

SQLskills Immersion Event IE2: Performance Tuning

Module 5: Waits and Queues

Paul S. Randal
Paul@SQLskills.com



Overview

- Introduction
- Thread lifecycle
- DMVs
- Common wait types

- Doctor, Doctor, my knee hurts...



2

© SQLskills, All rights reserved.
<http://www.SQLskills.com>



Don't Assume Symptom=Root Cause

- **Performance troubleshooting is not an exact science**
 - The same symptoms can result from many root causes
- **For example, how many different things could cause I/O latencies?**
 - Overloaded/incorrectly-configured I/O subsystem
 - Synchronous I/O-subsystem mirroring
 - Buffer pool memory pressure
 - From plan cache bloat
 - From external Windows pressure
 - From an ad hoc query
 - From an inefficient query plan
 - Network latency
 - And more...

Interpreting the Data

- **Don't do 'knee-jerk' performance tuning**
 - Work through the data to see what may be the root cause
 - You'll end up spending less time overall
- **Proficiency in using wait statistics data comes from:**
 - Retrieving the data correctly
 - Understanding what common wait types mean
 - Recognizing patterns
 - Avoiding inappropriate Internet advice
 - Practice!
- **Even better is to have a series of snapshots of wait statistics over time**
 - Allows identification of changes and the time of the change
 - Allows trending

What are Waits?

- The term 'wait' means that a thread running on a processor cannot proceed because a resource it requires is unavailable
 - It has to wait until the resource is available
- The resource being waited for is tracked by SQL Server
 - Each resource maps to a wait type
- Example resources that may be unavailable:
 - A lock (LCK_M_XX wait type)
 - A data file page in the buffer pool (PAGEIOLATCH_XX wait type)
 - Results from part of a parallel query (CXPACKET wait type)
 - A latch (LATCH_XX wait type)

What are Queues?

- The term 'queues' is a generic term describing what is preventing a resource being available to the thread that is waiting for it
 - This does not mean all resources are held in actual queue structures
- Example queues:
 - For a LCK_M_XX wait, another thread holding an incompatible lock
 - For a PAGEIOLATCH_XX wait, the I/O subsystem needs to complete the I/O
 - For a CXPACKET wait, another thread needs to complete its portion of work
 - For a LATCH_XX wait, another thread holding an incompatible latch
- Queues are investigated using DMVs, performance counters, and other tools

Waits and Queues Methodology

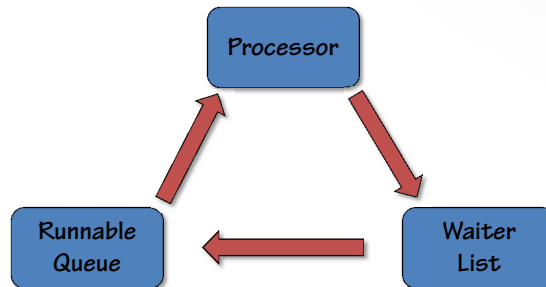
- **'Waits and queues' name coined by Tom Davidson in 2005**
 - Whitepaper: Performance Tuning Using Waits and Queues
 - The 'bible' of using wait statistics
 - Available at <http://bit.ly/aUh6S>
- **Very powerful method to get initial direction on a problem**
 - Avoid flailing and investigating the wrong problem
 - Can also show problems that are not obvious
- **Most commercial performance monitoring tools capture and show wait statistics**
 - Many free tools also do this, such as the popular Who Is Active tool
- **Various releases of SQL Server have provided wait statistics views**
 - SQL Server 2005 had Performance Dashboard reports
 - SQL Server 2008 onwards has the Management Data Warehouse

Thread Scheduling

- **SQL Server performs its own thread scheduling**
 - Called non-preemptive scheduling
 - More efficient (for SQL Server) than relying on Windows scheduling
 - Performed by the SQLOS layer of the Storage Engine
- **Each processor core (whether logical or physical) has a scheduler**
 - A scheduler is responsible for managing the execution of work by threads
 - Schedulers exist for user threads and for internal operations
 - Use the sys.dm_os_schedulers DMV to view schedulers
- **When SQL Server has to call out to the OS, it must switch the calling thread to preemptive mode so the OS can interrupt it if necessary**

Components of a Scheduler

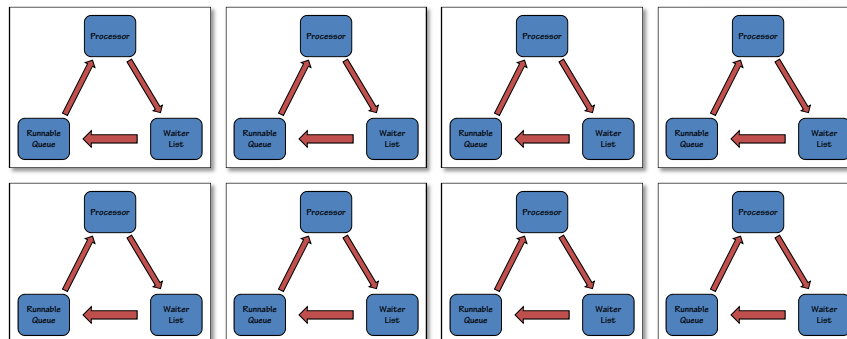
- All schedulers are composed of three 'parts'



