A SQL statement in statement cache
  - One or more query plans in proc cache (as LWP objects)

- **Other configs & tips:**
  - enable literal autoparam
    - ✓Enable if OLTP and a lot of static SQL (vs. fully prepared statements)
    - ✓Disable if DSS or predominantly fully prepared statements
  - streamlined dynamic SQL
    - ✓Enable if predominantly fully prepared statements
    - ✓Enable if using a lot of app servers connection pools with rapid grab/release from connection pool and using fully prepared statements
  - Optimize temp table resolution
    - ✓May help reduce excessive proc cache thrashing/recompilation for #temps and subprocs

Proc & Statement Cache

Proc cache

- Minimum needed will likely be 4GB for most enterprise class systems (e.g. ~1TB)
- You will likely need 8GB or more
  - ✓Especially if doing a lot of bulkloads
- Maximum you will *likely* need is 12-16GB
  - ✓Anything higher than this is due to misconfigured ‘number of sort buffers’
  - ✓…..or a ton of engines/concurrent queries

Statement cache

- Each 1GB will hold ~15,000 statements
  - ✓Rule of thumb sizing – number of user connections * estimated statements (minimally 10)
- Plan for 1-2GB
- For each 1GB of stmt cache, you will need 512MB more proc cache if using streamlined dynamic SQL.

Some proc cache sizing formulas

On average….

- Each stored proc plan uses about 500K-1MB+ of proc cache once optimized
- On average, due to concurrency, most procs use a total of 5-10MB of proc cache
- A typical query is 150-300KB of proc cache for the final plan
- Due to all the possible candidate work plans, the optimizer will use 2-5MB of proc cache per query

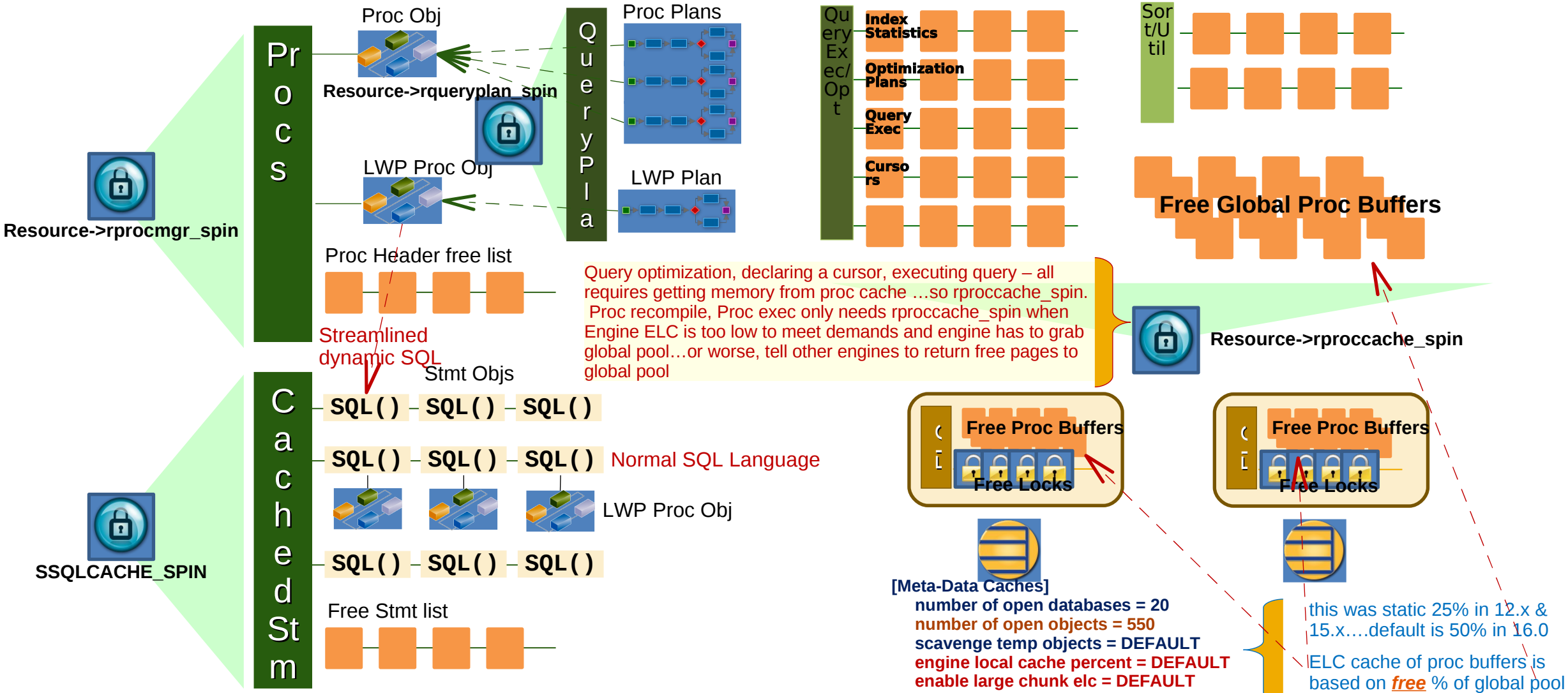
Sooooo…..

