SQL Server: Performance Troubleshooting Using Wait Statistics

Module 5: Troubleshooting Patterns

Paul S. Randal
Paul@SQLskills.com

Introduction

- **The core knowledge necessary to perform wait statistics analysis is an understanding of the most common wait types that are linked to performance problems**
- **There are many more wait types and latch classes that might become prevalent in the environment**
  - It is impractical to include every one in this course so the course covers the most common ones

- **In this module we'll cover:**
  - Common wait types
    - What they mean and how to investigate further
  - Common latch classes
    - What they mean and how to investigate further
  - Miscellaneous waits, latches, and spinlocks

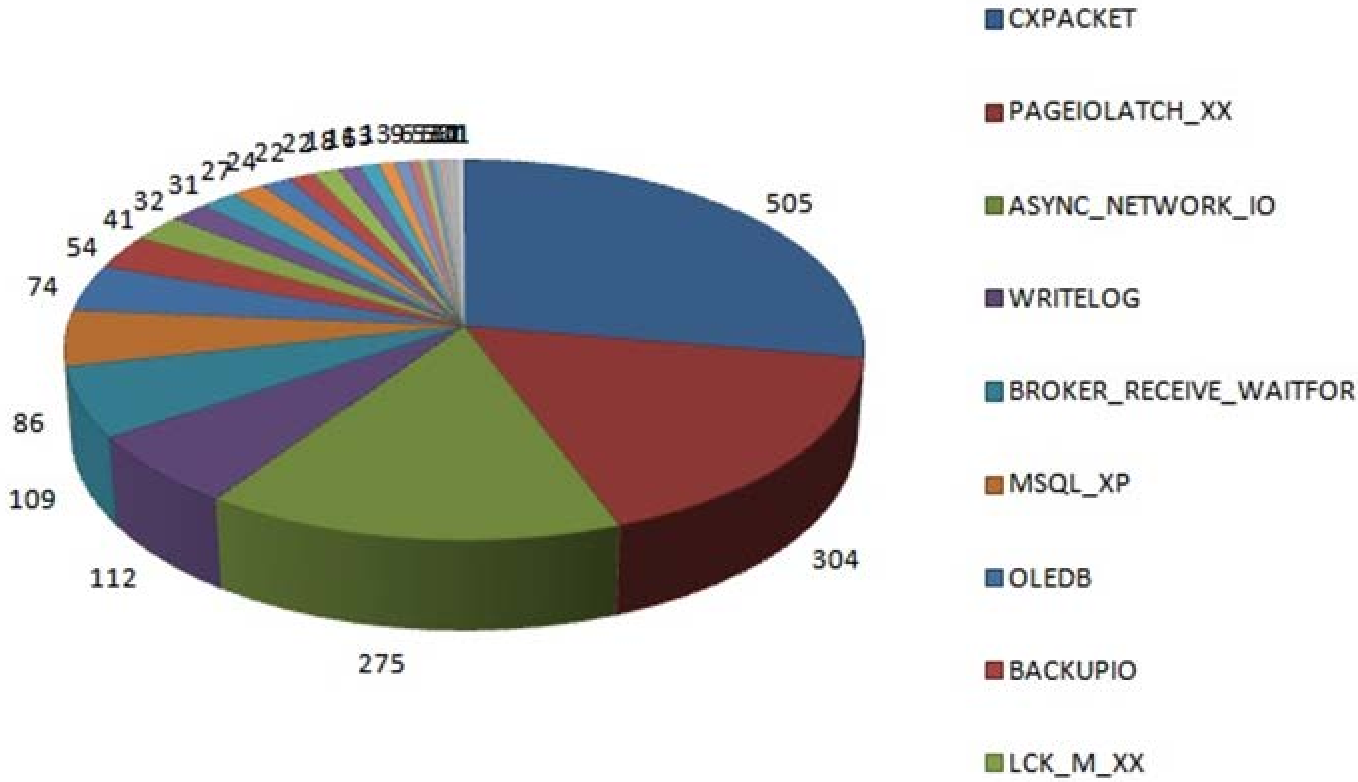
What is Relevant?