- Threads transition around these parts until their work is complete

Schedulers in SQL Server

- One scheduler per logical or physical processor core
 - Plus some extra ones for internal tasks and the Dedicated Admin Connection
- For example, for a server with four physical processor cores, with hyper-threading enabled, there will be eight user schedulers

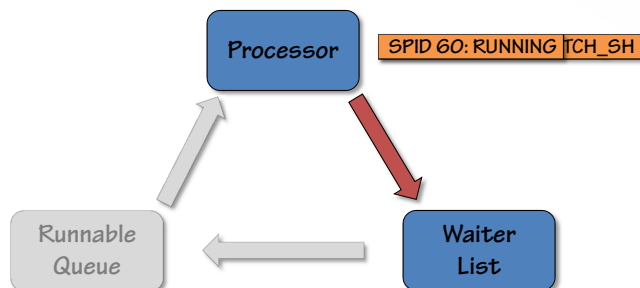


Thread States

- A thread can be in one of three states when being actively used as part of processing a query
- **RUNNING**
 - The thread is currently executing on the processor
- **SUSPENDED**
 - The thread is currently on the Waiter List waiting for a resource
- **RUNNABLE**
 - The thread is currently on the Runnable Queue waiting to execute on the processor
- Threads transition between these states until their work is complete

Transition: RUNNING to SUSPENDED

- A thread continues executing on the processor until it must wait for a resource to become available
 - The thread's state changes from RUNNING to SUSPENDED
 - The thread moves to the Waiter List
 - This process is called being 'suspended'

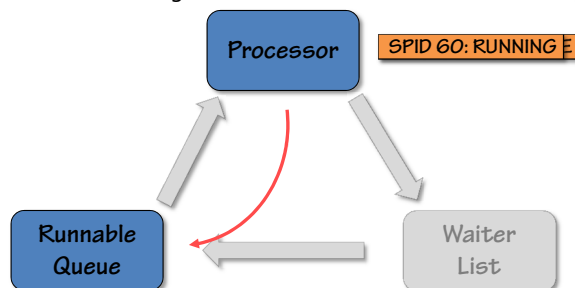


The Waiter List

- The Waiter List is an unordered list of threads that are suspended
- Any thread can be notified at any time that the resource it is waiting for is now available
 - Again, absolutely no ordering
- No limit to how long a thread remains on the waiter list
 - Although execution timeouts or lock timeouts may take effect
- There is no practical limit to how many threads can be on a scheduler's waiter list at any time
- The sys.dm_os_waiting_tasks DMV shows which threads are currently waiting and what they are waiting for

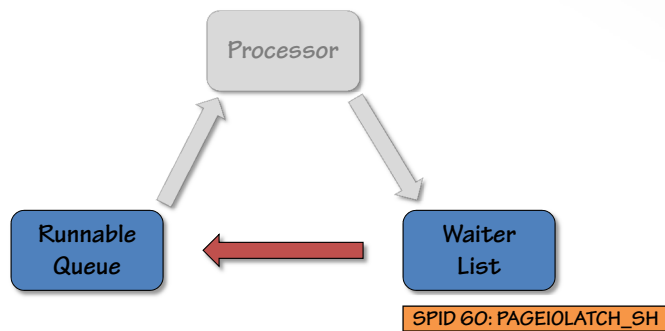
Special Case: Quantum Exhaustion

- If a thread does not need to wait for a resource, it will continue executing until its quantum is exhausted
 - Thread quantum is fixed at 4 milliseconds and cannot be changed
- If this occurs, thread moves to bottom of the Runnable Queue
 - The thread's state changes from RUNNING to RUNNABLE



Transition: SUSPENDED to RUNNABLE

- A thread continues to wait until it is told that the resource is available
 - The thread's state changes from SUSPENDED to RUNNABLE
 - The thread moves to the bottom of the Runnable Queue
 - This process is called being 'signaled'



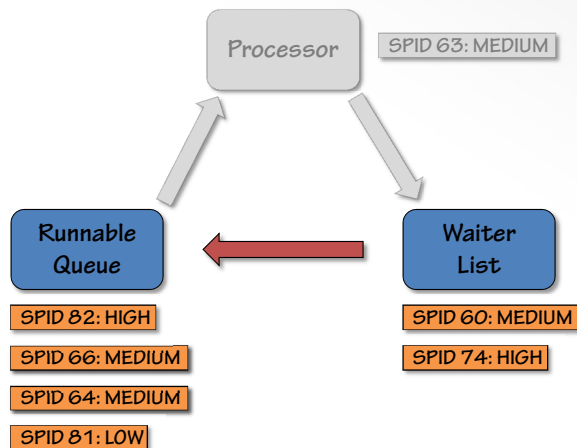
The Runnable Queue

- The Runnable Queue is a strict First-In-First-Out (FIFO) queue
 - There is a special case that will be discussed on the next slide
- Threads enter the queue at the bottom and progress to the top
- The thread at the top of the queue is the one that will execute on the processor when the processor becomes free
 - When the currently executing thread is suspended or exhausts its quantum
- The size of the Runnable Queue can be seen from the `runnable_tasks_count` column in `sys.dm_os_schedulers`