- If we have 10000 connections, and we assume 1000 are active with 100 executing procs
  - ✓Assume that 33.3% of them are optimizing at any time
- We need:
  - ✓900*0.333 * 5MB = 1.5GB of proc cache for qp optimization (not all at same time, but possible peak)
  - ✓900*0.666 * .15MB = ~100MB of proc cache for query execution
  - ✓100*0.333 * 10MB = ~350MB of proc cache for proc reoptimization
  - ✓We need ~1.5GB to cache 250 procs
- We minimally need 3.5GB of proc cache - but then with ELC, more likely 7GB+

After sizing - monitor

- Keep proc cache and statement cache removals to <<10% of procs/statements cached
- Use monProcedureCacheModuleUsage to track optimization, execution and other uses

Proc Manager vs. Proc Cache Spinlocks



How it gets used

·Any query optimization

- Needs to grab proc cache for work plans, loading histograms, etc.
- Retains some proc cache for final optimization plan
- Grabs some proc cache for execution plan

·Fully prepared statement

- Needs proc cache manager to create LWP
- Needs proc cache
  - ✓First to optimize….then to hold final opt plan
- Adds query opt plan to query plan manager
- If streamlined dynamic SQL enabled, adds SQL text and links to LWP to statement cache

·Stored procedure

- Similar to fully prepared statement
- However, due to multiple statements, query plan will be much bigger
- In addition, due to concurrency, multiple plans may be in query plan manager same time
- Other problem is that if proc creates a #temp and calls subproc, subproc is recompiled
  - ✓Unless ‘optimize temp table resolution’ enabled
  - ✓Can be really nasty if subproc is called in a loop

·Auditing

- Auditing ‘all’ on tables/logins and then executing stored procs for audited tables/logins
- Audit trail during proc exec is cache in proc cache….Gigs and gigs of proc cache

Essentially, all roads lead to rproccache_spin

If you see high contention higher in the chain….

- Contention ≠ Grabs. Contention = spins. Spins are the *most* important
- E.g. SSQLCache, Proc Manager, etc.
- You will also likely see high turn over in proc cache/stmt cache (removals/inserts)
- ASE 16 might help as use count instructions may be using CAS/atomic CPU instructions
- Likely causes are
  - ✓Proc cache not big enough causing object turnover
  - ✓App behavior is driving high turnover

…but eventually we will hit proc cache as it is our "memory pool"…so sizing is critical

- Even if stmt/rpc found and query opt plan also found….we need to create query exec plan
- The more common issue is that due to concurrency, we are raiding proc cache a lot to create plans
- ….so it becomes a common complaint
- The trick is if the other earlier spinlock *spins* are also high - it is the cache volatility driving the proc cache spin lock - fix the cache volatility first as it will fix the proc cache spinlock problem
- If not (earlier spinlocks are fine) then reality is proc cache sizing
  - ✓Too big can be just as big of an issue as too small.

