SQLskills Immersion Event

IEPTO2: Performance Tuning and Optimization

Module 5: SQLOS Scheduling and CPU Performance Tuning

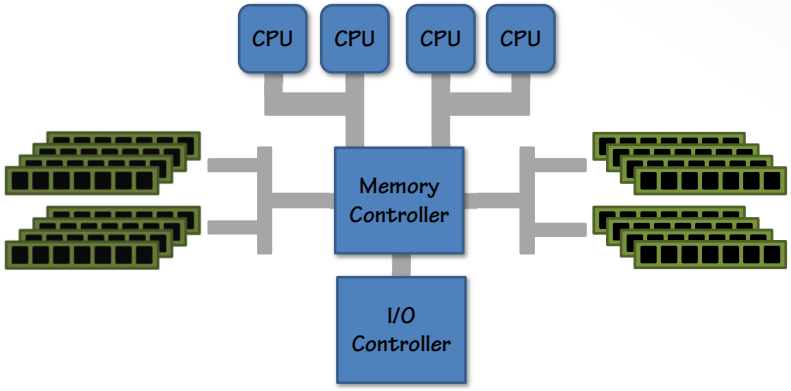
Jonathan Kehayias
Jonathan@SQLskills.com

Overview

- Understanding Windows scheduling
- Server hardware and NUMA
- CPU scheduling under SQLOS
- DMV monitoring
- Troubleshooting CPU performance issues
- Using Resource Governor to limit CPU usage

Traditional SMP Systems

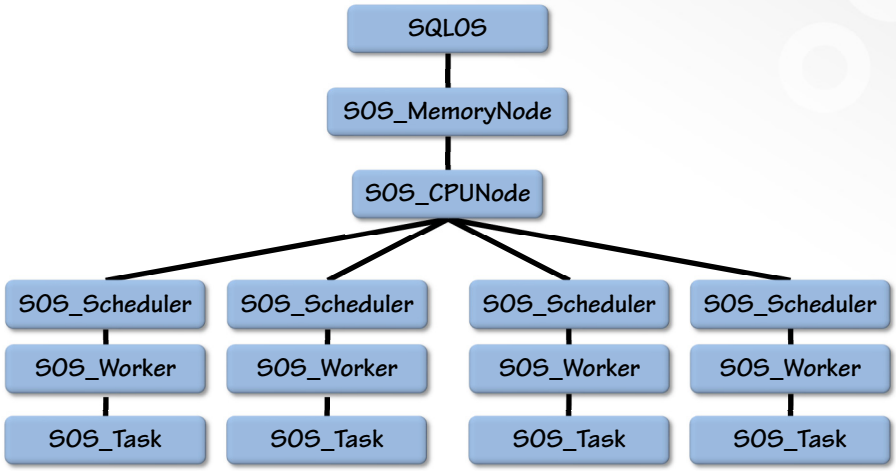
- Processors share front-side bus or cross bar
- Memory access uniform across all CPU cores



SQLskills immerse yourself in sql server

3

SQLOS Under SMP



SQLskills immerse yourself in sql server

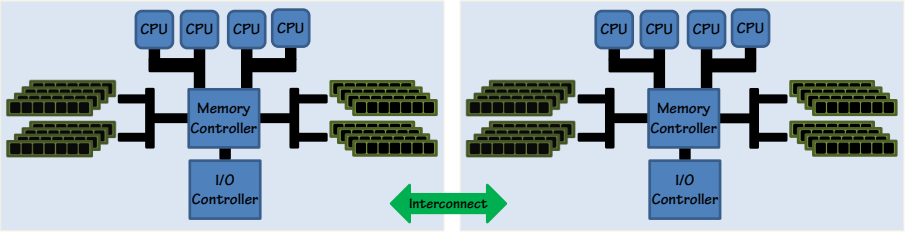
4

The Case for Non-Uniform Memory Access (NUMA)

- Modern CPUs operate faster than the memory to which they are attached
- Only one processor can access memory at a time for coherency
- As the number of processor cores increases, the cross bar/front-side bus becomes the system bottleneck starving multiple processors for memory access

Traditional NUMA

- Traditionally NUMA configurations required special hardware (e.g. IBM xSeries Servers)
- Nodes were physically segregated servers connected by special interconnect
  - Two SMP servers running together as a single system
  - Required consideration for I/O and controller placement