Special Case: Resource Governor

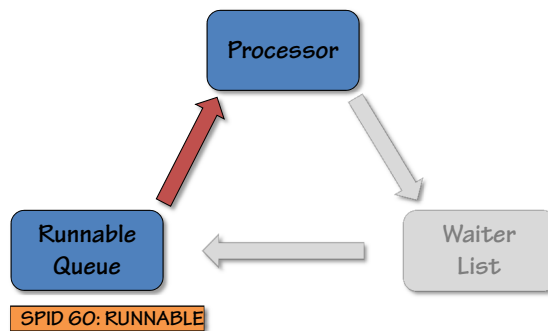
- **Using Resource Governor (in Enterprise Edition of SQL Server 2008 onwards) it is possible to define multiple workload groups that share the same resource pool**
 - A workload group can be thought of as a bucket of connections
 - A resource pool can be thought of as a set of CPU and memory limits, but more details are beyond the scope of this module
- **Multiple workload groups for a resource pool can be assigned relative priorities**
 - The default priority is medium
 - Possible values are high, medium, and low
- **The relative priorities change how the Runnable Queue works**
 - High to medium to low equates to 9 to 3 to 1 in terms of the priority of the threads that are permitted to execute
 - For example, for each low-priority thread on the Runnable Queue, nine high-priority threads will be allowed to jump over it and execute first

Resource Governor Example

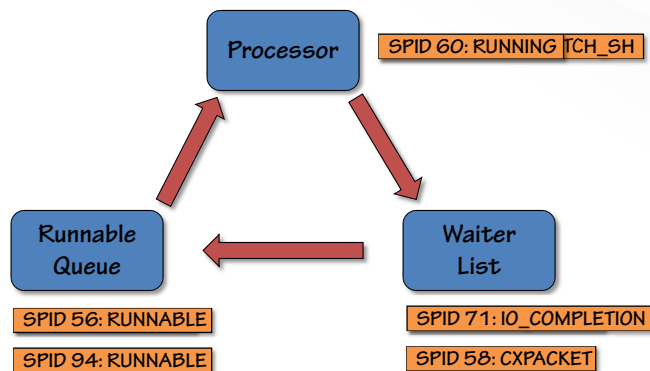


Transition: RUNNABLE to RUNNING

- The thread waits on the Runnable Queue until it reaches the top and the processor becomes available
 - The thread's state changes from RUNNABLE to RUNNING



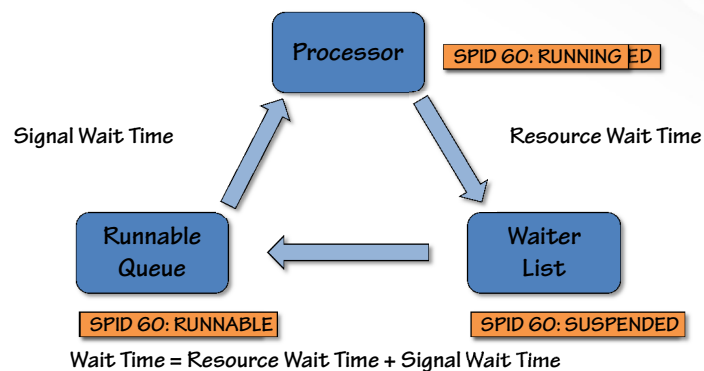
Pulling It All Together



Wait Times Definition (1)

- **Total time spent waiting:**
 - Known as 'wait time'
 - Time spent transitioning from RUNNING, through SUSPENDED, to RUNNABLE, and back to RUNNING
- **Time spent waiting for the resource to be available:**
 - Known as 'resource wait time'
 - Time spent on the Waiter List with state SUSPENDED
- **Time spent waiting to get the processor after resource is available:**
 - Known as 'signal wait time'
 - Time spent on the Runnable Queue with state RUNNABLE
- **Wait time = resource wait time + signal wait time**

Wait Times Definition (2)



sys.dm_os_waiting_tasks DMV

- This DMV shows all threads that are currently suspended
- Think of it as the 'what is happening right now?' view of a server
- Most useful information this DMV provides:
 - Session ID and execution context ID of each thread
 - Wait type for each suspended thread
 - Description of the resource for some wait types
 - E.g. for locking wait types, the lock level and resource is described
 - Wait time for each suspended thread
 - If the thread is blocked by another thread, the ID of the blocking thread
 - Useful to find what's at the head of a blocking chain
 - Can show non-intuitive patterns
- Usually very first thing to run when approaching a 'slow' server
 - The data is more useful when joined with other DMV results

sys.dm_os_wait_stats DMV

- This DMV shows aggregated wait statistics for all wait types
 - Aggregated since the server started or the wait statistics were cleared
- Think of this as the 'what has happened in the past?' view of a server
- This DMV provides:
 - The name of each wait type
 - The number of times a wait has been for this wait type
 - The aggregate overall wait time for all waits for this wait type
 - The maximum wait time of any wait for this wait type
 - The aggregate signal wait time for all waits for this wait type
- Some math is required to make the results useful
 - Calculating the resource wait time
 - Calculating the average times rather than the total times

Filtering Benign Waits

- **An extremely important point to bear in mind is that waits ALWAYS occur inside SQL Server**
 - I.e. just because waits exist does not mean there is a perf problem
- **Rather than looking at all waits, most useful is to focus on highly prevalent wait types**
 - More processing of the sys.dm_os_wait_stats results is required
 - Common method is to show the top 95% of all waits by wait time
- **Some wait types are almost always benign and can be safely ignored**
 - Some have pathological, very rare cases where they can be problematic
- **For example, the WAITFOR wait type**
 - Only occurs when a WAITFOR DELAY statement is executed
 - When filtering the top 95% of waits by total wait time, not filtering out this wait can badly skew the results

