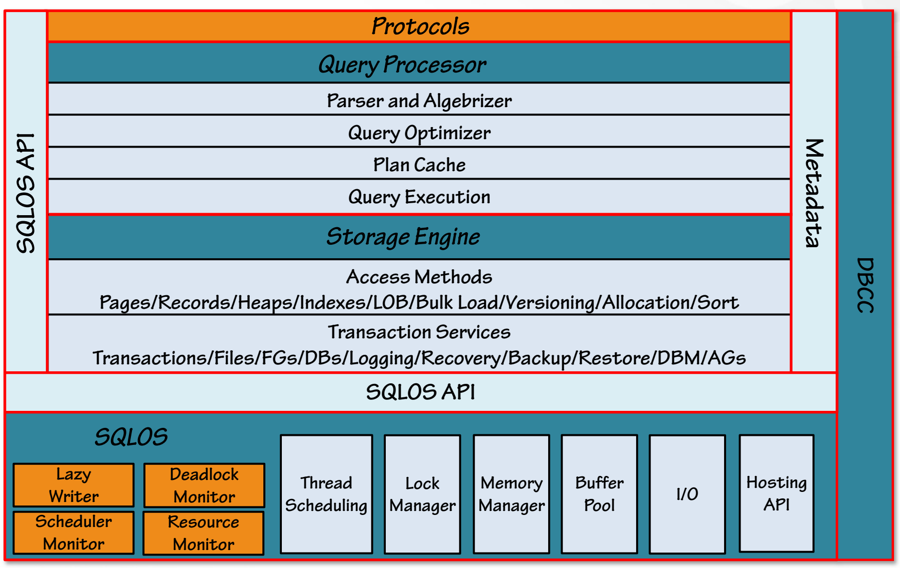
SQLskills Immersion Event

IEPTO2: Performance Tuning and Optimization

Module 3: Wait Statistics

Paul S. Randal
Paul@SQLskills.com

Server Architecture



Overview

- **How thread scheduling works in SQL Server**
- **Fundamentals of waits, latches, and spinlocks**
- **Investigating waits, latches, and spinlocks using DMVs**
- **Common scenarios, including:**
  - Data and log file I/O
  - Latch contention in tempdb and user tables
  - Parallelism
  - Quantum exhaustion

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 troubleshooting**
  - 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
  - E.g., Michael Swart's GitHub to trend/chart at https://sqlskills.com/p/122
- **Remember: not as valuable when SQL Server is running well**

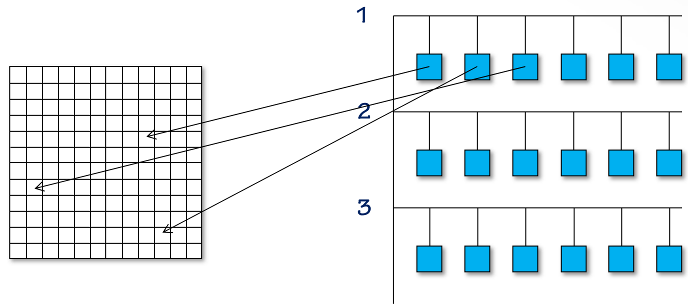
What are Waits?

- **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)

© SQLskills, All rights reserved.
https://www.SQLskills.com

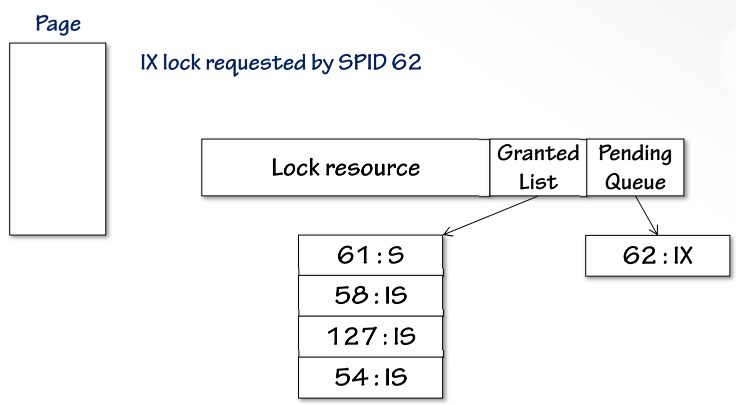
Page Read Example

- Hash list of BUF structures per database, ordered by page ID, for quick access and determining if a particular page is in memory or not



- If requested page not in memory, thread starts asynchronous physical read and has to wait for it to complete (PAGEIOLATCH_SH or _EX)

Locking Example



- If the lock isn't available, the thread enters itself on the lock's pending queue and has to wait (LCK_M_IX)

Why are Resources Unavailable?

- **Some other thread is holding the resource, or the 'resource' needs some process to occur (e.g. page read from disk)**
- **Examples:**
  - 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
- **Resource waits are investigated using DMVs, performance counters, and other tools**

9

Wait Statistics Analysis

- **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
- **Whitepaper**
  - SQL Server Performance Tuning Using Wait Statistics: A Beginners Guide
  - (Resource links at end of deck)
- **Comprehensive waits and latches library**
  - https://www.SQLskills.com/help/waits
- **Most commercial performance monitoring tools capture and show wait statistics**
  - Many free tools also do this, such as sp_whoisactive
- **Various releases of SQL Server have provided wait statistics views**

10

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**

How Many Threads?

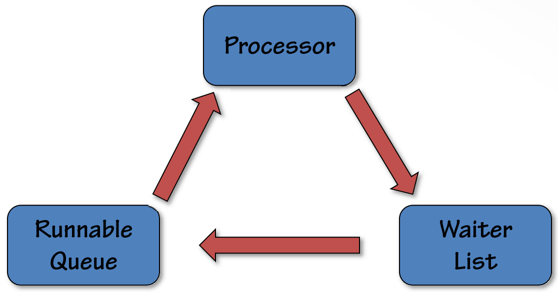
- **Startup calculation to determine the maximum number of threads for the thread pool, although a much smaller number created initially**

# logical cores	# threads
1 (or <2GB mem)	256 (only on 2017+)
<= 4	512
> 4 and <= 64	512 + ((cores – 4) * 16)
> 64	512 + ((cores – 4) * 32)

- **E.g. my laptop with 8 cores has a maximum of 576 threads**
- **Can be changed using 'max worker threads' sp_configure option**
- **The thread pool will dynamically grow and shrink as needed**