Query Optimization

·Changes in query optimization in ASE 16 from ASE 15.7

Query Optimization in 15.7 and 16.0

- **A lot of query optimization was done in 15.7**
  - Much of this was to support Business Suite as well a custom application QP issues
  - To avoid impacting applications a scheme of optimizer levels was implemented

- **ASE 16sp01**
  - Extensive changes to parallel query optimization
  - Support for large numbers of partitions w/ fast optimization
  - Improved support for parallel index creation (with consumers)
    - ✓Remember if using the consumers=X clause, you need 2X+1 worker threads
      - X worker threads for consumers
      - X worker threads for producer
      - 1 parent thread

- **A word on compatibility mode….**
  - It was intended as a way to have ASE 15.x emulate ASE 12.5 as much as possible
  - …statement always was that ‘there are no plans (yet)’ to deprecate ‘compatibility mode’
  - …doesn’t mean there won’t be plans in the future

Some defaults that shouldn't have been (15.0+)

- **Defaults were likely okay for small systems**
- **But on large systems, they cause problems**
  - ….especially for OLTP
- **The biggest culprits:**
  - Adhoc queries - max resource granularity
  - Queries in stored procs - sproc timeout
    - ✓Includes queries in execute() clauses inside procs

Configuration	Default	Recommend
max resource granularity	10	3
optimization timeout limit	10	5
sproc optimize timeout limit	40	10
enable merge join	2	0

Partitioned tables & parallel query

- **A lot of changes in parallel query optimization**
  - Bloom filters (useful especially with hash joins)
  - Parallel sort operators
  - Parallel hash join operators (hash build/hash probe operators)
  - Dynamic Thread Assignment
  - Fact table hints
  - …..but only if using opt_level ase_current
    - ✓…or at least a 16sp01+ opt_level

- **Parallel utilities now controlled via different configurations**

Query compilation time seconds/partitions on the batch of 16 various queries

On high partitioned tables (> 1000 partitions) we use in average 16-18% of the initial time.

Query Optimizer Levels

Optimizer levels added in ASE 15.7

- Each level represents changes that are reflected in a specific version of SAP ASE.
  - ✓Optimizer levels are named for the version of SAP ASE in which they were included with this format:
  - ✓ase+<version_number>+<esd_number>
  - ✓For example, ase1503esd2 includes the optimizer functionality added to SAP ASE version 15.0.3 ESD #2
- ase_default is ASE 15.0.3 ESD #1 compatible (this is the default….ugh!)
- ase_current is ASE 16.x capable (e.g. sp01 as well as sp02 optimization fixes)

Optimizer criteria changes – specific optimizer functionality

- Two types – CR/planned feature
- CR optimization criteria - usually added with a change request (CR).
  - ✓Optimizer criteria names use the formats: cr+<change_request_number>
  - ✓For example, cr497066
- Planned feature – usually a descriptive title
  - ✓For example, imdb_costing, which includes PIO costing for scans for in-memory databases.
-

Exploiting Optimizer Levels

- **What this means is you can upgrade to ASE 16 and not have any optimization concerns**
  - The default is still as if ASE 15.0.3 ESD #1

- **If you hit issues, you can create your own custom optimization level:**
  - SAP does this for business suite

```
set plan optlevel ase_current
set plan optgoal allrows_mix
set merge_join 0
go
exec sp_optgoal 'sap_oltp', 'save'
go
sp_configure 'goal',1, 'sap_oltp'
go
```

  - To create your own
    - ✓Set the desired opt level as starting point
    - ✓Enable/disable as many opt criteria as desired
    - ✓Save it with sp_optgoal
    - ✓Set the server optgoal with sp_configure to the saved custom optlevel