Storing Wait Statistics

- **Capturing wait statistics information over time allows:**
 - Trending
 - Point-in-time analysis to see when a problem started to occur
- **Simple method:**
 - Use sys.dm_os_wait_stats demo script and add a GETDATE () call
 - Store the results in a table
 - Create SQL Agent job to capture the wait statistics every hour or so
 - Create another SQL Agent job to purge wait statistics older than a month

Using Extended Events

- **Extended Events were added in SQL Server 2008 to all Editions**
 - Very lightweight mechanism for tracing activity in SQL Server
 - However, mis-configuration can make Extended Events very expensive
- **When a wait starts and ends, the `sqlos.wait_info` event fires**
 - Captures similar information to `sys.dm_os_wait_stats`
- **When a preemptive wait starts and ends, the `sqlos.wait_info_external` event fires**
 - Used when a thread is waiting for a call out to the OS and has to switch from non-preemptive to preemptive scheduling
- **Using the Extended Events system allows:**
 - Capturing of all wait types for a single operation
 - Monitoring for specific wait types occurring
 - Advanced analysis of SQL Server internals

What are Latches?

- **A latch is a synchronization mechanism between threads**
 - Many people equate latches with locks, but they are quite different
- **A latch protects access to an in-memory data structure**
 - Whereas a lock protects transactional consistency
- **Latches are lightweight and are held only for a short time**
 - Whereas a lock may be held until the end of a transaction
- **Latches cannot be controlled by SQL Server users**
 - Whereas locks can be controlled with hints and configuration options
- **Latches have a variety of modes, equating to the level of access to the in-memory data structure that is required**
 - E.g. an EX latch is required to change a data structure, and a SH latch is required to read most data structures
 - This is similar to the modes that a lock can have
- **SQL Server tracks latch wait times just like other waits**

Types of Latches

- **There are three types of latches:**
 - Latches waiting for data file pages to be read from disk into memory
 - Manifest as PAGEIOLATCH_XX waits
 - Latches for access to in-memory data file pages
 - Manifest as PAGELATCH_XX waits
 - Latches for access to all other data structures
 - Manifest as LATCH_XX waits
- **Examples of non-page latches:**
 - FGCB_ADD_REMOVE
 - ACCESS_METHODS_HOBT_VIRTUAL_ROOT

© SQLskills, All rights reserved.
<http://www.SQLskills.com>

B-tree Page Split Example

The diagram illustrates a B-tree structure during a page split. At the top is a green circle labeled "Virtual Root". Below it is a blue rectangle representing a page, which is being split (indicated by a jagged line). A red arrow labeled "EX" points to this page from the left, and another red arrow labeled "EX" points away from it to the right, labeled "PAGELATCH". Below this page are two more blue rectangles, labeled "Prev" and "Next". A third blue rectangle is shown below "Next", labeled "LOCK". Red arrows labeled "EX" and "PAGELATCH" point to the "Prev" and "Next" pages respectively. A red arrow labeled "X" points to the "LOCK" page. The text "LATCH (ACCESS_METHODS_HOBT_VIRTUAL_ROOT)" is written to the left of the top page. A box on the left contains the text: "From slide deck by Thomas Kejser (with permission)".

LATCH
(ACCESS_METHODS_HOBT_VIRTUAL_ROOT)

From slide deck by
Thomas Kejser
(with permission)

Virtual Root

EX

EX

PAGELATCH

EX

PAGELATCH

EX

EX

Prev

Next

LOCK

X

Latch Contention

- **Just like with locks, latches can be a source of contention**
 - This means that what appears to be traditional blocking involving locks may actually be blocking involving latches
- **If one thread has a latch held exclusively then other threads must wait until that thread releases the exclusive latch**
- **This does not become a performance problem until there are many concurrent threads competing for access to the same latch**
 - As latches are only held for a short duration, a single thread waiting a very short time for another thread does not cause a problem
 - However, if hundreds of threads are waiting for a single thread, then that aggregates into a noticeable performance problem
- **Whitepaper on investigating latch contention:**
 - <http://bit.ly/pS1kd1>

sys.dm_os_latch_stats DMV

- **This DMV shows aggregated wait statistics for all non-page latch classes**
 - Aggregated since the server started or the latch statistics were cleared
- **This DMV provides:**
 - The name of each latch class
 - The number of times a wait has been for this latch class
 - The aggregate overall wait time for all waits for this latch class
 - The maximum wait time of any wait for this latch class
 - It does NOT list the latch modes being acquired
- **Some math is required to make the results useful**
 - Calculating the average times rather than the total times

Clearing Wait and Latch Statistics

- Clearing the aggregated wait statistics can be done at any time using the code below:
 - DBCC SQLPERF ('sys.dm_os_wait_stats', CLEAR);
- And for latch statistics:
 - DBCC SQLPERF ('sys.dm_os_latch_stats', CLEAR);
- Clearing the wait statistics allows the effect of a workload change to be measured against previous wait statistics
- Be careful if you are taking periodic snapshots of wait statistics as this will invalidate your series of snapshots
- When were they last cleared?
 - <http://www.sqlskills.com/blogs/erin/post/Finding-When-Wait-Stats-Were-Last-Cleared.aspx> (<http://bit.ly/OkXQVC>)