Components of a Scheduler

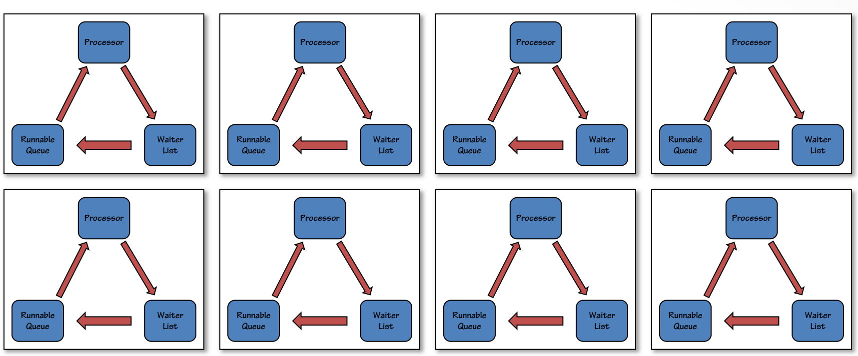
- All schedulers are composed of three 'parts'



- 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

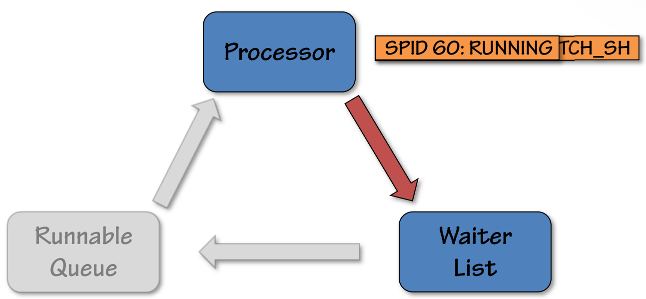


Thread States

- **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 a Waiter List waiting for a resource/operation
- **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 a Waiter List (on the scheduler or for a resource)
  - This process is called being 'suspended'

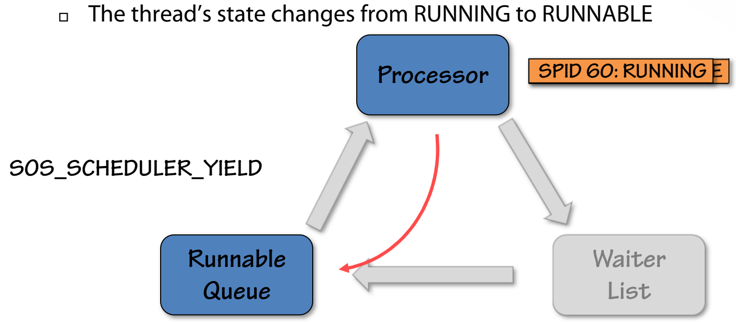


The Waiter List

- **The "Waiter List" is set of threads that are suspended on that scheduler**
  - I/O and timer-task waiter lists for the scheduler
  - Pending queues for other resources (e.g. locks, latches)
- **Any thread can be notified at any time that the resource it is waiting for is now available**
- **No limit to how long a thread remains on a waiter list**
  - Although execution timeouts or lock timeouts may take effect
- **No limit to how many of a scheduler's threads may be waiting**
- **The sys.dm_os_waiting_tasks DMV shows which threads are currently waiting and what they are waiting for**

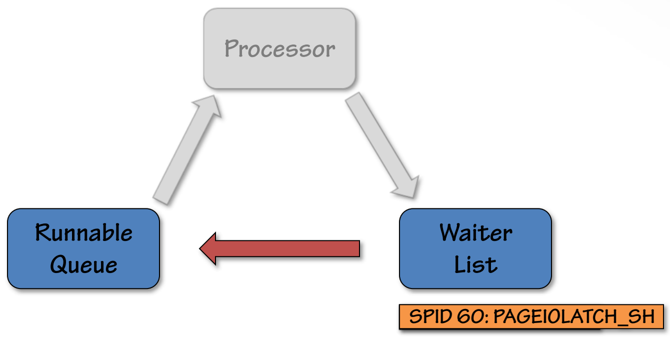
Special Case: Quantum Exhaustion

- **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

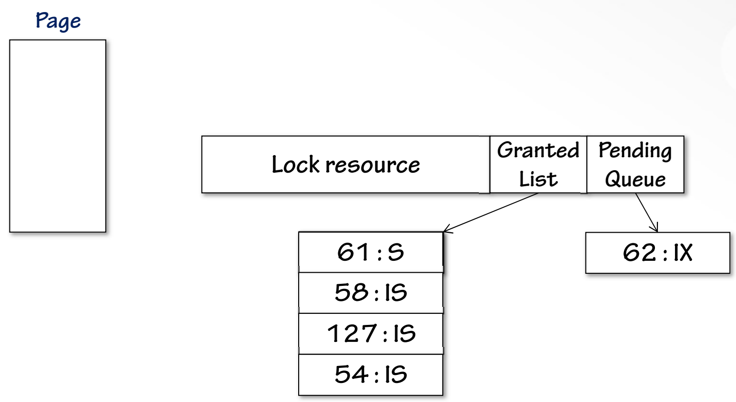
- **A thread continues to wait until it is told that the resource is available**
  - The thread's state changes from SUSPENDED to RUNNABLE
  - The thread moves to the Runnable Queue
  - This process is called being 'signaled'



Page Read Example

- **Thread is suspended waiting for the asynchronous physical read to complete, but how does it know when that is?**
  - It can't run any code to check!
- **When it issued the I/O, it essentially added itself to the scheduler's list of threads waiting for I/Os to complete**
- **Another thread on the scheduler checks the list and signals any threads whose I/Os have completed**

Locking Example



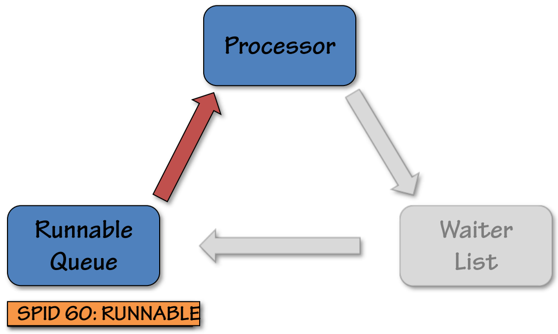
- **Thread dropping a lock checks whether any other locks can be granted, does that, and signals the waiting thread**