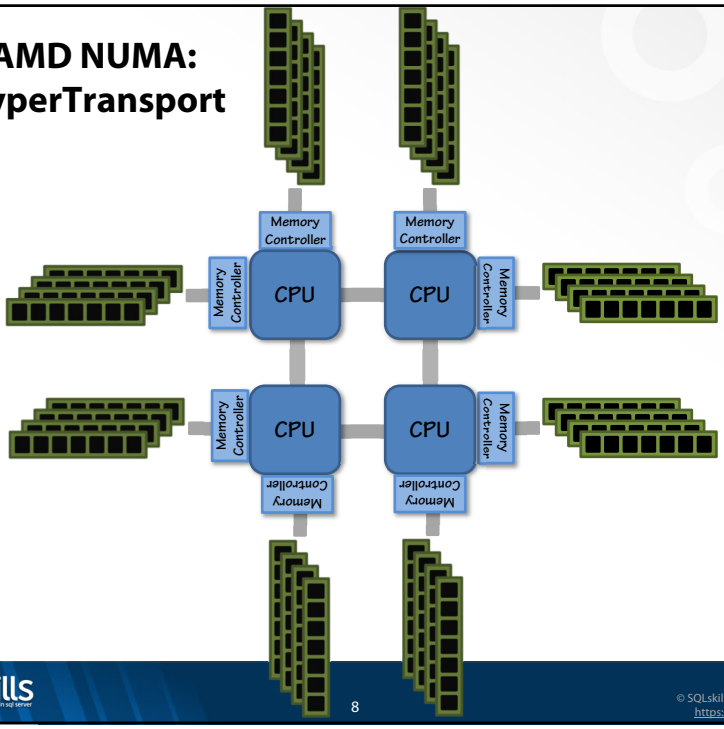


Current NUMA

- Memory controllers are built into the processor die
- Each processor socket presents itself as an individual NUMA node to the OS unless node interleaving is enable in the BIOS
- Inter-node communication paths are extremely fast and foreign-memory allocations have a lower impact on overall performance

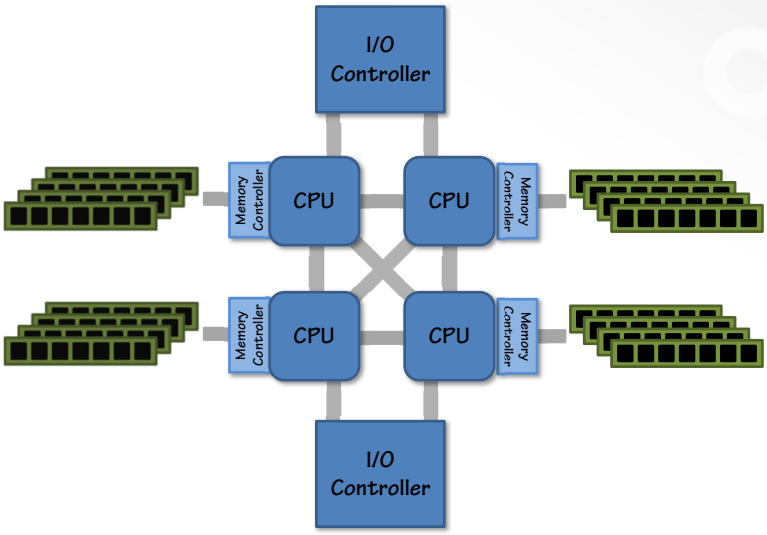
SQLskills immerse yourself in sql server

AMD NUMA: HyperTransport



SQLskills immerse yourself in sql server

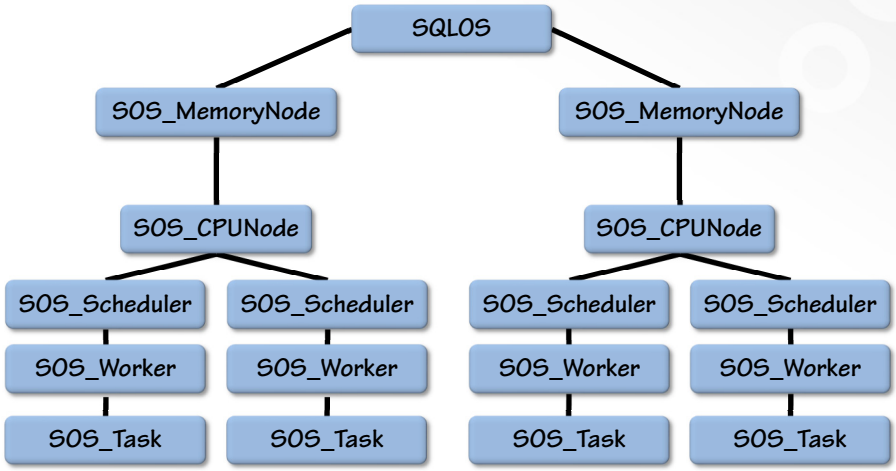
Intel Nehalem: QPI



SQLskills immerse yourself in sql server

9

SQLOS Under Hardware-NUMA



SQLskills immerse yourself in sql server

10

Is NUMA Important Still?