Using Extended Events

- For very advanced troubleshooting there are events that allow tracking of latches
- For latches:
 - sqlserver.latch_suspend_begin
 - sqlserver.latch_suspend_end
 - These are similar to the sqllos.wait_info and sqllos.wait_info_external events but have a lot more information about the latch itself
- These are used rarely and are included here for completeness

Transaction Log Example

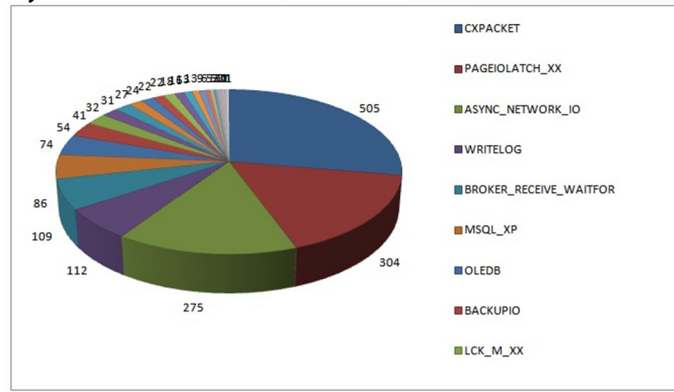
- Taking the transaction log and the logging system as an example, there are waits, and latches associated with it
- Waits:
 - WRITELOG, LOGBUFFER, LOGGENERATION, LOGMGR, LOGMGR_FLUSH, LOGMGR_QUEUE, LOGMGR_RESERVE_APPEND
- Latches:
 - LOG_MANAGER, LOGBLOCK_GENERATIONS
- From waits to latches, understanding the uses and troubleshooting becomes progressively harder and less likely to be required
 - This is common across the SQL Server engine

What's Relevant?

- Just because there are waits, does not mean they are the problem
 - Look for actionable items and filter out things like background tasks
 - Look at the demo code to see what I mean
- Need to identify the top, relevant waits and then drill in
- Example:
 - 1000 waits for LCK_M_S
 - Is it a problem?
 - No, if that was over 8 hours, there were 10 million locks acquired, and total wait time for the LCK_M_S locks was only 50s altogether
 - Yes, if each wait was for 50s
- If signal wait times are low, CPU pressure is not an issue
 - CPU pressure addressed in Module 9

Top Wait Types

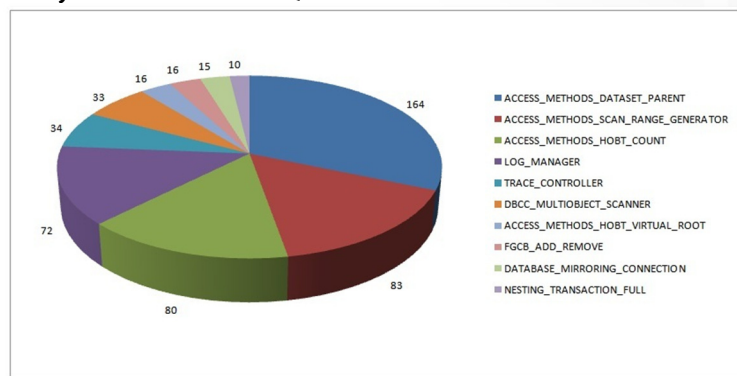
- Survey results from 1823 SQL Server instances across Internet



- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/Wait-statistics-or-please-tell-me-where-it-hurts.aspx> (<http://bit.ly/fSWeO5>)

Top Latch Classes

- Survey results from 581 SQL Server instances across Internet



- Source: [http://www.sqlskills.com/BLOGS/PAUL/post/Most-common-latch-classes-\(survey-results\).aspx](http://www.sqlskills.com/BLOGS/PAUL/post/Most-common-latch-classes-(survey-results).aspx) (<http://bit.ly/QXQB44>)

PAGEIOLATCH_XX Wait

- **What does it mean:**
 - Waiting for a data file page to be read from disk into memory
 - Common modes to see are SH and EX
 - SH mode means the page will be read
 - EX mode means the page will be changed
- **Avoid knee-jerk response:**
 - Do not assume the I/O subsystem or I/O path is the problem
- **Further analysis:**
 - Determine which tables/indexes are being read
 - Analyze I/O subsystem latencies with sys.dm_io_virtual_file_stats and Avg Disk secs/Read performance counters
 - Correlate with CXPACKET waits, suggesting parallel scans
 - Examine query plans for parallel scans and implicit conversions
 - Investigate buffer pool memory pressure and Page Life Expectancy

PAGEIOLATCH_XX Wait Solutions

- Create appropriate nonclustered indexes to reduce scans
- Update statistics to allow efficient query plans
- Move the affected data files to faster I/O subsystem
- If data volume has simply increased, consider increasing memory

PAGELATCH_XX Wait

- **What does it mean:**
 - Waiting for access to an in-memory data file page
 - Common modes to see are SH and EX
 - SH mode means the page will be read
 - EX mode means the page will be changed
- **Avoid knee-jerk response:**
 - Do not confuse these with PAGEIOLATCH_XX waits
 - Does not mean add more memory or I/O capacity
- **Further analysis:**
 - Determine the page(s) that the thread is waiting for access to
 - Analyze the queries encountering this wait
 - Analyze the table and index structures involved