The Runnable Queue

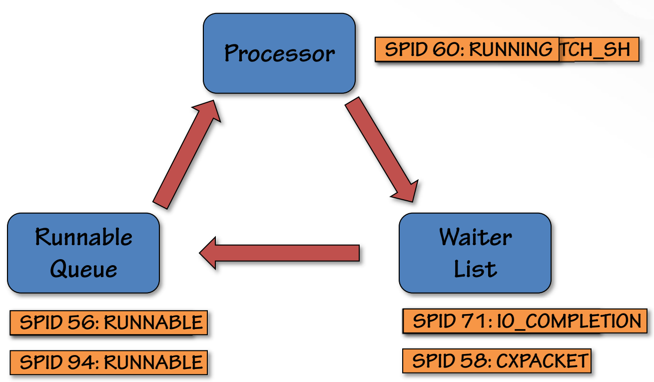
- **Scheduling code chooses which thread to execute next**
  - When the currently executing thread is suspended or exhausts its quantum
- **The Runnable Queue is mostly a First-In-First-Out (FIFO) queue**
  - Threads enter queue at bottom and progress to top
  - Special case to avoid unfair scheduling in 2016+
    - E.g. two threads where T1 can use entire 4ms, but T2 only 0.5ms
    - Pre-2016, T1 will get 8x the CPU time of T2
    - 2016+: T1 and T2 will get roughly equal CPU time
    - See blog post at https://sqlskills.com/p/077
    - This helps with things like the log writer background threads
  - Special case with Resource Governor High/Medium/Low priority workload groups, but rarely used
- **The size of the Runnable Queue can be seen from the runnable_tasks_count column in sys.dm_os_schedulers**

Transition: RUNNABLE to RUNNING

- **The thread waits on the Runnable Queue until it is chosen as the next thread when the processor becomes available**
  - The thread's state changes from RUNNABLE to RUNNING
  - 2019+: it might move to a different scheduler in the same NUMA node



Pulling It All Together



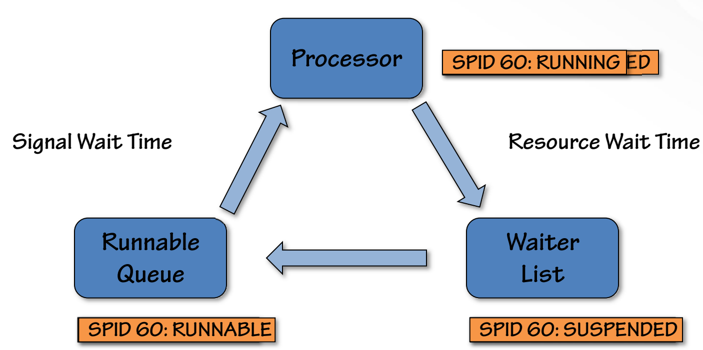
Wait Times Definition (1)

- **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 a 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**

SQLskills immerse yourself in sql server

25

Wait Times Definition (2)



Wait Time = Resource Wait Time + Signal Wait Time

SQLskills immerse yourself in sql server

26

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

27

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 and averages

28

Additional Sources of Wait Info

- **sys.dm_exec_session_wait_stats added in 2016**
  - Gives all waits for the entire session, not per batch, so be careful
  - When connection pooling, clears wait info when connection reset
- **Actual query execution plan contains wait info in 2016 SP1+**
  - All waits encountered by the execution
  - Look in Properties of left-most operator
    - Must be using 2016 SP1 or higher SSMS
  - Look in plan XML for the <waitstats> node
  - **Not accurate with parallel plan**
- **Query store captures wait statistics**
  - Aggregated into groups, not individual wait types
  - See sys.dm_db_query_store_wait_stats

SQLskills immerse yourself in sql server 29

Collecting Waits on Azure SQL Database

- **sys.dm_os_wait_stats gives stats for the container the Azure SQL Database is in, so do not use it**
- **Use sys.dm_db_wait_stats for waits for just the database**

- **Tim Radney explains how to get it working here:**
  - https://sqlskills.com/p/078

SQLskills immerse yourself in sql server 30

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

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:**
  - 100,000 waits for LCK_M_S over 8 hours
  - Is it a potential problem?
  - No, if over 8 hours total wait time for the LCK_M_S locks was only 50s altogether, each wait is only 0.5ms
  - Yes, if *each* LCK_M_S wait was for 50s

Demo

Simple example: waits DMVs and filtering

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

Methodology (1)

- 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
- Is the problem still happening?
- Examine any 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
- Examine the output from sys.dm_os_waiting_tasks/dm_os_wait_stats
  - What is happening on the server right now?
  - What has happened in the past?

Methodology (2)

- 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 info from relevant sources to pin-point problem
  - DMVs, query plans, performance counters, code analysis
  - Try one solution to see if that solves the problem
    - Repeat analysis, etc.

Using Extended Events

- **When a wait starts and ends, the sqlos.wait_info event fires**
  - Captures similar information to sys.dm_os_wait_stats
  - Also the sqlos.wait_completed event added in SQL Server 2014
    - **Note:** has same resource description as DMV from SQL Server 2016 – very useful!
- **For preemptive waits, 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

SQLskills immerse yourself in sql server

37

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**

SQLskills immerse yourself in sql server

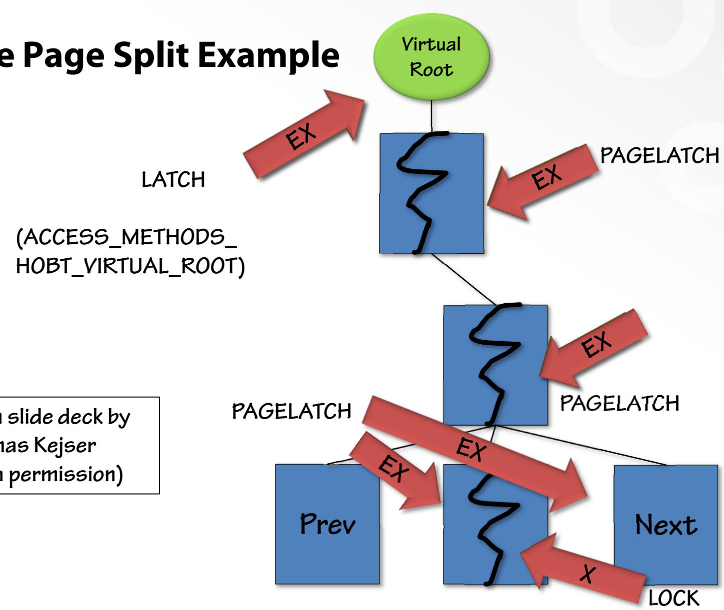
38

© SQLskills, All rights reserved.
https://www.SQLskills.com

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

B-tree Page Split Example



From slide deck by Thomas Kejser (with permission)

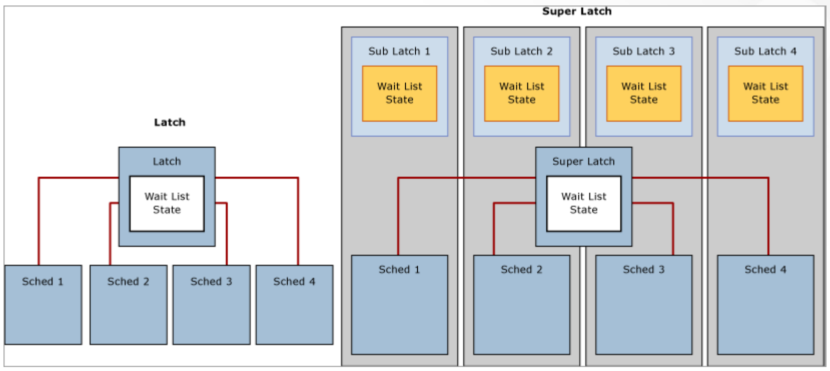
Latch Contention

- **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: https://sqlskills.com/p/079**