- **Even with enhancements, NUMA still matters**
  - Separate lazy writer process per NUMA node
  - Impact for checkpoint operations
  - Latency with remote memory still matters
- **Max Degree of Parallelism**
  - Number of physical cores per NUMA node
  - Prevent cross-node scheduling of parallel processing
    - Leverage Level-1 cache on die to prevent translation look-aside buffer (TLB) misses
      - The TLB is a CPU cache that memory management hardware uses to improve virtual address space translation speed
      - The TLB maps virtual and physical address spaces to reduce the number of page table lookups performed, which read multiple memory areas to compute the address needed

Node Interleaving

- **The option to run the server in an SMP configuration**
- **Sufficiently Uniform Memory Architecture (SUMA) configuration**
  - Memory mapped in 4KB regions to nodes in round-robin even fashion instead of as a single, sequential block under NUMA
- **Beneficial to certain workloads, but should not generally be used with SQL Server**
- **Trace Flag 8015 – disables NUMA detection by SQLOS**
  - Should not be used by most SQL Server workloads
  - Limits SQL Server to K-group 0 and 64 CPU's or less
- **Trace Flag 8048 – prior to SQL Server 2016**
  - Useful when > 8 schedulers per NUMA node or node interleaved
  - Partitions CMEM-OBJ heap space per CPU instead of per-NUMA node to reduce CMEMTHREAD contention
  - SQL Server 2016+ dynamically repartitions on contention by default

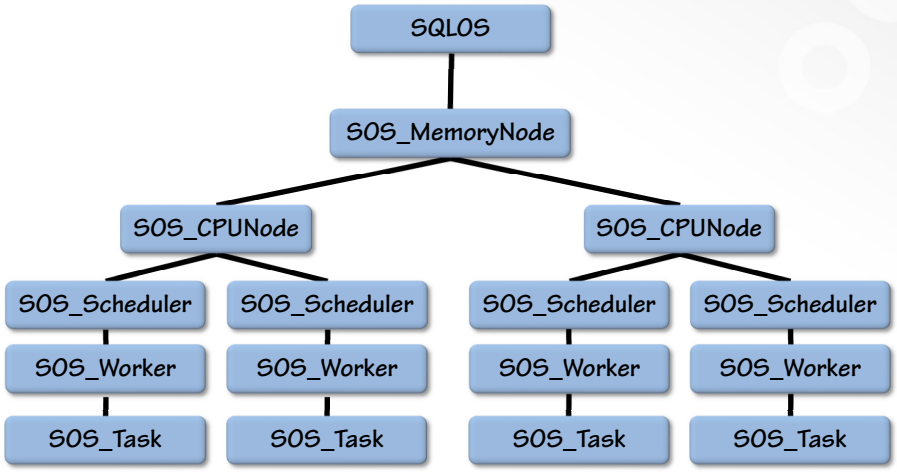
Soft-NUMA

- **Creates a custom NUMA configuration independent of hardware NUMA configuration**
  - Registry settings control soft-NUMA configuration
  - Can provide greater performance, scalability, and manageability on SMP as well as on real NUMA hardware for specific workloads
  - ETL world record set using soft-NUMA configuration
  - Allows the ability to mask independent nodes to specific TCP ports for increased performance
- **Soft-NUMA nodes can only include processors from a single hardware-NUMA node**
- **Automatically configured in SQL Server 2016 by default if > 8 cores per NUMA Node**
  - https://blogs.msdn.microsoft.com/psssql/2016/03/30/sql-2016-it-just-runs-faster-automatic-soft-numa/

Automatic Soft-NUMA

- **SQL Server 2016 will configure Soft NUMA partitioning automatically if > 8 cores per hardware NUMA Node**
  - ErrorLog: *Server Automatic soft-NUMA was enabled because SQL Server has detected hardware NUMA nodes with greater than 8 physical cores.*
- **Can be disabled using ALTER SERVER CONFIGURATION**
  - Requires SQL Agent be stopped before issuing command or SQL Agent shutting down will issue RECONFIGURE and revert the setting
  - Execute ALTER SERVER CONFIGURATION
  - Restart SQL Server
  - Start SQL Agent
- **SQL Server 2014 SP2, can enable Automatic Soft NUMA when Trace Flag 8079 is set as a startup parameter**
- **https://blogs.msdn.microsoft.com/psssql/2016/03/30/sql-2016-it-just-runs-faster-automatic-soft-numa/**