PAGELATCH_XX Wait Solutions

- **Classic tempdb contention**
 - Add more tempdb data files
 - Enable trace flag 1118
 - Reduce temp table usage
- **Excessive page splits occurring in indexes**
 - Change to a non-random index key
 - Avoid updating index records to be longer
 - Provision an index FILLFACTOR to alleviate page splits
- **Insertion point hotspot in a clustered index with an ever-increasing key**
 - Spread the insertion points in the index using a random or composite key, plus provision a FILLFACTOR to prevent page splits
 - Shard into multiple tables/databases/servers

LCK_M_XX Wait

- **What does it mean:**
 - A thread is waiting for a lock that cannot be granted because another thread is holding an incompatible lock
- **Avoid knee-jerk response:**
 - Do not assume that locking is the root cause
- **Further analysis:**
 - Follow the blocking chain using sys.dm_os_waiting_tasks to see what the lead blocking thread is waiting for
 - Use the blocked process report to capture information on queries waiting too long for locks
 - See Michael Swart's blog post for details about the various methods and further links (<http://bit.ly/ki3bYI>)

LCK_M_XX Wait Solutions

- **Lock escalation from a large update or table scan**
 - Possibly configure partition-level lock escalation, if applicable
 - Consider a different indexing strategy to use nonclustered index seeks
 - Consider breaking large updates into smaller transactions
 - Consider using snapshot isolation, a different isolation level, or locking hints
 - All the general strategies for alleviating blocking problems
- **Unnecessary locks for the data being accessed**
 - Consider using snapshot isolation, a different isolation level, or locking hints
- **Something preventing a transaction from releasing its locks quickly**
 - Determine what the bottleneck is and solve it appropriately

Demo

Page latches and locks

WRITELOG Wait

- **What does it mean:**
 - Waiting for a transaction log block buffer to flush to disk
- **Avoid knee-jerk response:**
 - Do not assume that the transaction log file I/O system has a problem (although this is often the case)
 - Do not create additional transaction log files
- **Further analysis:**
 - Correlate WRITELOG wait time with I/O subsystem latency using `sys.dm_io_virtual_file_stats`
 - Look for LOGBUFFER waits, showing internal contention for log buffers
 - Look at average disk write queue length for log drive
 - If constantly 31/32 then the internal limit has been reached for outstanding transaction log writes for a single database
 - Look at average size of transactions
 - Investigate whether frequent page splits are occurring

WRITELOG Wait Solutions

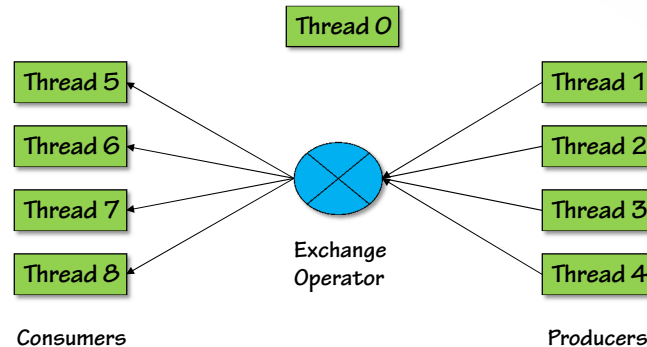
- Move the log to a faster I/O subsystem
- Increase the size of transactions to prevent many minimum-size log block flushes to disk
- Remove unused nonclustered indexes to reduce logging overhead from maintaining unused indexes during DML operations
- Change index keys or introduce FILLFACTORs to reduce page splits
- Potentially split the workload over multiple databases or servers

Demo

Slow transaction log

Parallel Threads Example

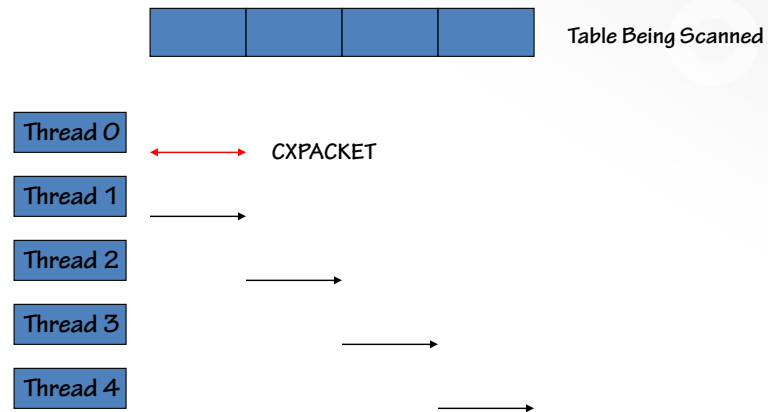
- As part of a query plan, you may see the  operator, for example
- This is a Repartition Streams operation
 - Uses producer and consumer threads, plus a control thread
- For a degree-of-parallelism = 4 operation, the threads would look like:



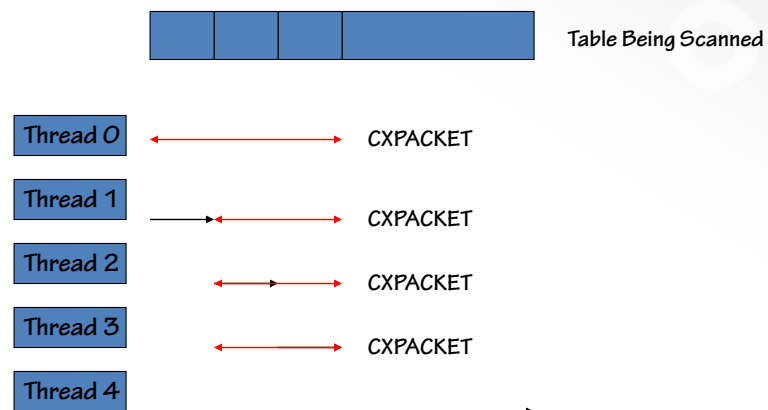
CXPACKET Wait Explanation

- **What does it mean:**
 - Parallel operations are taking place
 - Accumulating very fast implies skewed work distribution amongst threads or one of the workers is being blocked by something
- **Avoid knee-jerk response:**
 - Do not set server-wide MAXDOP to 1, disabling parallelism
- **Further analysis:**
 - Correlation with PAGEIOLATCH_SH waits? Implies large scans
 - Also may see with ACCESS_METHODS_DATASET_PARENT latch or ACCESS_METHODS_SCAN_RANGE_GENERATOR latch
 - Examine query plans of requests that are accruing CXPACKET waits to see if the query plans make sense for the query being performed
 - What is the wait type of the parallel thread that is taking too long? (i.e. the thread that does not have CXPACKET as its wait type)

CXPACKET Wait Example (1)



CXPACKET Wait Example (2)



CXPACKET Wait Solutions

- **Possible root-causes:**

- ❑ Just parallelism occurring
- ❑ Table scans being performed because of missing nonclustered indexes or incorrect query plan
- ❑ Out-of-data statistics causing skewed work distribution

- **If there is actually a problem:**

- ❑ Make sure statistics are up-to-date and appropriate indexes exist
- ❑ Consider MAXDOP for the query
- ❑ Consider MAXDOP = physical cores per NUMA node
- ❑ Consider MAXDOP for the instance, but beware of mixed workloads
- ❑ Consider using Resource Governor for MAX_DOP
- ❑ Consider setting 'cost threshold for parallelism' higher than the query cost shown in the execution plan

Demo

Parallelism

PREEMPTIVE_OS_XX Waits

- **What does it mean:**
 - A thread has called out to the OS
 - Threads must switch to preemptive mode when doing so
 - Note that the thread status will be RUNNING instead of SUSPENDED
- **Further analysis:**
 - 194 PREEMPTIVE_OS_XX waits in SQL Server 2012
 - These waits are very minimally and poorly documented
 - To determine what SQL Server is asking the OS to do, remove the PREEMPTIVE_OS_ prefix and search in MSDN for the remainder of the wait type, as it will be the name of a Windows API
- **Possible root-causes and solutions:**
 - Depends on the wait type
 - For instance, increasing PREEMPTIVE_OS_CREATEFILE waits occur when using FILESTREAM on an incorrectly prepared NTFS volume

PREEMPTIVE_OS_CREATEFILE Wait

- **What does it mean:**
 - A thread is calling out to Windows to create a file
 - Common to see lots of these when using FILESTREAM as each new FILESTREAM object is stored as an NTFS file
- **Further analysis:**
 - Watch for increasing wait times
- **Possible root-causes and solutions:**
 - The NTFS volume hosting the FILESTREAM data container(s) may not have 8.3 name generation and last access time tracking disabled
 - The NTFS volume hosting the FILESTREAM data container(s) may be on a portion of the I/O subsystem that is under heavy load

ASYNC_NETWORK_IO Wait

- **What does it mean:**
 - SQL Server is waiting for a client to acknowledge receipt of sent data
- **Avoid knee-jerk response:**
 - Do not assume that the problem is network latency
 - It is a network delay as far as SQL Server is concerned though
- **Further analysis:**
 - Analyze client application code
 - Analyze network latencies
- **Possible root-causes and solutions:**
 - Nearly always a poorly-coded application that is processing results one record at a time (RBAR = Row-By-Agonizing-Row)
 - Very easy to demonstrate using a large query and SQL Server Management Studio running on the same machine as SQL Server
 - Otherwise look for network hardware issues, incorrect duplex settings, or TCP chimney offload problems (see <http://bit.ly/aPzoAx>)

OleDb Wait

- **What does it mean:**
 - The OLE DB mechanism is being used
- **Avoid knee-jerk response:**
 - Do not assume that linked servers are being used
- **Further analysis:**
 - What are the queries doing that are waiting for OleDb?
 - If linked servers are being used, what is causing the delay on the linked server?
- **Possible root-causes:**
 - DBCC CHECKDB and related commands use OLE DB internally
 - Many DMVs use OLE DB internally so it could be a third-party monitoring tool that is repeatedly calling DMVs
 - Poor performance of a linked server

Demo

Other wait types

SOS_SCHEDULER_YIELD Wait

- **What does it mean:**
 - A thread exhausted its 4 millisecond quantum and voluntarily yielded
 - A thread is spinning on a spinlock and backing off every so often
- **Avoid knee-jerk response:**
 - Do not assume that CPU pressure is the problem
- **Further analysis:**
 - Examine query plans to see whether scans are occurring
 - Look to see whether there is a very small or non-existent number of PAGEIOLATCH_XX waits occurring, which indicates that the workload is memory-resident
 - Look for long Runnable Queues
 - Capture SQL Server code call stacks to see where the waits are occurring
- **Note: these waits have zero resource wait time so regular methods of aggregating and prioritizing waits will miss them**
 - They do not appear in sys.dm_os_waiting_tasks

