

SQLskills Immersion Event

IE0: Accidental/Junior DBA

Module 7: Troubleshooting and Monitoring

Jonathan Kehayias
Jonathan@SQLskills.com



Troubleshooting Basics

- **Don't Assume Symptom=Root Cause**
 - Troubleshooting is not an exact science, and the same symptoms can result from many root causes
- **Don't do 'knee-jerk' troubleshooting**
 - Work through the data to see what may be the root cause
- **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



Diagnostic Data Collection

- Data collection should include multiple sources to facilitate root cause identification and validation from multiple sources
- **Good starting points:**
 - Wait statistics and file statistics inside SQL Server
 - Performance Monitor collection of performance counters for hardware, the Windows OS and SQL Server
- **More advanced:**
 - Buffer and plan cache usage
 - Index statistics (query optimization and Storage Engine)
 - Trace Data
 - CPU, reads, writes, duration
 - Extended Events now more viable for SQL Server 2012

Overview

- Waits and Queues methodology
- Collecting and analyzing Perfmon data
- Automating baseline data collections
- Blocking
- Deadlocks

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

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

Interpreting the Data

- 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

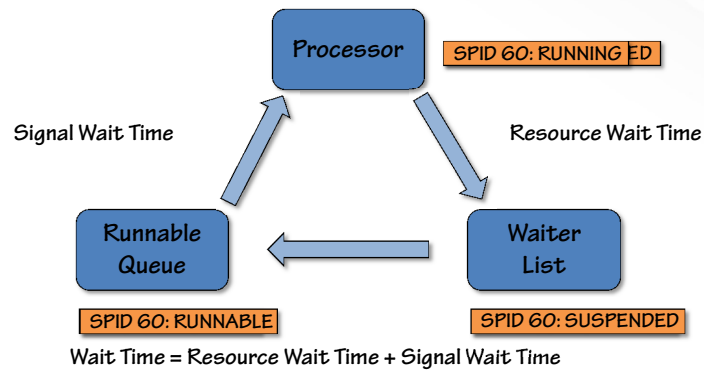
Thread States

- 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

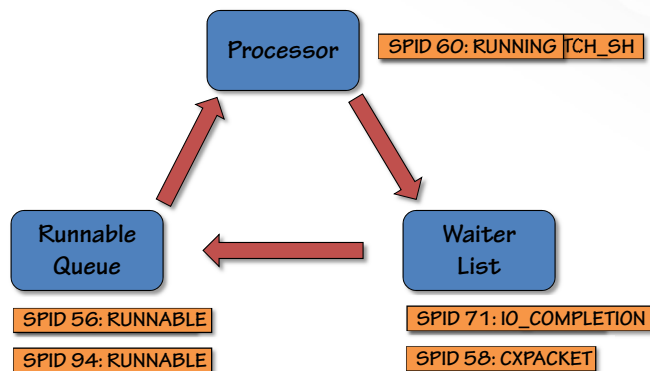
Wait Times Definition (1)

- **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)



Execution Lifecycle Multiple Threads



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

What's Relevant?

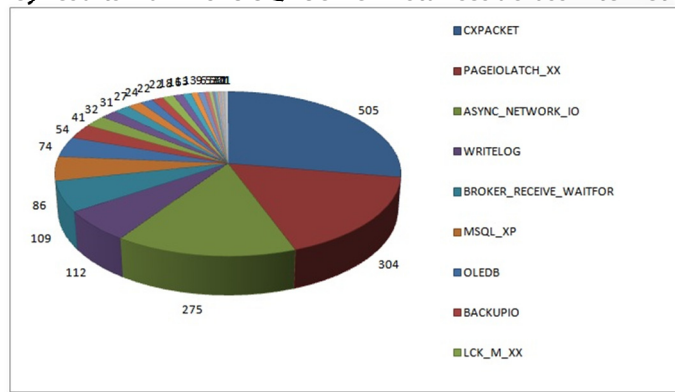
- **Just because there are waits, does not mean they are the problem**
 - Look for actionable items and filter out things like background tasks
 - Look at the demo code to see what I mean
- **Need to identify the top, relevant waits and then drill in**
- **Example:**
 - 1000 waits for LCK_M_S
 - Is it 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**