This can be really useful for migrations

- **Create a ‘safety’ optgoal**
  - If upgrading from ASE 15.7 sp135

```
set plan optlevel ase157sp135
set plan optgoal allrows_mix
set merge_join 0
go
exec sp_optgoal 'ase157sp135', 'save'
go
sp_configure 'goal',1, 'ase157sp135'
go
```

  - Create a new optgoal

```
set plan optlevel ase_current
set plan optgoal allrows_mix
set merge_join 0
go
exec sp_optgoal 'ase16sp02', 'save'
go
sp_configure 'goal',1, 'ase16sp02'
go
```

- **Run with the new one - if issues, switch back to previous optgoal**
  - Select ‘exec sp_recompile ‘’’+name+’’’’ from sysobjects where type =‘U’

Other Major Changes

·PLC Queue, Parallel Utilities, etc.

Things to be aware of

·PLC Queue

- Prevents buffer unpins from causing increased log contention
- Even if not used now, if you increase page size, this could be more important to you

·Backup Server allow.hosts file

·Configuration changes you need to do (or ought strongly consider)

- ASE Stack Size
  - ✓Needs to be larger - this started in ASE 15.7 sp130 series
    - stack guard size max(default+8K,24K)
    - stack size max(default+8K,512K)
- Other configuration changes to consider
  - ✓Disable aggressive task stealing
  - ✓Enable error log rollover (errorlog size=100MB)
  - ✓Enable dump history
  - ✓Enable delta dump tran
  - ✓Enable select into in tran (allows replication of more stored procs)
  - ✓Disable tempdb object scavenging (reduce object manager contention)
  - ✓Enable bulk inserts (not related to bcp - more for SQL merge, etc.)

Your deprecation list

- **Async Log Service**
  - Use delayed commit on scratch databases
  - Use PLC Queue for buffer unpin

- **Update stats with parallel consumers**
  - Use update statistics with hashing (and consider increasing step count to 50 or 100)

- **Exec proc with recompile (create procedure with recompile)**
  - If used a lot, could be a cause of procedure cache fragmentation
  - Largely a holdover from pre-15 and deferred compilation
  - Causes entire proc plan to be recompiled (5MB) vs. a single statement (150KB)
  - Use the select…with recompile *statement* level instead if needed

- **Dbcc tune(des_bind)**
  - Often not implemented fully due to missing column defaults, procs, triggers
  - Scavenge tempdb objects B 0 (false) - reduces object manager spinlock contention due to temp tables
  - Should not be needed (or drastically reduced) due to lockless changes on keep counts & MDA stats

Your deprecation list (cont)

- **Configs:**
  - IO Polling Process Count number of disk tasks
  - Runnable Process Search Count thread pool idle timeout
  - User log cache size user log cache queue size

Sizing

- How to estimate ASE sizing and which configs to adjust

Memory….where it all goes

Some configs to consider:

- Locks 1.2GB per 5 million
- Open Objects , 2KB/obj or 115MB/50,000
- Open Indexes , 2KB/idx or 105MB/50,000
- Open Partitions 1.4KB/ptn or 71MB/50,000
- Compression Info Pool 32-64MB per 500 users
- Large IO buffers 256MB for 64 or 4MB each
- User Connections S 512MB per 1000
- Heap Memory per User 256MB
- Kernel Resource Memory 300MB

The impact….on just server “resources”

Resource	Setting	Memory
Locks	10,000,000	2048
Open Objects/Indexes	100,000	512
Compression Info Pool	2,000 users	256
Large IO Buffers	64	256
User Connections	5000	2560
Heap Memory per User	64KB@	320
Kernel Resource Memory		300
		~6GB

Named Caches in an Enterprise System

Cache	Size	Contents
Systables_cache	256-512MB	All system tables in user DB's
Tempdb_cache	2GB+	
Log_cache	1GB	Syslogs in DB (75% logio size, 25% pagesize)
Volatile_cache	256MB	Small, frequently updated tables
Reference_cache	256-512MB	Small, static tables
Restriction_cache	1-5GB	History tables, text columns, large nuisance tables, etc.