SOS_SCHEDULER_YIELD Solutions

- **Possible root-causes:**

- SQL Server is executing code that can use a lot of CPU without having to wait for a resource (e.g. a large scan with few PAGEIOLATCH_SH waits)
- Look also for long Runnable Queues, indicating CPU pressure
- Spinlock contention

- **Solutions:**

- For the quantum exhaustion case on slower processors, potentially enable hyper-threading to give more schedulers and more potential for concurrent work, especially for OLTP workloads
- For the spinlock case, the solution varies by spinlock
 - More often than not, call stack analysis shows that spinlocks are not involved

Real-World Example: Symptoms

- **Auto dealership hosting service**

- Lots of auto dealers from across the US hosted on one site
- Each auto dealer uploads inventory each day
- One large Listing table storing all inventory for all dealers
- One large Visitor table tracking clicks on web pages
- No DBA

- **System had performance problem:**

- User queries on inventory and prices regularly timed out
- Inventory updates regularly timed out
- Climbing CPU usage
- Response time getting longer and longer
- Car dealers pressuring hosting service for fixes

- **First step: analyze wait statistics...**

Real-World Example: Analysis

- No historical data so gathered wait statistics data using the queries shown in earlier modules
- Both DMVs showed the same three wait types:
 - CXPACKET wait = parallelism
 - PAGEIOLATCH_SH wait = reading data file pages from disk
 - WRITELOG wait = waiting for log writes, with average wait more than 20ms
- Possible issues from just wait statistics
 - Many queries doing parallel table scans of data that is not memory resident
 - I/O subsystem for the log file over-loaded and/or high number of log flushes
- Investigated further using DMVs to analyze:
 - Query plans
 - Index and table structures, index usage, fragmentation, and statistics
 - I/O subsystem latencies
- Next step: determine root-causes...

Real-World Example: Root-Causes

- Both large tables had random GUID cluster keys
 - High fragmentation in the clustered indexes leading to poor readahead
 - Lots of page-split transaction log activity during web page click tracking
- Both large tables had more than 50 single-column nonclustered indexes
 - Indexes not being used for seeks, resulting in table scans
 - Large amounts of nonclustered index maintenance from inserts, updates, deletes contributing to transaction log activity
- Insufficient buffer pool memory for application workload data
- Poorly laid out I/O subsystem contributing to high latencies
- Poorly written code from using an ORM system
- Final step: propose and implement solution

Real-World Example: Solution

- **Solution included:**
 - Increasing server memory and provisioning more appropriate I/O subsystem
 - Changing main tables to have bigint IDENTITY cluster keys
 - Removing useless nonclustered indexes
 - Analyzing ORM-generated code and query plans to determine appropriate nonclustered indexes
 - ORM system could not be removed for political reasons
 - Implementing index maintenance and periodic health checks
- **End result: no performance problems and a happy client, with minimal investigation time**

Methodology: No Historical Data

- **Gather information about exactly when the performance problem arose and the user-visible characteristics of the problem**
- **Gather information about what changed before the problem arose**
- **Examine the output from sys.dm_os_waiting_tasks**
 - What is happening on the server right now?
- **Examine the output from sys.dm_os_wait_stats**
 - What has happened in the past?
- **Look at the top 3-4 relevant waits**
 - If LATCH_XX is present, examine the output from sys.dm_os_latch_stats
- **Avoid the temptation to knee-jerk and equate symptoms with the root-cause**
- **Gather further information from relevant sources to pin-point the problem**
 - DMVs, query plans, performance counters, code analysis

Methodology: Historical Data

- Gather information about exactly when the performance problem arose and the user-visible characteristics of the problem
- Gather information about what changed before the problem arose
- Examine the historical data sets from before the change and correlate through the time the problem arose
 - Look to see how the pattern of waits changes over time
- Investigate current output from `sys.dm_os_wait_stats` and `sys.dm_os_waiting_tasks` to see whether the pattern is continuing or has returned to 'normal'
 - If normal now, correlate with any other historical data captured from monitoring tools to try to determine why the problem occurred
 - If not normal, proceed with further data gathering, focusing on the waits that have risen to prevalence over time

Resources

- **Whitepapers:**
 - Performance Tuning Using Waits and Queues
 - Diagnosing and Resolving Latch Contention on SQL Server
 - Diagnosing and Resolving Spinlock Contention on SQL Server
 - Gnarly links – see top of www.SQLskills.com/whitepapers.asp
- **Blog: Wait statistics, or please tell me where it hurts**
 - <http://www.sqlskills.com/BLOGS/PAUL/post/Wait-statistics-or-please-tell-me-where-it-hurts.aspx> (<http://bit.ly/fSWeO5>)
- **Websites**
 - Service Broker Wait Types – SQL SSB Team
 - http://blogs.msdn.com/b/sql_service_broker/archive/2008/12/01/service-broker-wait-types.aspx (<http://bit.ly/jqmwZC>)
 - PSS Wait Stats Repository
 - <http://blogs.msdn.com/b/psssql/archive/2009/11/03/the-sql-server-wait-type-repository.aspx> (<http://bit.ly/9P5qNW>)
 - `sys.dm_os_wait_stats`
 - <http://technet.microsoft.com/en-us/library/ms179984.aspx>

Review

- Introduction
- Thread lifecycle
- DMVs
- Common wait types

Questions?