- **Just because there are waits, does not mean they are the problem**
- **Identify the top, relevant waits and then do further analysis**

- **For example, with 1000 waits for LCK_M_S, are they relevant?**
- **The answer is no, if:**
  - They occurred over 8 hours
  - There were 10 million locks acquired over the 8 hours
  - Total wait time for the LCK_M_S locks was only 50s altogether
- **The answer is yes, if:**
  - Each LCK_M_S wait was for 50s

Top Wait Types Worldwide Survey

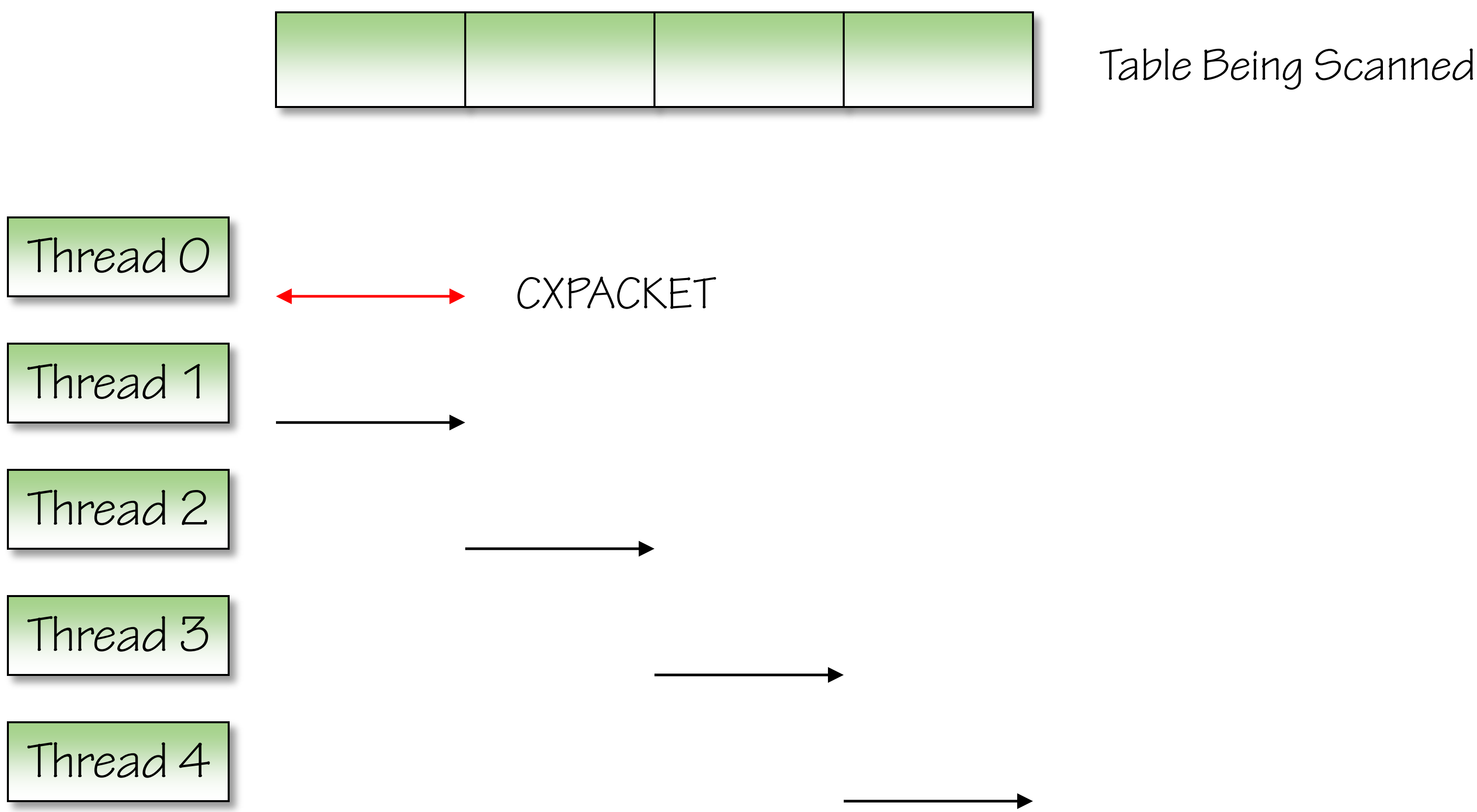
- **I conducted an Internet survey of top wait types in December 2010**
  - See http://bit.ly/fSWeO5
  - Results from more than 1800 SQL Server instances across the world