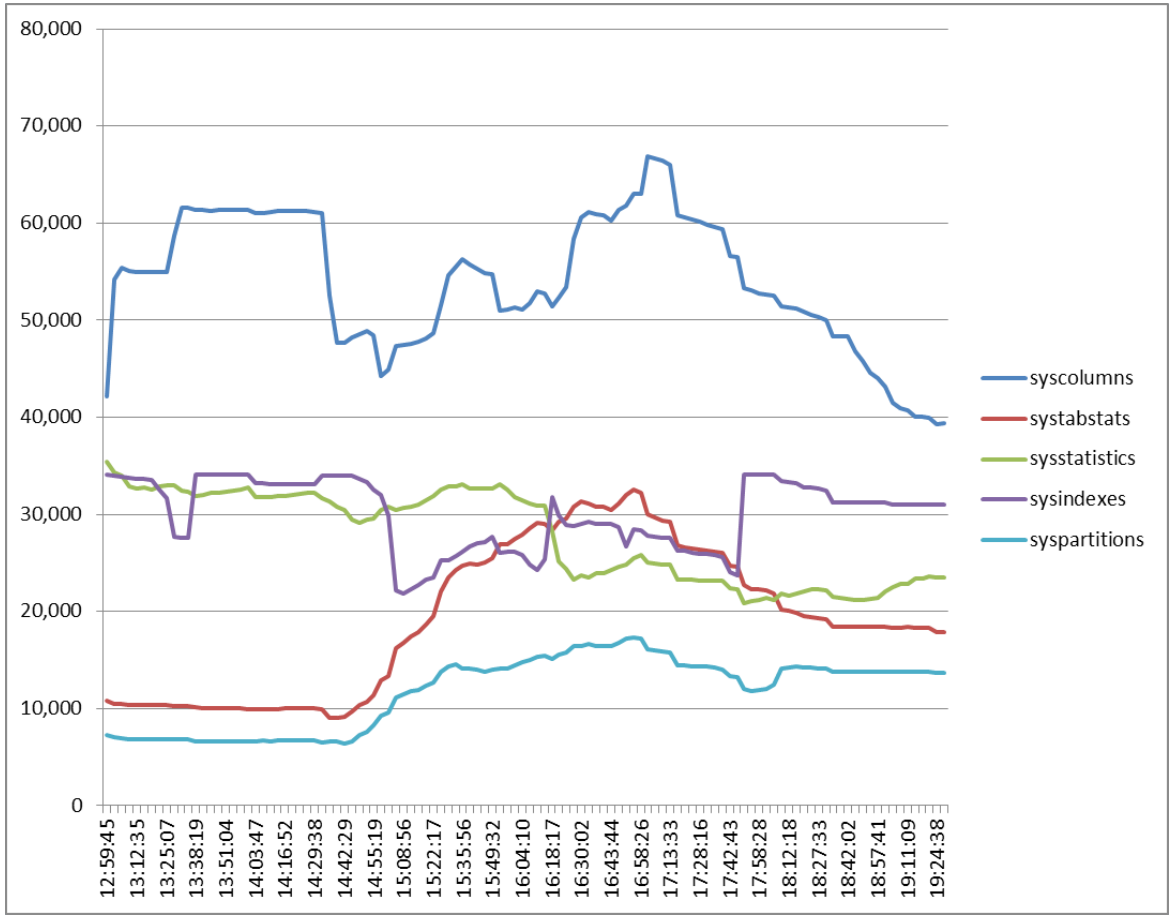
The only reasonable excuse for not having these at a minimum is due to using NVCache