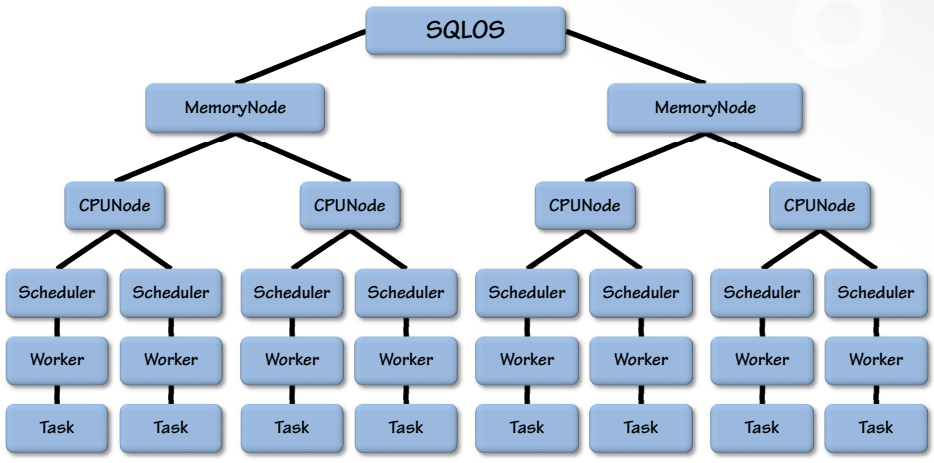
SQLOS Under SMP with Soft-NUMA



SQLskills immerse yourself in sql server

15

SQLOS Under Hardware-NUMA with Soft-NUMA



SQLskills immerse yourself in sql server

16

SOS Scheduler

- **Provides a thin layer between SQL Server and the OS**
  - Provides cooperative scheduling and I/O processing for SQL Server instead of preemptive scheduling used by Windows
    - Cooperative scheduling: tasks have an execution quantum (duration of time) and voluntarily yield the CPU to other tasks
    - Preemptive scheduling: highest priority task gets the CPU
- **Two modes of execution: thread or fiber**
  - Thread is default
  - Fiber can provide performance boost for specific usage scenarios
    - Rarely used in production environment
    - CLR not supported
    - XML not supported

SOS Workers (Threads)

- **Default configuration of 0**
  - 32-bit SQL Server
    - Actual number is: ((NumberOfSchedulers - 4) * 8) + 256
  - 64-bit SQL Server
    - Actual number is: ((NumberOfSchedulers – 4) * 16) + 512
- **Current maximum number of workers**

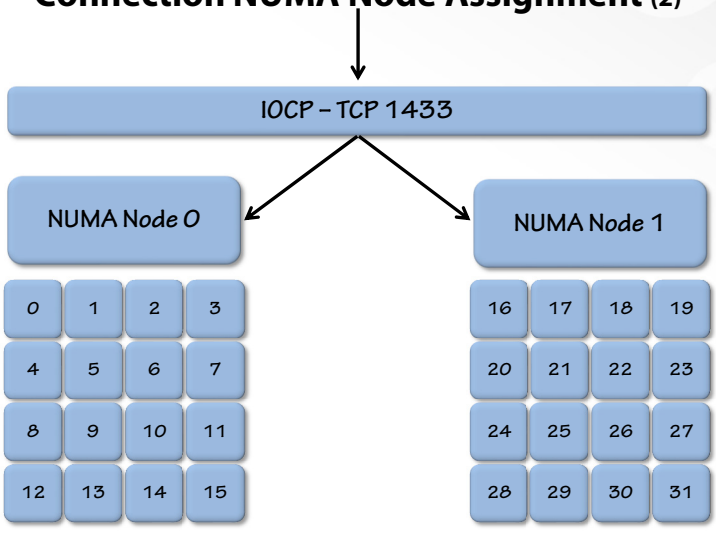
```
SELECT max_workers_count
FROM sys.dm_os_sys_info
```

- **Edge cases for changing default**
  - Database mirroring on 32-bit servers
  - Principal server uses 1 global thread, 2 threads per mirrored database
  - Mirror server uses 1 global thread, 2 threads per mirrored database and 1 additional thread per mirrored database for every 4 processors

Connection NUMA Node Assignment

- **Connection scheduling assignment starts at the NUMA node level**
- **The basic algorithm is a round-robin assignment for new connections across available NUMA nodes to the listener**
  - A separate I/O Completion Port (IOCP) listener can be associated to each NUMA node or a groups of nodes
  - It is possible to make a TCP connection followed by a Named Pipe connection and get connection affinity assignments to the same node
- **Within the NUMA node assigned to the connection by the round-robin algorithm, the connection is assigned to the scheduler with the smallest load factor**
  - Load factor is derived from the number of tasks assigned to the scheduler and can be seen in sys.dm_os_schedulers
  - The assigned scheduler becomes the preferred scheduler for the life of the connection

Connection NUMA Node Assignment (2)