Demo

Querying DMVs

Top Wait Types

- Survey results from 1823 SQL Server instances across Internet

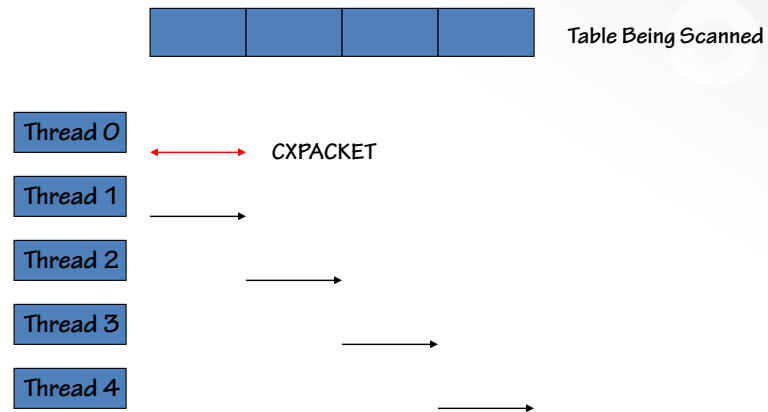


- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/Wait-statistics-or-please-tell-me-where-it-hurts.aspx> (<http://bit.ly/fSWeO5>)

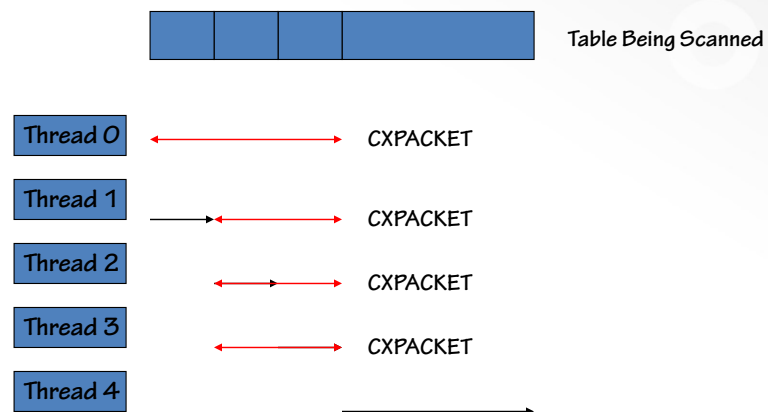
CXPACKET Wait Explanation

- **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
 - 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

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 (reading) and EX (modifying)
- **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 (reading) and EX (modifying)
- **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

Demo

Page latches

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>)

Demo

RBAR processing of results

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

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

Locks and blocking

Overview

- Waits and Queues methodology
- Collecting and analyzing Perfmon data
- Automating baseline data collections
- Blocking
- Deadlocks

Performance Monitor

- **Performance Monitor is built in to Windows**
 - Very easy to use and configure
- **Hardware, OS and SQL Server counters can be captured**
 - Excellent resource for capturing information about resources external to SQL Server that have a significant role in performance
- **It can be used to monitor performance real-time, or capture metrics over a period of time**
- **Data collection can be automated**
- **Data can be processed manually or automatically**
- **Overhead is minimal as long as sampling interval is greater than every 1 second**
 - Hard disk usage and performance based on number of counters, interval, and underlying storage

Reference: OS Counters to Collect

- | | |
|---|--|
| <ul style="list-style-type: none">▪ Processor<ul style="list-style-type: none">▫ % Processor Time▫ % Privileged Time▪ System<ul style="list-style-type: none">▫ Processor Queue Length▪ Memory<ul style="list-style-type: none">▫ Available Mbytes▫ Pages/sec▪ Paging File<ul style="list-style-type: none">▫ %Usage | <ul style="list-style-type: none">▪ PhysicalDisk<ul style="list-style-type: none">▫ Avg. Disk sec/Read▫ Avg. Disk sec/Write▫ Disk Reads/sec▫ Disk Writes/sec▪ Process (sqlservr.exe)<ul style="list-style-type: none">▫ % Processor Time▫ % Privileged Time |
|---|--|