Superlatches

- **When the Engine detects lots of SH latch requests on a buffer in the buffer pool, it can promote the latch to a super latch**
- **The latch is partitioned so there is one sub-latch per scheduler, rather than one latch overall**
  - Reduces contention, as even SH latch access requires coordination
- **The superlatch will be demoted again if a series of EX latch requests are detected, once the page is dropped from the buffer pool**
  - As an EX request for a superlatch requires EX latching each superlatch
- **See Latches perfmon counters for total, promotions, demotions**
- **May see this error:**
  - *Message Warning: Failure to calculate super-latch promotion threshold.*
  - Benign message as thresholds are recalculated every 60s by the lazy writer
  - If seeing it lots, check OS power plan is set to High Performance

Superlatches



- Source: Latches whitepaper at https://sqlskills.com/p/079

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? https://sqlskills.com/p/080**

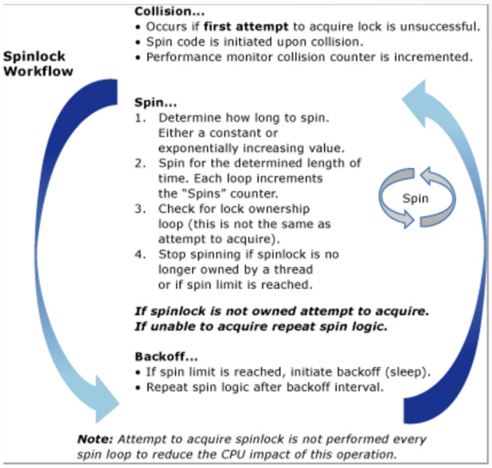
Using Extended Events

- **For very advanced troubleshooting there are events that allow tracking of latches**
  - sqlserver.latch_suspend_begin
  - sqlserver.latch_suspend_end
  - Similar to the sqlos.wait_info and sqlos.wait_info_external events but have a lot more information about the latch itself
- **Demo of this later in the module**

What are Spinlocks?

- **A spinlock is an even lighter-weight thread synchronization mechanism than a latch**
  - Used like a latch for data structure access control
- **Spinlocks are used when the data structure access will be for an extremely short time so the overhead of acquiring a latch is too much**
- **Examples of spinlocks:**
  - FGCB_PRP_FILL
  - BUF_FREE_LIST
- **Troubleshooting spinlocks usually requires very deep knowledge of SQL Server internals**
  - However, it is interesting and useful to know what spinlocks are
- **Great spinlocks post from Chris Adkin at https://sqlskills.com/p/082**

Spinlock Mechanism



- Source: Spinlocks whitepaper at https://sqlskills.com/p/081

Spinlock Mechanism

- **There is no waiting mechanism for spinlocks like there is for latches**
  - Once a thread starts acquiring a spinlock, it will remain on the processor until it has acquired and then dropped the spinlock
- **A thread tests the spinlock to see if it can be acquired**
- **If not, the thread sits in a loop checking whether it has the spinlock**
  - When the thread cannot acquire the spinlock, this is called a 'collision'
  - The thread then loops and tries again, this is called a 'spin'
  - Spins required after a collision do not count as more collisions
  - The number of collisions and the number of spins are tracked
- **After a certain number of spins, the thread stops trying for a bit**
  - This is called a 'backoff'
  - Simply calls the Windows sleep() function and stays on the processor
  - Can cause other threads to have high signal wait times
- **SQL Server tracks all of this**

SQLskills immerse yourself in sql server 49

Spinlock Contention

- **When a large number of threads are contending for access to a single spinlock, this can lead to performance problems**
- **All these symptoms must be present for high CPU usage to potentially be from spinlock contention:**
  - High and increasing spins and backoffs for a spinlock (billions or more)
  - High CPU usage with many connections to the server, OLTP workload
  - CPU usage, spins, and backoffs increasing much faster than the workload is increasing (possibly an exponential divergence)
- **However, it is likely to NOT be spinlock contention so investigate other waits and latches first**
  - Common for some spinlocks to have very high spins with an OLTP workload
- **Troubleshooting spinlock contention is very advanced**
- **Whitepaper on investigating spinlock contention https://sqlskills.com/p/081**

SQLskills immerse yourself in sql server 50

Some Common Spinlocks (1)

- OPT_IDX_STATS
  - Updating counters for sys.dm_db_index_usage_stats / missing_index_stats
  - This could be from many concurrent updates to table with lots of indexes
- LOCK_HASH
  - Lock Manager looking in the list of hash buckets for lock hash collisions
  - Consider smaller transactions, using NOLOCK, turning off page locks
- LOGFLUSH_ACCESS and LOGFLUSHQ
  - Involved with writing log buffers to disk – see WRITELOG wait
  - Contention could be from very heavy load of very small transactions
- XVB_LIST
  - Involved with versioning system
  - Contention investigation blog post at https://sqlskills.com/p/124
  - SQL2019 bug fixed in CU9 – see https://sqlskills.com/p/123

Some Common Spinlocks (2)

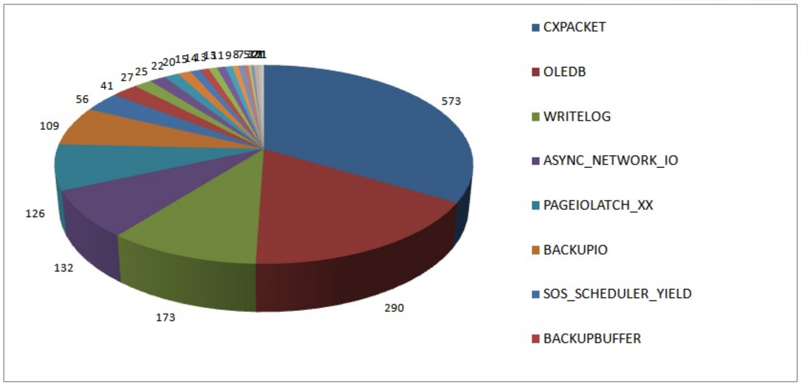
- DP_LIST
  - Used to control access to the dirty page list for indirect checkpoints
    - Allows much faster checkpoint mechanism
  - Indirect checkpoint in very busy tempdb – see https://sqlskills.com/p/041
  - Fixed in latest builds of 2016 and 2017
- SOS_CACHESTORE
  - Bug: when plan cache reaches maximum size – https://sqlskills.com/p/109
    - Enable trace flag 174 to increase cache size
  - Also can be a problem with excessive use of temp tables
    - Contention for adding/removing from the temp table cache
    - Alleviated in SQL Server 2019