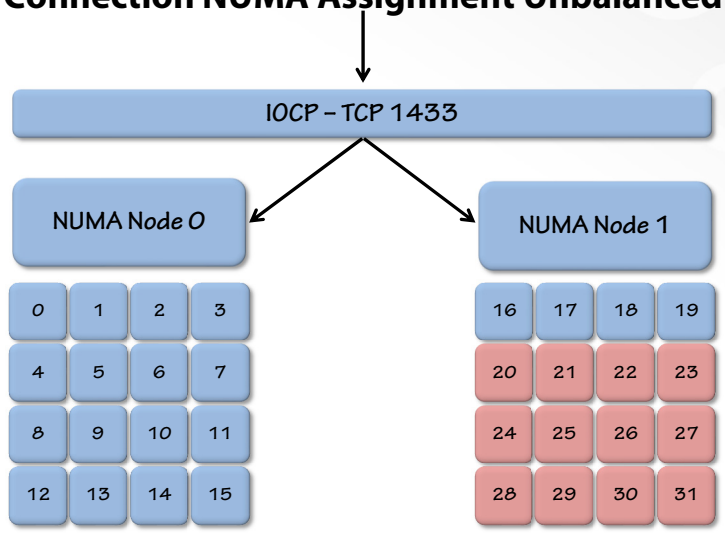
Task Scheduler Assignment

- **Each new command submitted by a client is assigned a controlling task in SQLOS**
- **The associated controlling task is assigned to a scheduler and remains associated with that scheduler for the life of the command execution**
- **Task assignment is also based on load factor, or the 120% rule:**
  - Once the preferred scheduler has120% more load than the other schedulers, on the same NUMA node, new tasks are assigned to other schedulers within the same NUMA node

Scheduler Licensing Limitations

- **SQL Server 2012+ introduced core based licensing limits**
  - Standard Edition limited to 4 sockets and 16 cores
    - VMs created with 8 sockets and 1 CPU per socket will only use 4 schedulers
  - Enterprise Server + CAL limited to 20 cores
- **Excess cores are assigned a scheduler in SQLOS but the scheduler will be offline**
  - Check for NUMA imbalance in sys.dm_os_schedulers
  - Manually set PROCESS AFFINITY using ALTER SERVER CONFIGURATION to balance the number of schedulers ONLINE across NUMA nodes
    - Prevents unbalanced connection and task assignment
    - http://sqlperformance.com/2012/11/system-configuration/2012-cal-problems

Connection NUMA Assignment Unbalanced



SQLskills immerse yourself in sql server

Parallel Query Scheduling (1)

- The controlling task always follows the preferred scheduler assignment rules in SQLOS
- For parallel sub-tasks the decision is based on the ideal node to get the most resources for the parallel execution (number of available schedulers, number of available workers, memory, DOP target for the query)
- Only online nodes are considered for parallel task scheduling
  - If the DOP target for the query fits within a single node, the ideal node is used to execute the parallel sub-tasks
    - This may be a different node than the controlling tasks and is not a problem because the majority of the work is done by the parallel workers which are local to each other on the same node
  - If the DOP target for the query does not fit within a single node, the work is spread across the available online nodes

SQLskills immerse yourself in sql server

Parallel Query Scheduling (2)

- **The internal resource state of the nodes is updated for the query execution engine approximately every 2 seconds**
  - A single node can be selected as the destination for multiple parallel queries when they begin executing at approximate the same time
  - Staggering large queries by more than 2 seconds can allow alternate node choices to be available to the execution engine

Parallel Query Placement Decision Logic

SMP or Connection Bound	Place threads on one node Requires: -T2479	Either the system only has a single node or it is treated as if the connection node is the only node on the system. Parallelism is allowed as long as free threads on the node are >= (dop * branches in query)
Full	Place threads on all nodes	Node zero will always be the starting node. Starting at node id zero SQL Server loops across schedulers and nodes until all workers are placed. DOP is allowed as long as all workers can be placed on the full system.
Least Loaded Node	Place threads on the least loaded node. Requires: -T2467	Loop over the online nodes determining if there are enough free threads on any single node to support the current DOP request. Making sure there are enough schedulers online within the node to support the request without stacking up requests from the query on the same scheduler(s.)
Use Next Node	Place threads within node Requires: -T2468	Find the next node that can service the DOP request. Unlike *full* mode, the global, resource manager keeps track of the last node used. Starting from the last position, and moving to the next node, SQL Server checks for query placement opportunities. If a node can't support the request SQL Server continues advancing nodes and searching.
Spread	Place threads on multiple nodes	This is the most common decision made by SQL Server. The decision spreads the workers across multiple nodes as required. The design is similar to *full* except the starting position is based on the saved, next node, global enumerator.