Reference: SQL Counters to Collect

- **SQL Server:Access Methods**
 - Forwarded Records/sec
 - Full Scans/sec
 - Index Searches/sec
- **SQL Server:Buffer Manager**
 - Buffer cache hit ratio?
 - Free List Stalls/sec
 - Lazy Writes/sec
 - Page Life Expectancy?
 - Page Reads/sec
 - Page Writes/sec
- **SQL Server:General Statistics**
 - User Connections
- **SQL Server:Locks**
 - Lock Waits/sec
 - Number of Deadlocks/sec
- **SQL Server:Memory Manager**
 - Total Server Memory (KB)
 - Target Server Memory (KB)
- **SQL Server:SQL Statistics**
 - Batch Requests/sec
 - SQL Compilations/sec
 - SQL Re-Compilations/sec
- **SQL Server:SQL Statistics**
 - Batch Requests/sec
 - SQL Compilations/sec
 - SQL Re-Compilations/sec
- **SQL Server:Latches**
 - Latch Waits/sec

What does all this actually tell us?

Performance Analysis of Logs Tool

- Free tool on Codeplex designed to analyze Performance Monitor counter logs using industry standard thresholds
- Provides a built in template for SQL Server created by David Pless, a Premier Field Engineer at Microsoft
- Template can be used to create a Data Collector Set template that can be imported into PerfMon
- Details of the individual performance counters, what they mean, how they relate to each other, and the thresholds being tested are available in the user interface
- The template can be customized to add additional counters or change thresholds if necessary

Demo

Data Collector Sets and Performance Analysis of Logs

Other Free Tools

- **SQLDiag and SQLNexus automate collection and analysis of:**
 - PerfMon logs
 - SQL Trace files
 - Windows event logs
 - SQL Server error logs
 - Msinfo32 information
- **Clear Trace/Read Trace/Query Analyzer**
 - Automate and simplify analysis of SQL Trace data captures by normalizing and aggregating ad-hoc statements to identify the worst performing queries trending over time
- **Insider demo videos exist for each of these free tools**

Overview

- Waits and Queues methodology
- Collecting and analyzing Perfmon data
- **Automating baseline data collections**
- Blocking
- Deadlocks

Capturing a Baseline

Do You Know What “Normal” Looks Like?

- Provide a starting point for comparison of additional benchmarks over time
- Past baseline data is invaluable during a performance “crisis” and is also helpful in capacity planning
- Measure the impact of changes
 - Code and design changes
 - Increased workload
 - Hardware configuration changes
 - Upgrades to the OS or SQL Server
 - Test the changes on another server before going to production and measure the impact

Purpose of Benchmarking

What Are the Capabilities?

- Measure performance using a specific set of indicators to determine the performance level in a manner that can be compared to other systems, business requirements or previous benchmarks
- Determine the capacity limits of a system
 - Maximum number of concurrent connections
 - Maximum number of transactions/batches per second
 - Be able to forecast replacement/upgrade requirements before exceeding limits

Data Collection Examples

- Performance Monitor
 - Single collection of performance counters for hardware, the Windows OS and SQL Server
 - Easy to use for trending over time
- DMV output
 - Wait statistics
 - File statistics
 - Buffer and plan cache usage
 - Index statistics (query optimization and Storage Engine)
- Trace Data
 - CPU, reads, writes, duration
 - Extended Events now more viable for SQL Server 2012
- Notice that these are the same as what we used for troubleshooting earlier

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

Demo

Setting up baseline data collection

Overview

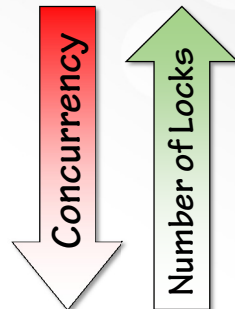
- Waits and Queues methodology
- Collecting and analyzing Perfmon data
- Automating baseline data collections
- **Blocking**
- Deadlocks

Terminology

- Transaction – a unit of work performed within the database
- Lock - the synchronization mechanism on a resource that protects changes amongst multiple concurrent transactions
- Lock mode - defines the level of access that other transactions have while the resource is locked