Transaction Log Example

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

Top Wait Types

- **Survey results from 1700+ SQL Server instances across Internet**



- **Source: my blog at https://sqlskills.com/p/083**

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 is the problem
- **Further analysis:**
  - Determine which tables/indexes are being read
    - Take the page ID and follow steps in this post: https://sqlskills.com/p/084
  - 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**
- **Possibly In-Memory OLTP in SQL Server 2014+**

- **A quick band-aid could be to add more memory regardless to increase the buffer pool size**
    - Cheap to do, provides temporary relief, maybe less risky than immediate code change

Data Reading

- **Reads can be:**
  - Single/multiple pages from a data file
  - Single/multiple extents from a data file
  - Variable size chunks of FILESTREAM files
  - Usually random, except for large scans and backups
- **Misconception that SQL Server always reads extents**
  - But it will do sometimes to 'ramp up' the buffer pool
- **Read performance can be dramatically affected by:**
  - Number of files and file placement
  - I/O subsystem configuration
  - Buffer pool memory and memory pressure
  - Query plan choice
  - Ability to perform efficient read-ahead on indexes

SQLskills immerse yourself in sql server

57

Buffer Pool Usage

- **Unfortunately the query optimizer knows nothing about the contents of the buffer pool otherwise it might choose a less optimal index that's already in memory (to save physical reads)**
- **sys.dm_os_buffer_descriptors**
  - Lists all pages currently in memory
  - Allows aggregating by database, table
  - Allows view of aggregate empty space in pages in memory
  - Can look at how memory pressure affects need to perform physical vs. logical I/Os

SQLskills immerse yourself in sql server

58

Demo

Buffer pool usage

Data Writing

- **Data file writes can be:**
  - Single/multiple pages
  - Single/multiple extents (for bulk operations)
  - Up to 32 contiguous pages before 2016, up to 128 in 2016+
- **Data file pages are written when:**
  - A checkpoint occurs (for whatever reason)
  - The lazy writer forces a dirty page from the buffer pool
  - A bulk operation flush occurs (a.k.a. 'eager writes')
  - A database mirror is processing log records
    - Dirty pages are continuously flushed out, leading to heavy I/O load
      - Use trace flag 3499 on the mirror to disable this
    - Does not happen for Availability Groups
- **Write performance can be dramatically affected by:**
  - Number of files and file placement
  - I/O sub-system configuration

Read/Write Latency

- **Many systems these days are I/O bound, but is the problem the I/O subsystem or your queries?**
- **If you've optimized your queries and performance is still slow, look into the I/O subsystem**
- **sys.dm_io_virtual_file_stats**
  - Gives total stall time (aggregated latencies) for reads and writes along with read/write counts
  - Explanation: https://sqlskills.com/p/085
  - Snapshot over time: https://sqlskills.com/p/086
  - Better than Physical Disk performance counters as these DMVs are per database file and give SQL Server's view of the I/O subsystem
- **sys.dm_io_pending_io_requests**
  - Lists all pending I/Os, and usually joined with sys.dm_io_virtual_file_stats
- **To use on Azure SQL Database, see https://sqlskills.com/p/078**

Demo

I/O latencies

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_UP Wait Explanation

- **Some query workloads cause multiple concurrent threads to repeatedly create/drop small temp tables and/or worktables**
  - Can also be from repeated population/truncation of temp tables
- **Easy to cause PAGELATCH_UP contention on allocation bitmaps prior to SQL Server 2019, especially PFS**
  - Use sys.dm_os_waiting_tasks to see waits on PAGELATCH_UP
  - SGAM page to manipulate mixed extents (resource 2:1:3)
  - PFS page to allocate/deallocate pages (resource 2:1:1 and then any page ID that's a multiple of 8088)
- **Contention can occur in all versions, but much reduced in 2019+**
- **This can sometimes (rarely) happen in user databases with VERY high-end allocation workloads**

PAGELATCH_UP Wait Solutions

- **TF 1118 (KB 328551) removes mixed extents (SGAM contention)**
  - **All instances across the world should have this enabled before 2016**
  - **Behavior on by default in SQL Server 2016+**
- **Use multiple data files to reduce contention (KB 2154845)**
  - <= 8 cores: #files = #cores; > 8 cores, #files=8, then increase by 4 at a time
  - 2016+ install automatically configures multiple tempdb data files
  - Adding just one file may not work (see https://sqlskills.com/p/029)
  - Investigation article on Simple Talk: https://sqlskills.com/p/030
- **Alleviated a bit in latest 2016/2017 builds by spreading allocations over multiple PFS intervals (see https://sqlskills.com/p/106 and 108)**
- **2019 enhancements**
  - No latch for PFS updates, using special CPU instructions
  - Temp table cache optimizations to reduce spinlock contention when adding/removing entries

SQLskills immerse yourself in sql server

65

Demo

tempdb allocation bitmap contention

Temp Table Misuse

- **Very common for us to see temp table problems**
- **Lack of filtering when populating a temp table**
  - Bad: pulling columns into a temp table that are not used
  - Large waste of space and CPU
  - Minimize the column list in a temp table
- **Incorrect temp table indexing**
  - Bad: creating indexes before populating the table
  - Bad: creating indexes that are not used
- **Temp table when none is required**
  - Forcing an intermediate result set into a temp table could disrupt the efficient data pipeline through a query
  - Query may run much faster without a temp table
- **Great post on temp table usage: https://sqlskills.com/p/089**

Tempdb Space Tracking

- **On several client systems I have an automated tempdb space tracking system (that I'll show you)**
- **sys.dm_db_file_space_usage**
  - How is space usage broken out per use (internal/user/version store)
    - Tempdb only before SQL Server 2012
  - Internal is allocations done automatically by SQL Server
    - Work tables for cursor or spool operations and temporary large object (LOB) storage, work files for operations such as a hash join, sort runs
- **sys.dm_db_task_space_usage**
  - Tracks cumulative page allocation and deallocation counts for each thread
  - Can see parallelism happening

Tempdb Log Space Investigation

- **Tempdb log grows out of control – what's going on?**
- **Use my script based on sys.dm_tran_database_transactions**
- **Easy to get confused:**
  - Lots of tempdb log growth, but…
  - …only active transaction(s) have small amounts of space usage
- **Where did the space usage come from?**
  - Lots of already-committed smaller transactions
  - One long-running transaction that doesn't do much still makes the log grow by preventing log clearing/truncation
- **Solution**
  - Don't allow long-running tempdb transactions