Systables_cache vs. metadata cache

- **Metadata cache**
  - Mostly used by ASE to find objects in memory
    - ✓E.g. DES will have addresses of IDES
  - Also used to track object concurrency
    - ✓e.g UsedCounts
    - ✓Previously used spinlocks on DES/IDES, etc. to avoid issues
      - E.g. <16.0, we used a spinlock on systypes when creating a temp table to avoid a type being dropped that was getting used for a temp table creation
    - ✓>16GA uses CAS to avoid spinlocks on systypes & sysdatabases (for cross-db operations)

- **System tables**
  - Used by ASE to parse query SQL/resolve objects
  - Used by optimizer to:
    - ✓Get a list of all indexes on a table (DES has a limit of ~12 IDES links – plus LOB col index chains)
    - ✓Get index column histograms (stats)

CachedKB for Top 5 System Tables in Default Data Cache



Adding Up the Memory for a ~1TB system

Cache	Size
Resource configs	6GB
Proc cache	8GB
Statement cache	2GB
Systables_cache	320MB
Tempdb_cache	2GB
Log_cache	1GB
Volatile_cache	256MB
Reference_cache	256-512MB
Restriction_cache	1-5GB
TOTAL	**21-26GB**

…and we haven't even seen default data cache yet!!!!!

Sp_configure T-shirt Sizing Quick Hits (1)

config option	category	subcategory	current default	Petite (Test) 1-2	Small 3-7	Medium 8-16	Large 17-32	Extra Large 33-64	Super Scale 65+
number of alarms	application	resources	400	1000	2500	5000	10000	15000	20000
global cache partition number	cache		1	1,2	4,8	8,16	8,16,32	16,32,64	64
compression info pool size	compression		4096	64MB	128MB	512MB	1GB	1GB	1GB
enable console logging	db maint	error logging	0	1	1	1	1	1	1
histogram tuning factor	db maint	histograms	20	20	20	20	10	10	10
number of histogram steps	db maint	histograms	20	50	50	50	100	100	100
number of sort buffers	db maint	histograms	500	10000	10000	10000	10000	10000	10000
update statistics hashing	db maint	histograms	0	default	default	on	on	on	on
utility statistics hashing	db maint	histograms	0	off	off	on	on	on	on
max repartition degree	db maint	parallel utilities	1	3	12	24	48	64	64
max utility parallel degree	db maint	parallel utilities	1	3	12	24	48	64	64
enable async database init	db maint	space mgmt	0	default	default	1	1	1	1
enable housekeeper GC	db maint	system tasks	1	5	5	5	5	5	5
disk i/o structures	disk i/o	OS limits	256	1024	4000	8000	16000	32000	32000
max async i/os per engine	disk i/o	OS limits	2147483647	1000000	1000000	1000000	1000000	1000000	1000000
max async i/os per server	disk i/o	OS limits	2147483647	1000000	1000000	1000000	1000000	1000000	1000000
i/o batch size	disk i/o	performance	100	default	default	200	300	400	400
number of disk tasks	disk i/o	performance	1	1	1	1	2	4	4
number of large i/o buffers	disk i/o	performance	6	32	64	64	128	128	256
number of pre-allocated extents	disk i/o	performance	2	2	2	4	4	8	8
text prefetch size	disk i/o	performance	16	default	64	128	256	512	512
number of devices	disk i/o	resources	10	100	100	250	512	1024	1024

Sp_configure T-shirt Sizing Quick Hits (2)