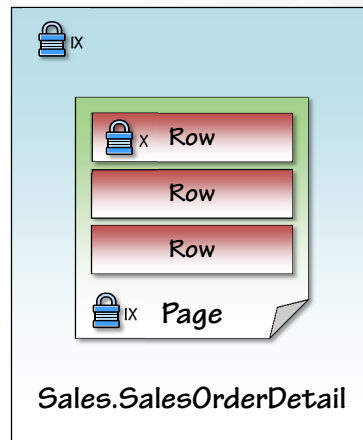
Lock Granularity

- RID/KEY – a single row is locked
 - RID – row identifier for a single row in a heap
 - KEY – index key for a single row in a index
- PAGE - a single page in the database is locked
- HoBT – a heap or B-tree (index) partition is locked
- TABLE – the entire table is locked
- METADATA - the table schema definition is locked
- Locks are acquired at multiple levels of granularity to fully protect the lowest-level resource
- Locks are always acquired 'top-down', from the table level down to individual rows
 - This forms the lock hierarchy



Lock Hierarchy

- DELETE statement begins executing that affects one row in a table
- A intent exclusive (IX) lock is acquired for the table
- A intent exclusive (IX) lock is acquired for the page containing the row
- A exclusive (X) lock is acquired for the row being modified



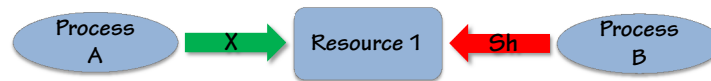
Lock Compatibility

- If a resource is already locked when a transaction requests a lock on it, the new lock can only be acquired if it is compatible with the existing lock on the resource
- The most common locks are shown here but a full compatibility matrix is available in Books Online (<http://bit.ly/SQLLockCompat>)

		Existing lock mode					
Requested mode		IS	S	U	IX	SIX	X
	Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
	Shared (S)	Yes	Yes	Yes	No	No	No
	Update (U)	Yes	Yes	No	No	No	No
	Intent exclusive (IX)	Yes	No	No	Yes	No	No
	Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
	Exclusive (X)	No	No	No	No	No	No

What is Blocking

- A condition where a transaction requests a lock mode that conflicts with a currently held lock and has to wait for that lock to be released



Troubleshooting Blocking

- Look at current waiting tasks in sys.dm_os_waiting_tasks DMV for a blocking_session_id and then look at the lock mode and resource
 - May also use the blocked process report in SQL Trace
- Common causes of blocking
 - Poor application design resulting in poor concurrency
 - Long running queries or transactions
 - Missing indexes that result in scan operations
 - Lock escalation during large data modifications
 - Bad query execution plans
- Resolving blocking
 - Change application design to improve concurrency
 - Minimize transaction duration and tune slower operations/indexing
 - Break large data modifications up into batches to reduce lock escalation

Demo

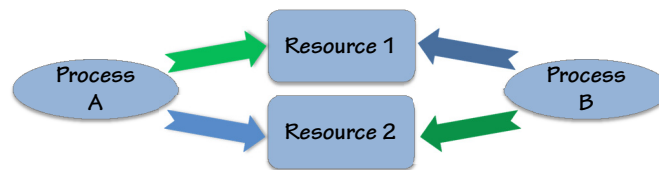
Blocking identification

Overview

- Waits and Queues methodology
- Collecting and analyzing Perfmon data
- Automating baseline data collections
- Blocking
- Deadlocks

What is a Deadlock

- A condition when two or more processes are holding locks on resources and are each waiting for the other to release its locks to progress, or where two or more processes are waiting for locks on resources in a circular chain



Misconceptions

- Deadlocks are a bug in SQL Server
- Deadlocks cannot be prevented
- Using NOLOCK on all SELECT statements is the best way to prevent deadlocks from occurring
- Adding covering indexes for every type of query will prevent deadlocks from occurring
- Troubleshooting deadlocks is a complex task that requires an experienced SQL Server developer or administrator

Deadlock Detection

- The lock monitor thread is responsible for deadlock detection and initiates periodic searches to identify and resolve deadlocks
- The default between deadlock searches is 5 seconds
- Each time a deadlock is detected, the search interval decreases to as low as 100ms based on the frequency of the deadlocks occurring in the server
 - When a deadlock search does not find a deadlock, the search interval increases again towards the 5 second default
 - When a deadlock is detected, the lock monitor thread assumes that the next lock waits are entering a deadlock cycle and will automatically trigger a deadlock search, allowing a true deadlock to be detected immediately
- During a deadlock search, the lock monitor identifies blocked tasks and then finds the blocking resource owner by recursively searching the tasks to identify the cyclic blocking that forms a deadlock