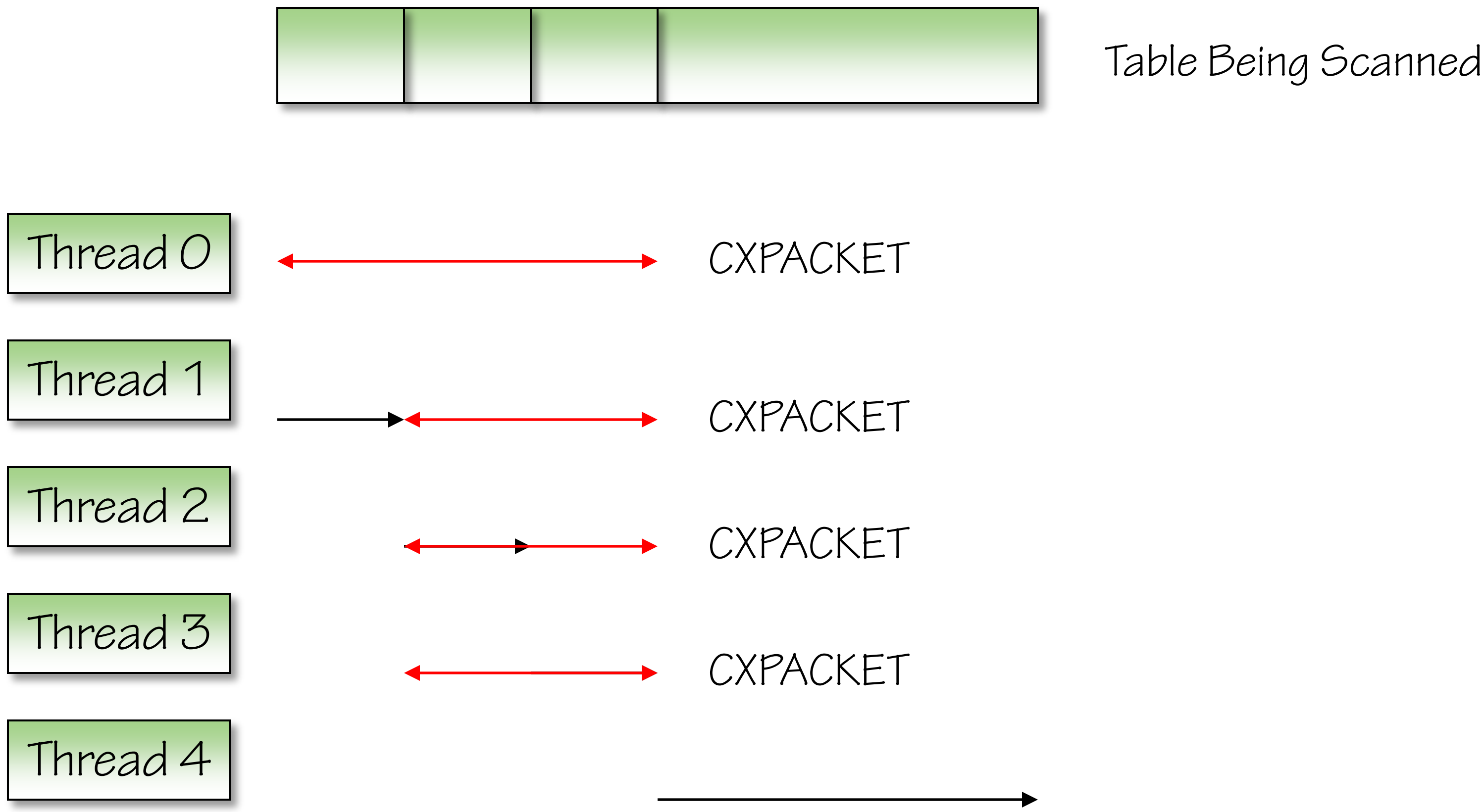
CXPACKET Wait Explanation

- **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



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-mode workloads
  - Consider using Resource Governor for MAX_DOP
  - Consider setting ‘cost threshold for parallelism’ higher than the query cost shown in the execution plan