config option	category	subcategory	current default	Petite (Test) 1-2	Small 3-7	Medium 8-16	Large 17-32	Extra Large 33-64	Super Scale 65+
number of dtx participants	DTM	resources	500	500	1000	2500	5000	7500	10000
aggressive task stealing	engines		1	default	default	default	0	0	0
max online engines	engines		1	2	8	16	32	64	128
number of engines at	engines		1	1-2	3-8	8-16	17-32	33-63	64+
kernel mode	kernel		threaded	threaded	threaded	threaded	threaded	threaded	threaded
lock scheme	locking	concurrency	allpages	datapages	datapages	datapages	datapages	datapages	datapages
lock wait period	locking	concurrency	214748364	1800	1800	1800	1800	1800	1800
page lock promotion HWM	locking	lock escalation	200	5000	10000	25000	50000	100000	100000
page lock promotion LWM	locking	lock escalation	200	5000	10000	25000	50000	100000	100000
page lock promotion PCT	locking	lock escalation	100	70	70	70	70	70	70
row lock promotion HWM	locking	lock escalation	200	10000	50000	100000	200000	300000	400000
row lock promotion LWM	locking	lock escalation	200	5000	25000	50000	100000	150000	200000
row lock promotion PCT	locking	lock escalation	100	70	70	70	70	70	70
lock address spinlock ratio	locking	resources	100	default	50	25	15	10	5
lock hashtable size	locking	resources	2048	8192	16384	32767	65536	65536	131072
lock spinlock ratio	locking	resources	85	default	default	40	40	20	20
lock table spinlock ratio	locking	resources	20	default	default	15	10	5	5
number of locks	locking	resources	10000	500000	2500000	5000000	10000000	15000000	20000000
kernel resource memory	memory	kernel	6396	8192	12288	16384	32768	65536	131072
enable HugePages	memory	OS integration	0	0	0	2	2	2	2
heap memory per user	memory	process	4096	16384	16384	32768	32768	65536	65536
allocate max shared	memory	resources	0	0	1	1	1	1	1
dynamic allocation on	memory	resources	1	1	1	1	1	1	1
max memory	memory	resources	120832	4-8GB+	16-32GB+	32-64GB+	64-256GB+	256GB-1TB+	256GB-

Sp_configure T-shirt Sizing Quick Hits (3)

config option	category	subcategory	current default	Petite (Test) 1-2	Small 3-7	Medium 8-16	Large 17-32	Extra Large 33-64	Super Scale 65+
number of open databases	metadata	dbtable	12	25	50	100	150	200	250
number of open objects	metadata	des	500	10000	50000	100000	150000	200000	200000
open object spinlock ratio	metadata	des	100	40	40	20	20	10	10
scavenge temp objects	metadata	des	1	1	1	0	0	0	0
number of open indexes	metadata	ides	500	5000	25000	50000	100000	100000	100000
open index hash spinlock	metadata	ides	100	40	40	20	20	10	10
open index spinlock ratio	metadata	ides	100	40	40	20	20	10	10
number of open partitions	metadata	pdes	500	5000	25000	50000	100000	100000	100000
partition spinlock ratio	metadata	pdes	10	40	40	20	20	10	10
cpu accounting flush	monitoring	resources	200	2147483647	2147483647	2147483647	2147483647	2147483647	2147483647
enable spinlock monitoring	monitoring	resources	0	1	1	1	1	1	1
i/o accounting flush	monitoring	resources	1000	2147483647	2147483647	2147483647	2147483647	2147483647	2147483647
max SQL text monitored	monitoring	SQL	0	2048	4096	4096	4096	8192	8192
SQL batch capture	monitoring	SQL	0	1	1	1	1	1	1
sysstatistics flush interval	monitoring	systabstats	0	5	5	5	5	5	5
max network packet size	network i/o	compatibility	2048	16384	16384	65024	65024	65024	65024
additional network memory	network i/o	performance	0	1048576	4194304	16777216	67108864	268435456	107374182
number of network tasks	network i/o	performance	1	1	1	2	4	4	4-8
max number network	network i/o	resources	5	5	5	10	10	20	20