Deadlock Priority

- Any user can set the DEADLOCK_PRIORITY session option to control deadlock resolution behavior
 - It is not possible to stop a user setting DEADLOCK_PRIORITY, even with Resource Governor
- Setting a higher DEADLOCK_PRIORITY for important transactions will ensure that those transactions are not selected as the deadlock victim if a deadlock occurs with a lower priority session
 - Setting DEADLOCK_PRIORITY should usually not be used by developers to prevent deadlock involving SELECT statements
 - The exception is where deadlocks cannot be prevented in other ways, and it is critical that the SELECT succeeds

Deadlock Victim Selection

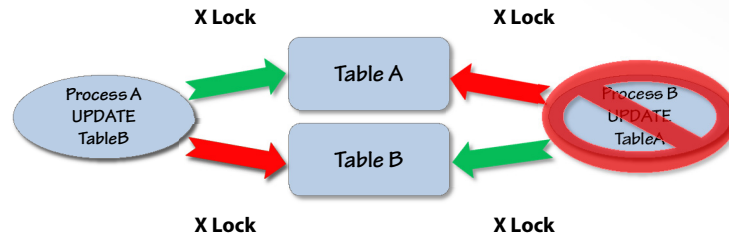
- **When a deadlock is detected, the lock monitor ends it by choosing one of the threads as the deadlock victim**
 - The deadlock victim is killed, rolling back its transaction
 - The client receives a 1205 error
- **The deadlock victim is selected based on the following criteria:**
 - The DEADLOCK_PRIORITY of the two sessions is compared and the lowest priority session is selected as the victim
 - If both of the sessions have the same DEADLOCK_PRIORITY value, the transaction that is the least expensive to rollback, based on the log records that have been generated, is selected as the victim (default)

Collecting Deadlock Graphs

- Trace flags (1222, 1205, 1204)
- SQL Trace / Profiler
- Event Notifications / WMI
- Extended Events

Reverse Object Order Deadlocks

- Two processes access objects in reverse order

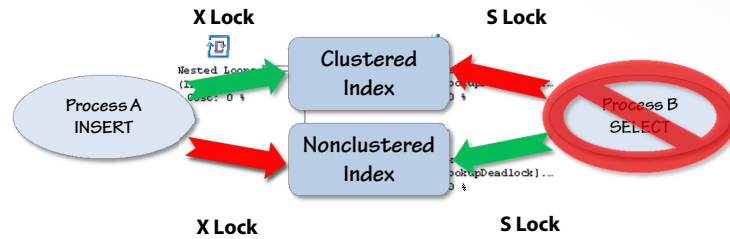


Demo

Reverse object order deadlocks

Bookmark Lookup Deadlock

- Bookmark lookup deadlock



Demo

Bookmark lookup deadlocks

Resolving Deadlocks

- Access objects in the same order within transactions
- Change indexing to cover queries
- Enable Read Committed Snapshot Isolation
 - Writers won't block readers
 - Reads won't block writers
- Change isolation level
- Use locking hints to force specific lock types to prevent lock conversion

Catching Deadlock Errors

- TRY/CATCH blocks in Transact-SQL can handle 1205 errors from deadlocks when they occur
 - The ERRORNUMBER() function will return the error number being raised
- ADO.NET can handle deadlocks when they occur by catching the SqlException that is raised by the 1205 error returned by SQL Server when a deadlock occurs
 - The Number property of SqlException will return the error number raised

Retrying After a Deadlock

- **Custom retry logic can be implemented to reattempt the operation that was selected as the deadlock victim**
 - Typically the lock scenario that resulted in the deadlock occurring only lasts a short duration, generally milliseconds, and will not exist when the transaction is resubmitted
 - The retry logic must be coded so that an infinite loop does not occur if the deadlocking persists in the engine
- **Logging of the deadlock can occur to allow for diagnosis and potential prevention in the future**

Review

- Waits and Queues methodology
- Collecting and analyzing Perfmon data
- Automating baseline data collections
- Blocking
- Deadlocks

Questions?