- https://blogs.msdn.microsoft.com/psssql/2016/03/04/sql-server-parallel-query-placement-decision-logic/

Scheduler Monitor

- **Per node monitoring for issues every 5 seconds**
  - 17883: non-yielding task, single scheduler
  - 17887: I/O completion stall, single scheduler
  - 17884: no work progressing, all schedulers
  - 17888: 50% of same resource, all schedulers
- **Ring buffer entries using sys.dm_os_ring_buffers and Extended Events**

Scheduling DMVs

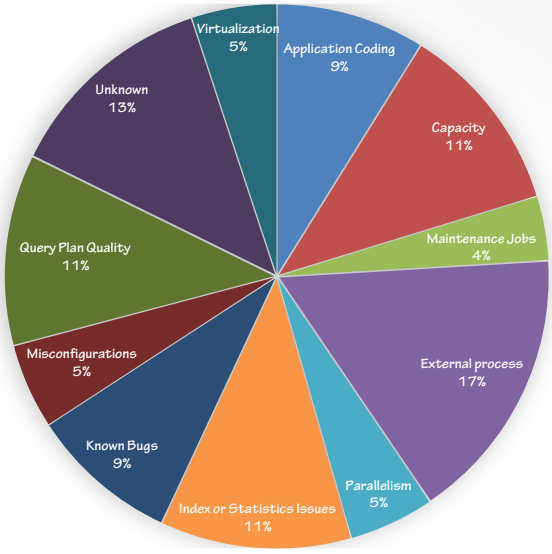
- **sys.dm_os_schedulers**
- **sys.dm_io_pending_io_requests**
- **sys.dm_os_workers**
- **sys.dm_os_tasks**
- **sys.dm_os_waiting_tasks**
- **sys.dm_os_wait_stats**

Demo

Scheduling DMVs

CPU Issue "Themes"

Informal categorization of 79 "high CPU" issues from serverfault.com, stackoverflow.com, recent clients, MSDN forums (snapshot from January 2013)



Defining CPU Performance Issues

- **Define the problem space:**
  - Availability impacted
    - For example "pegged CPUs" bring all else to a halt
    - Low CPU and no requests completing
  - Degradation of overall performance
    - Workloads run, but slower than desired (SLAs)
  - Capacity and scalability limitations
    - Workloads run fine, but throughput has ceiling
  - Workload competition
    - Some queries run fast, some slow: winners/losers
    - One request takes all the resources: the rest suffer
  - Capacity constrained
    - Power options?
    - Over-provisioned virtualization host?

Baseline Information (1)

- **Physical server**
  - Sockets, cores, hyper-threading
  - Processor model, architecture (32-bit/64-bit)
  - NUMA, L2/L3 cache sizes
- **Virtual server**
  - Host info (same as physical SQL Server) and guest VM virtual cores
  - Hyper-V CPU Reserve, VMware CPU Reservation
  - Hyper-V CPU Relative Weight, VMware CPU Shares
  - vCPUs and co-scheduling to other guests
- **SQL Server instance configuration settings:**
  - Max degree of parallelism setting
  - Cost threshold for parallelism option
  - Processor affinity setting
  - Priority boost setting
  - Max worker threads setting
  - Lightweight pooling setting

Baseline Information (2)

- **CPU power-option settings:**
  - Windows
    - High Performance, Balanced, Power Saving
  - ESX
    - See VMWare KB article 1018206 (http://kb.vmware.com/kb/1018206)
- **Resource Governor configuration (more on this later)**
- **SQL Server error log**
  - We'll discuss CPU related DMVs, but it helps to take a quick look for unusual errors and warnings prior to going deeper
    - You may see downstream errors as well
- **Windows Event Logs**
  - Look for hardware related errors (drivers, failures, degradation)

Demo

CPU-Z, powercfg.cpl, and sys.configurations

But is SQL Server the Culprit?

- **Don't jump to conclusions until you've confirmed that the issue is indeed SQL Server**
  - Process: % Privileged Time (kernel mode)
  - Process: % User Time (user mode)
  - Task Manager
  - Or if you're already connected to SQL, you can rule out SQL itself via sys.dm_os_ring_buffers (more on this later)

Kernel or User Time?

- **Validated through Process: high SQL % Privileged Time (sqlservr object)**
  - Multiple instances?
    - Map SELECT SERVERPROPERTY ('processid') to PID
- **sys.dm_os_ring_buffers**
  - 100 – (SystemIdle + ProcessUtilization)

SQLSERVR Privileged Time Causes

- **What could cause high kernel (privileged) time for SQL?**
  - Filter-drivers inject themselves into the Windows driver stack (Anti-virus, encryption services)
  - E.g. large I/O operations causing high CPU (privileged time)
    - Windows Server 2008 R2 Hotfix KB 976700
  - Missing firmware updates or drivers
  - Defective or significantly insufficient I/O components

Demo

Measuring % User vs. Kernel time

What About Wait Statistics?