Sp_configure T-shirt Sizing Quick Hits (4)

config option	category	subcategory	curent default	Petite (Test) 1-2	Small 3-7	Medium 8-16	Large 17-32	Extra Large 33-64	Super Scale 65+
max parallel degree	parallel query	limit	1	3	12	24	48	64	64
max query parallel degree	parallel query	limit	1	1	4	6	8	10	10
max scan parallel degree	parallel query	limit	1	1	4	4	4	4	4
min pages for parallel scan	parallel query	limit	200	500000	500000	500000	500000	500000	500000
prod-consumer overlap	parallel query	limit	20	20	20	20	20	20	20
memory per worker process	parallel query	resources	1024	65536	65536	65536	65536	65536	65536
number of aux scan	parallel query	resources	256	default	1000	1500	2000	2500	2500
number of messages	parallel query	resources	64	50-100	150-350	400-800	850-1600	1650-3200	3200+
number of worker processes	parallel query	resources	0	6	20	50	100	200	250
procedure cache size	proc cache	resources	14000	256-512MB	1-2GB	4-8GB	8-12GB	12-16GB	12-16GB
statement cache size	proc cache	stmt cache	0	100MB-	512MB-1GB	1-2GB	2-4GB	4-8GB	4-8GB
enable merge join	query	opt feature	2	0	0	0	0	0	0
mnc_full_index_filter	query	opt feature	2	0-2	0-2	0-2	0-2	0-2	0-2
optimization goal	query	opt feature	allrows_mi	allrows_oltp	allrows_oltp	allrows_mix	allrows_mix	allrows_mix	allrows_mix
optimizer level	query	opt feature	ase_defau	ase_current	ase_current	ase_current	(as is)	(as is)	(as is)
max buffers per lava	query	opt resources	2048	default	10240	10240	10240	10240	10240
max resource granularity	query	opt resources	10	10	5	3	1	1	1
optimization timeout limit	query	opt resources	10	10	10	5	5	5	5
sproc optimize timeout limit	query	opt resources	40	40	20	10	10	10	10
allow resource limits	query	resources	0	1	1	1	1	1	1
enable logins during	recovery	access	1	default	0	0	0	0	0
compression memory size	recovery	backups	0	1MB	2MB	4MB	16MB	16MB	16MB

Sp_configure T-shirt Sizing Quick Hits (5)

config option	category	subcategory	curent default	Petite (Test) 1-2	Small 3-7	Medium 8-16	Large 17-32	Extra Large 33-64	Super Scale 65+
housekeeper free write	recovery	checkpoint	1	10	10	5	5	5	5
number of checkpoint	recovery	checkpoint	1	1	1-2	2-4	4-8	8+	8+
recovery interval in	recovery	checkpoint	5	60	120	240	480	960	960
max concurrently	recovery	performance	0	1	2-7	7-15	16-31	32	32
recovery prefetch size	recovery	performance	0	0	1000	5000	10000	20000	20000
replication agent memory	replication		4096	4096	4096	8192	16384	32768	32768
audit queue size	security	audit	100	default	16384	32768	65535	65535	65535
suspend audit when device	security	audit	1	0	0	0	0	0	0
check password for digit	security	authentication	0	1	1	1	1	1	1
deferred name resolution	stored procs		0	1	1	1	1	1	1
optimize temp table	stored procs		0	0	0	0	1	1	1
procedure deferred	stored procs		1	1	1	1	1	1	1
permission cache entries	user	performance	64	64	100	100	250	250	250
number of user	user	resources	25	100	1000	1500	5000	10000	10000+
stack guard size	user	stack	19456	24576	24576	24576	24576	24576	24576
stack size	user	stack	493568	default+81	default+81	default+81	default+81	default+819	default+81
session tempdb log cache	user	ULC	4096	default	32768	65536	65636	131072	131072
user log cache queue size	user	ULC	1	0	0	1	1	1	1
user log cache size	user	ULC	16384	4x	4x	4x	4x	4x pagesize	4x
user log cache spinlock	user	ULC	20	20	15	10	5	5	5


Questions???
Follow-up…
Discussion…
Clarifications…

UKSUG

SAP DATABASE & TECHNOLOGY USER GROUP

THANK YOU

For more information on SAP ASE 16 visit:

www.sap.com/ase

http://help.sap.com/ase1602/

https://ideas.sap.com/SAPASE