69

Demo

Tempdb space tracking

PAGELATCH_EX Wait Explanations and Solutions

- **Tempdb system table contention – see next slide**
- **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 an index with ever-increasing key and row size such that multiple rows fit on a page**
  - Spread the insertion points in the index using a random or composite key, plus provision a FILLFACTOR to prevent page splits
  - Shard into multiple partitions/tables/databases/servers
  - Increase row size so only one row fits per page
  - Evaluate in-memory tables as a staging area
  - Try OPTIMIZE_FOR_SEQUENTIAL_KEY in 2019+, but might make it worse!
    - Limits number of threads, try when # connections > # schedulers
    - Will cause BTREE_INSERT_FLOW_CONTROL waits to appear

tempdb System Table Contention

- **Some query workloads cause multiple concurrent threads to repeatedly create/drop small temp tables and/or worktables**
  - Can cause PAGELATCH_SH/EX contention on sysobjvalues and sysseobjvalues tables ('insert hotspot')
  - Use sys.dm_os_waiting_tasks to see waits on PAGELATCH_SH/EX in tempdb
  - Check whether the page is in a system table using DBCC PAGE
- **Fixed somewhat in SQL Server 2016 builds**
  - See https://sqlskills.com/p/107 and /108
- **Can remove completely in 2019+ by setting system tables in-memory**
  - ALTER SERVER CONFIGURATION SET MEMORY_OPTIMIZED TEMPDB_METADATA = ON;
  - Restart instance
- **Also a new trace flag 3427 in latest 2016 builds that speeds up small transactions using tempdb (see https://sqlskills.com/p/108)**
  - Removes overhead from Common Criteria Compliance auditing

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 (https://sqlskills.com/p/090)
  - Are there any LCK_M_RS_XX locks? If so serializable isolation level was used

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
- **Serializable isolation level being used erroneously**
  - Distributed transactions, .Net TransactionScope default

Demo

Insert hotspots

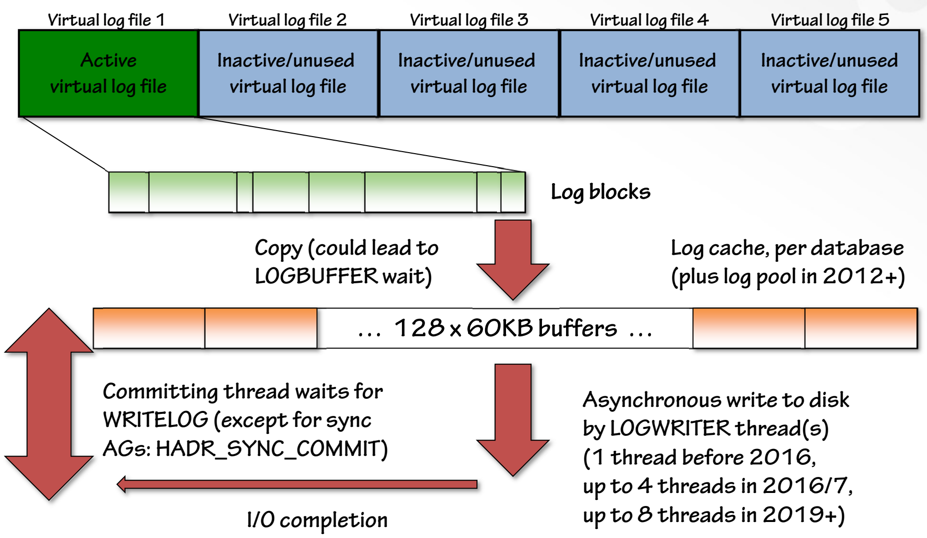
WRITELOG Wait

- **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 is overloaded (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 111/112 (31/32 prior to 2012) then the internal limit has been reached for outstanding transaction log writes for a single database
  - Look at average size and volume of transactions
  - Are all the log files for all databases on the same volume?

Transaction Log Writes

- **Writes are always sequential**
- **No performance gain from having multiple log files (except for NVDIMM case)**
  - SQL Server ALWAYS perform sequential writes of log records
- **There are specific limits on transaction log writes**
- **Limit on number of in-flight log writes**
  - 2012+: 112 outstanding writes
  - Older versions: 32 outstanding writes
- **Limit on total size of log writes of 3,840KB at any given time**
  - 32 in-flight writes of up to 60KB each plus 32 log blocks waiting to be written

Transaction Log Flushes



WRITELOG Wait Solutions

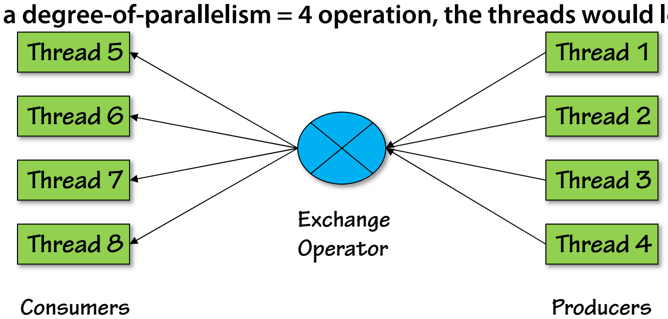
- **Move the log to a faster I/O subsystem**
  - E.g., NVDIMM for log tail in 2016+ - https://sqlskills.com/p/038
- **Spread log files from multiple databases over multiple volumes**
- **Increase size of transactions to reduce small log block flushes to disk**
- **Implement delayed durability in SQL Server 2014+**
- **Check for incorrect CACHE size on SEQUENCE objects**
- **Remove unused nonclustered indexes**
  - Reduce logging overhead from maintaining them during DML operations
- **Change index keys or introduce fillfactors to reduce page splits**
- **Are synchronous database mirroring/AGs causing delays?**
- **Potentially split the workload over multiple databases or servers**
- **Potentially isolate log writer threads – see https://sqlskills.com/p/091**
  - Remove their CPUs from the CPU affinity mask so no user threads there

Demo

Slow transaction log

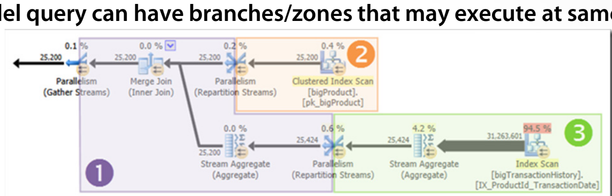
Parallel Threads Example

- As part of a query plan, you may see the operator, for example
  - This is a Repartition Streams operation
  - Producer threads fill packets of rows, fed into the exchange
  - Consumer threads read rows from the exchange output
  - No link/tie/mapping between producer and consumer threads
- For a degree-of-parallelism = 4 operation, the threads would look like:



Threads in a Parallel Query

- Always a single thread, thread ID 0, the parent, exists at the start of the query
  - Creates all other threads, based on DOP the query is able to run at, and runs query
- A parallel query can have branches/zones that may execute at same time