- **Wait statistics can point to CPU pressure**
  - High ratio of signal wait time to resource wait time
  - SOS_SCHEDULER_YIELD – when used in conjunction with other diagnostic data
  - CXPACKET – again, when used in conjunction with other diagnostic data
- **Otherwise, wait statistics point to contention for other resources (synchronization, I/O, network, memory, locking)**

What About Virtualization?

- **Is the guest on an over-committed vCPUs on virtual machine host?**
- **Even if not over-committed – are there co-scheduling issues causing a Ready state?**
- **What is CPU Ready Time (VMware) telling you?**
  - Covered further in IE3, but see *CPU Ready Time in VMware and How to Interpret its Real Meaning* (http://bit.ly/UTe7Wr) for further information

If It *Is* SQL User Time, What Next?

- **We next have to first identify which queries are driving the CPU consumption**
- **Common patterns:**
  - One "bad" serial query executed by multiple requests
    - Easier to prioritize (no triage required)
  - One "bad" parallel query using all schedulers
    - Again, this is easier to identify
  - Numerous requests consuming schedulers ("death by 1,000 cuts")
    - Look at cumulative worker time (we'll discuss)

Correlation of CPU to I/O

- **You may see high CPU queries responsible for minor amounts of I/O**
  - Calculations, loops, conversions
- **You'll also see high CPU with high I/O**
- **Resolving an I/O issue often solves your CPU issue as well**
  - Adding more memory or improving the I/O path may not help you though, because logical I/O still drives CPU
    - A reason why Resource Governor is limited in this scenario

Demo

The CPU and I/O relationship

Resist the Urge to Do the Following…

- **Not sure of the root cause yet? Don't fall back on the following "solutions" unless you want to risk masking the root cause:**
  - Update all statistics
  - Rebuild all indexes
  - Restart SQL Server
- **Example: you see high CPU so you blindly update all statistics and that "solves the problem" (for now)**
  - Why is this a problem?
    - The statistics caused the problem procedure to recompile, removing the actual issue (e.g. parameter sniffing), and the recompiled plan is "good" but will revert to "bad" at a future time (problem not solved!)

The Observer Effect

- **When possible, demote SQL Profiler and Trace in favor of other methods in order to avoid adding your own overhead**
- **Factors which increase overhead:**
  - Types of events tracked (e.g. Showplan XML Statistics Profile)
  - Volume of requests
  - Use of SQL Profiler versus server-side traces
  - Distance of SQL Profiler to the server
  - Number of concurrent traces

Live Troubleshooting Advice

- **Its okay to use pre-canned queries and procedures, but be sure you understand what exactly they're doing**
  - Procedures that create intermediate result sets or are calculation-intensive may not complete execution on a heavily-constrained system
  - Understand the core DMVs and don't be afraid to run them in isolation and in smaller pairings (execute-collect-execute)
- **Don't forget about the Dedicated Admin Connection (DAC)**

Answering Questions with DMVs (1)

- **Which request is running right now?**
  - sys.dm_exec_requests
- **What is it executing?**
  - sys.dm_exec_sql_text
- **Where is it coming from?**
  - sys.dm_exec_sessions
  - sys.dm_exec_connections
- **What's its plan? (careful on this one)**
  - sys.dm_exec_query_plan or
  - sys.dm_exec_text_query_plan (for very large plans)
- **Who's waiting on the scheduler and why?**
  - sys.dm_os_waiting_tasks

Answering Questions with DMVs (2)

- **Which queries have taken up the most CPU time since the last restart?**

- **sys.dm_exec_query_stats**
  - Aggregate by total_worker_time
  - Define averages with execution_count
  - If ad hoc workloads, group by query_hash or query_plan_hash
  - Use the plan_handle with sys.dm_exec_query_plan to grab the plan

Answering Questions with DMVs (3)

- **Is this query using parallelism?**

- **sys.dm_os_tasks**
  - Ordered by session_id, request_id
- **sys.dm_exec_query_plan**
  - Look at plan operators
- **sys.dm_exec_query_stats**
  - Filter total_elapsed_time less than total_worker_time
    - Can be a false negative for blocking scenarios – where duration is inflated due to a wait on resource

Resource Governor

- **A user-defined classifier function runs at login and evaluates properties (e.g. name, workstation, database) and assigns incoming connections to a pre-defined workload group**
- **A workload group is bound to a resource pool**
- **A resource pool can be associated with >=1 workload groups**
- **Monitor resource usage by resource pool and workload group**
- **Dynamically alter any of the above**
- **You can use it to:**
  - Implement quotas, with a chargeback system
    - SQL Server hosting partners
  - Learn about aggregated resource usage on a system
  - Limit concurrent users, parallelism
  - Avoid runaway queries
  - Guarantee resources for critical workloads

Concepts: Classification

- **Incoming connections are classified into workload groups**
- **Classification function is a T-SQL user-defined function**
  - Takes no parameters and returns group name as a sysname type
- **Various functions to use for classification:**
  - HOST_NAME, APP_NAME, SUSER_NAME, IS_MEMBER, new CONNECTIONPROPERTY
- **Easy to think of scenarios where connections classified by time of day too**
  - e.g. batch processing or maintenance
- **Registered with the resource governor using:**
    - ALTER RESOURCE GOVERNOR WITH (
    - CLASSIFIER_FUNCTION = <name>)
- **NOTE: Once a connection is assigned to a group it cannot be changed (i.e. you can't further throttle an active query)**
- **NOTE: Dedicated Admin Connection bypasses all of this**
- **Beware of classifier functions (or logon triggers) that take so long to execute that the query timeout limit is reached**

Concepts: Resource Pools

- **A resource pool is a way to limit resource consumption for one or more workload groups**
  - (2008/2008 R2) 18 pools, (2012+) 62 user-definable resource pools
  - Syntax: ALTER / CREATE / DROP RESOURCE POOL
- **CPU related options (with defaults in bold):**
  - MIN_CPU_PERCENT = value (**0**)
  - MAX_CPU_PERCENT = value (**100**)
    - Only enforced when multiple workers use a single scheduler
    - This can be confusing at first… an opportunistic maximum…
  - (2012+) CAP_CPU_PERCENT = value (**100**)
    - Hard cap limit on CPU resource usage
  - (2012+) AFFINITY = AUTO
    - Attach pool to specific schedulers or NUMA nodes
      - For example… WITH (AFFINITY SCHEDULER = (0))
    - If using this setting, be sure to reconfigure after CPU changes – VM's

Concepts: Workload Groups

- **A workload group allows grouping connections into a 'bucket'**
  - E.g. batch processes, reports, executives, helpdesk, maintenance
- **Syntax: CREATE / ALTER / DROP WORKLOAD GROUP**
- **Assigned set of resource limits by mapping to a resource pool**
- **Internal tasks (e.g. checkpoint, ghost cleanup) always are part of the internal group (cannot be changed)**
  - No limits on CPU or memory, will pressurize all other groups, regardless of their limits
- **Unassigned connections go into the default group**
- **CPU related options (with defaults in bold)**
  - IMPORTANCE = {LOW | **MEDIUM** | HIGH}
    - Not the same as thread priority; applies to position in RUNNABLE queue on a single scheduler for workers from groups using the same pool - LOW:MEDIUM:HIGH = 1:3:9 ratio
  - REQUEST_MAX_CPU_TIME_SEC = value (**0**)
    - Max amount of CPU time a request can use before event is fired
  - MAX_DOP = value (**0**)
    - Max degree of parallelism. Overrides the sp_configure option

MAXDOP Settings

- **Server-wide setting applies as a limit…**
- **Except that DATABSE SCOPED CONFIGURATION (2016+) will override it**
  - MAXDOP of the database applied
  - Watch out for tempdb and #temp tables – tempdb MAXDOP may apply
- **Except that MAXDOP query hint will override it**
  - And anyone can specify a MAXDOP hint that overrides server-wide and database scoped MAXDOP
- **Except if the query is running under Resource Governor in a workload group that specifies a non-zero MAX_DOP**
  - Workload group MAX_DOP cannot be exceeded

Demo

Resource Governor and parallelism

Limitations

- **Works with the Database Engine only**
- **Single instance only**
  - Each instance controlled individually
- **Controls for CPU usage and memory allocation ONLY**
  - CPU usage includes CLR, but not pre-emptive operations
  - CPU governing occurs per scheduler
  - Certain workloads may not be entirely suited, e.g. short-lived OLTP queries
- **No way to tell if a specific query was throttled in any way**

Key Takeaways

- Ensure that the server Power Options are configured for High Performance as a part of the SQL Server installation checklist
- Look at Kernel and User time for the sqlservr process in Performance Monitor whenever troubleshooting high CPU issues to determine where the CPU usage is actually going
- Check for Extended Events sessions and SQL Traces running on the server to verify that observer overhead isn't part of the problem

Review

- Understanding Windows scheduling
- Server hardware and NUMA
- CPU scheduling under SQLOS
- DMV monitoring
- Troubleshooting CPU performance issues
- Using Resource Governor to limit CPU usage


Questions?
SQLskills
immerse yourself in sql server