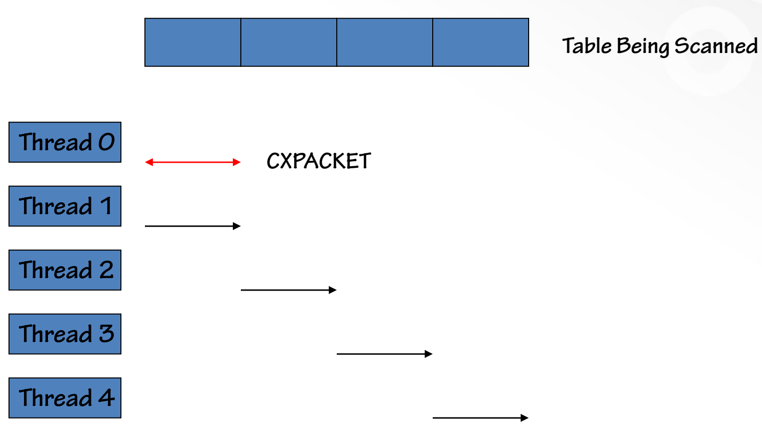


- Each zone can use up to degree-of-parallelism threads, reserved at start
  - MAXDOP limits the number of threads per parallel zone
  - Each thread on a different scheduler, so MAXDOP also limits number of schedulers
- Total possible threads:
  - Per query: (DOP x concurrent zones) + parent (which may be on different scheduler)
  - Per operator: DOP x 2
- More details: https://sqlskills.com/p/092 /093 /118 /119 /125

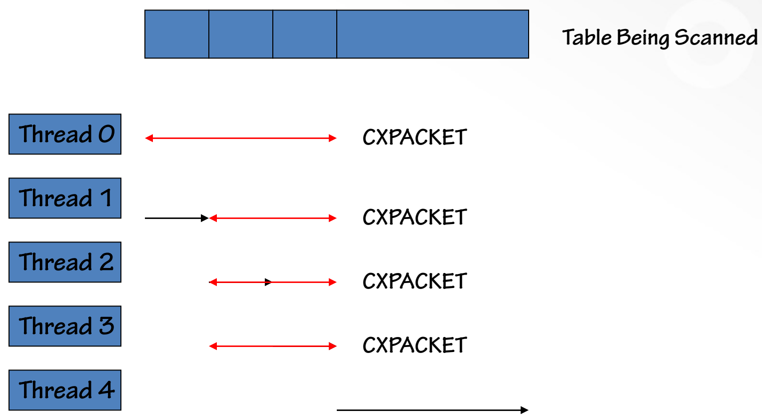
CXPACKET Wait Explanation

- **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
  - Examine query plans of requests that are accruing CXPACKET waits to see if the query plans make sense for the query being performed
  - Are there non-zero ID threads showing CXPACKET wait?
  - If seeing many CXCONSUMER waits in sys.dm_os_waiting_tasks:
    - Long CXCONSUMER waits may indicate skewed parallelism
    - Many short CXCONSUMER waits may indicate a problem on the producer side like a poor join condition making producer threads not push rows through

CXPACKET Wait Example (1)



CXPACKET Wait Example (2)

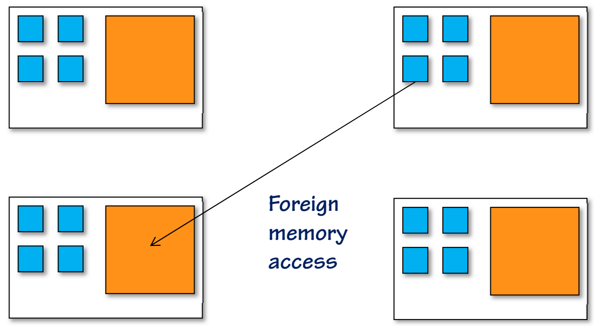


CXPACKET Wait Solutions

- **Possible root-causes:**
  - Just parallelism occurring
  - Table scans because of missing nonclustered indexes or incorrect query plan
  - Out-of-date statistics or cardinality issue causing skewed work distribution
- **If there is actually a problem:**
  - Make sure statistics are up-to-date and appropriate indexes exist
  - MAXDOP for a query? Or just a database (in 2016+)? Or Resource Governor?
  - MAXDOP for the instance? Test to figure out best value for *you*:
    - No NUMA then = # cores, up to max of 8
    - NUMA = # logical cores per NUMA node, up to 16 (2016+) or 8 (< 2016)
    - General guidance, soft-NUMA complicates this
  - Set 'cost threshold for parallelism' higher to avoid some parallel plans
    - Jon's blog post at https://sqlskills.com/p/094 provides a guestimate
    - Always set and test, don't just set to some blogger's value and walk away

NUMA

- Example with four NUMA nodes with four physical cores in each
- Buffer pool is split into four partitions, one per NUMA node using the node-local memory
- Foreign memory accesses are very expensive



87

Demo

Parallelism

SOS_SCHEDULER_YIELD Wait

- **What does it mean:**
  - A thread exhausted its 4 millisecond quantum and voluntarily yielded
- **Avoid knee-jerk response:**
  - Do not assume that CPU pressure is the problem
    - High signal wait times show CPU pressure
  - Do not assume that spinlock contention is the problem
- **Further analysis:**
  - Examine query plans to see whether scans are occurring
    - Check if 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

SQLskills immerse yourself in sql server

89

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
  - Virtual machine delays causing spurious SOS_SCHEDULER_YIELDs
- **Solutions:**
  - On slower processors, potentially enable hyper-threading to give more schedulers and more potential for concurrent work, especially for OLTP workloads
  - Make sure query plans are correct for query being executed
  - Fix any VM issues

SQLskills immerse yourself in sql server

90

Using Extended Events to Examine Call Stacks

- **The only way to see exactly why SOS_SCHEDULER_YIELD waits are occurring is to examine SQL Server code call stacks**
- **Download the correct symbols**
  - See my blog post at https://sqlskills.com/p/095
  - Or use the Callstack Resolver tool https://sqlskills.com/p/120
- **Enable trace flag 3656 to allow symbol resolution (and 2592 on 2019)**
  - Also disable error log printing about dbghelp.dll version
- **Create an Extended Event session that:**
  - Captures sqlos.wait_info events for wait_type = 120 (or 124 for 2012+)
  - Captures the package0.callstack action
  - Uses the package0.histogram target
- **Run the workload and examine the captured call stacks**
- **Very advanced!**
  - See my blog post for a walk-through example (https://sqlskills.com/p/096)

Demo

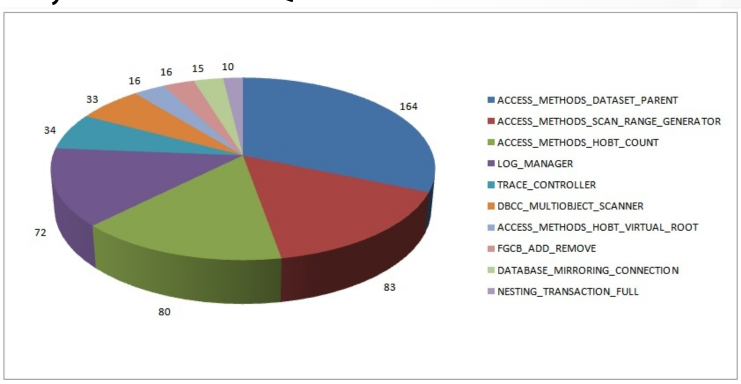
SOS_SCHEDULER_YIELD waits

LATCH_XX Wait

- **A non-page latch is the point of contention**
- **Further analysis:**
  - Use sys.dm_os_latch_stats to investigate which latch(s) are experiencing high wait times
  - Correlate with other prevalent wait statistics
    - For example, CXPACKET waits with LATCH_EX waits where the prevalent latch class is ACCESS_METHODS_SCAN_RANGE_GENERATOR
- **Possible root-causes and solutions:**
- **Depend on the latch class**
  - These are not documented so look in my waits/latches library
    - https://www.sqlskills.com/help/latches/

Top Latch Classes

- **Survey results from 581 SQL Server instances across Internet**



- **Source: my blog at https://sqlskills.com/p/097**

FGCB_ADD_REMOVE Latch

- **Access to the File Group Control Block (FGCB) when adding, removing, shrinking, or growing files in the filegroup**
- **Further analysis:**
  - Analyze the auto-growth settings of the file in all filegroups
  - Extended Events must be used to determine which database is involved
    - Use the latch_suspend_begin and latch_suspend_end events with latch class 48, and correlate to the database_data_file_size_change event using causality tracking
- **Possible root-causes:**
  - Auto-growth settings for a file are very low, requiring frequent growth, coupled with heavy, concurrent use of the filegroup

SQLskills immerse yourself in sql server

95

Demo

Using Extended Events to show effect of poor auto-growth settings

DBCC_XX Latches

- **Examples:**
  - DBCC_MULTIOBJECTSCANNER (the most common to see)
  - DBCC_CHECK_AGGREGATE
  - DBCC_OBJECT_METADATA
- **Do not stop running consistency checks**
- **Further analysis: none necessary as these are all DBCC CHECKDB**
- **Possible root-causes:**
  - DBCC_MULTIOBJECTSCANNER latch was identified as a contention point and fixed in 2012 and under a trace flag in SQL Server 2008 R2
    - See Bob Ward's post (https://sqlskills.com/p/098) and KB article 2634571
  - DBCC_OBJECT_METADATA is a bottleneck with computed column indexes
    - See my post at https://sqlskills.com/p/0099

PREEMPTIVE_XX_YY Waits

- **What does it mean:**
  - Usually a thread has called out to the OS
    - Sometimes a thread staying in SQL Server but doesn't want to give up CPU
  - Threads must switch to preemptive mode when doing so
  - Note that the thread status will be RUNNING instead of SUSPENDED
- **Further analysis:**
  - ~200 PREEMPTIVE waits
  - These waits are very minimally and poorly documented
  - To determine what the thread is doing, look in my waits library
    - https://www.sqlskills.com/help/waits/#p
- **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_WRITEFILEGATHER Wait

- **What does it mean:**
  - A thread is calling out to Windows to write to a file
- **Avoid knee-jerk response that I/O subsystem has a problem**
- **Further analysis:**
  - What database operations are under way? E.g. restore or file growth
- **Possible root-causes and solutions:**
  - Zeroing a large transaction log file during a restore or log file growth
  - Zeroing a large data file during restore or data file growth
    - Enable instant file initialization and set manage growth appropriately
    - Do not delete existing database files before performing a restore
  - Described in KB article 2091024 (https://sqlskills.com/p/100)

PREEMPTIVE_OS_WAITFORSINGLEOBJECT Wait

- **What does it mean:**
  - Thread calling Windows to wait on state change of synchronization object
  - Commonly seen with ASYNC_NETWORK_IO wait
- **Further analysis:**
  - Follow instructions/solutions as for ASYNC_NETWORK_IO
  - Check whether transactional replication is running
- **Possible root-causes and solutions:**
  - As for ASYNC_NETWORK_IO (see next slide)
  - Could also be transactional replication Agent jobs (such as the Log Reader and Distribution Agent jobs)
    - See Joe Sack's blog post for more details (https://sqlskills.com/p/101)

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
- **Further analysis:**
  - Analyze client application code and client app server
  - 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
  - Could be from using MARS with large result sets or BCP inbound
  - Otherwise look for network hardware issues, incorrect duplex settings, or TCP chimney offload problems (see https://sqlskills.com/p/102)

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 (especially if they're very short waits)
  - Poor performance of a linked server

Miscellaneous Other Common Wait Types

- THREADPOOL
  - Waiting for a worker thread to become available, for example on a heavily-loaded system with a lot of parallel queries running
- RESOURCE_SEMAPHORE
  - Waiting for a query execution memory grant, for example for a sort
  - Usually indicates concurrent, memory-hungry queries
- MSQL_XP
  - Waiting for an extended stored procedure call to complete

Demo

Other wait types

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**
- **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**

Key Takeaways

- Wait statistics are a key part of performance problem diagnosis
- Don't knee-jerk, follow the easy methodology
- Practice gathering and analyzing wait statistics using the DMVs
- Remember that waits always happen
- Don't get into latches and spinlocks unless absolutely necessary
  - Too many potential red herrings
- Know what the common waits mean and don't mean

Resources

- Waits/latches repository
  - https://www.SQLskills.com/helps/waits
- Whitepapers:
  - SQL Server Performance Tuning Using Wait Statistics: A Beginners Guide
    - https://sqlskills.com/p/103
  - Diagnosing and Resolving Latch Contention on SQL Server
    - https://sqlskills.com/p/079
  - Diagnosing and Resolving Spinlock Contention on SQL Server
    - https://sqlskills.com/p/081
- Blog post categories
  - https://www.sqlskills.com/blogs/paul/category/wait-stats/ and /latches/ and /spinlocks/
- Pluralsight: SQL Server: Performance Tuning Using Wait Statistics

Review

- **How thread scheduling works in SQL Server**
- **Fundamentals of waits, latches, and spinlocks**
- **Investigating waits, latches, and spinlocks using DMVs**
- **Common scenarios, including:**
  - Data and log file I/O
  - Latch contention in tempdb and user tables
  - Parallelism
  - Quantum exhaustion

Questions?