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
  - Examine query plans for 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

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)

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

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/ki3bYl)

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

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 occuring, 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 Wait 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
    - See the common spinlocks slide near the end of the module for more information
    - More often than not, call stack analysis shows that spinlocks are *not* involved

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 How to download a sqlservr.pdb symbol file (http://bit.ly/Lc1cpj)
- **Enable trace flag 3656 to allow call stack symbol resolution**
- **Create an Extended Event session that:**
  - Captures sqlos.wait_info events for wait_type = 120
  - Captures the package0.callstack action
  - Uses the package0.asynchronous_bucketizer target
- **Run the workload and examine the captured call stacks**
- **Very advanced!**
  - See my blog post for a walk-through example (http://bit.ly/vJXjA6)

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

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

PREEMPTIVE_OS_WRITEFILEGATHER Wait

- **What does it mean:**
  - A thread is calling out to Windows to write to a file
- **Avoid knee-jerk response:**
  - Do not immediately assume the I/O subsystem has a problem
- **Further analysis:**
  - What database operations are under way?
  - For example, restore operation, database/file creation/growth/autogrowth
    - Look for other threads waiting for BACKUPTHREAD
- **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 (http://bit.ly/Lpz88m)

PREEMPTIVE_OS_WAITFORSINGLEOBJECT Wait

- **What does it mean:**
  - A thread is calling out to Windows to wait for the state of a synchronization object to change
  - Commonly seen with (but not limited to) NETWORK_IO and ASYNC_NETWORK_IO waits
- **Further analysis:**
  - Follow instructions as for ASYNC_NETWORK_IO
  - Check whether transactional replication is running
- **Possible root-causes and solutions:**
  - As for ASYNC_NETWORK_IO
  - Using the Shared Memory Provider when the client is on the same machine as SQL Server
  - When seen with NETWORK_IO, this is commonly transactional replication Agent jobs (such as the Log Reader and Distribution Agent jobs)
    - See Joe Sack's blog post for more details (http://bit.ly/AAtn5i)

BACKUPXX Waits

- **Examples:**
  - BACKUPBUFFER
  - BACKUPIO
  - BACKUPTHREAD
- **What do they mean:**
  - The first is waiting for data or a buffer for data
  - The second is reading from database data files
  - The third is usually while waiting for some I/O to complete on another thread, like zero initialization of a data or log file
- **Further analysis and root causes:**
  - For the first, are the backups going to a slow tape or over a slow network? Is the I/O subsystem on the remote server slow?
  - For the second, is the I/O subsystem for the data files slow?
  - For the third, see the PREEMPTIVE_OS_WRITEFILEGATHER wait

DBMIRRORXX Waits

- **Examples:**
  - DBMIRROR_EVENTS_QUEUE
  - DBMIRROR_SEND
  - DBMIRRORING_CMD
  - DBMIRROR_DBM_MUTEX
- **What does it mean:**
  - These are waits for database mirroring resources
- **Avoid knee-jerk response:**
  - Do not remove database mirroring or change to asynchronous mirroring
- **Further analysis:**
  - Analyze the average wait time for DBMIRROR_DBM_MUTEX
- **Possible root-causes:**
  - If the wait time for DBMIRROR_DBM_MUTEX is high, you may be trying to mirror too many databases or with too much mirroring activity
  - High wait times for all of these are due to general system bottlenecks

HADR_XX Waits

- **Examples:**
  - HADR_SYNC_COMMIT
  - HADR_SYNCHRONIZING_THROTTLE
  - And quite a few others
- **What do them mean:**
  - All are related to SQL Server 2012 Availability Groups
  - The first is the delay time for a transaction on the principal replica waiting for transaction log hardening on synchronous replicas
  - The second is transaction commits being throttled while a synchronous replica catches up to become synchronized
- **Further analysis and root-causes:**
  - The first could indicate a network problem, or performance issue with a synchronous replica
  - The second could be an indication that replicas are dropping out of being synchronized, with the potential for data loss in a disaster
  - See Joe Sack’s article for more details (http://bit.ly/LQFiMk)

TRACEWRITE and SQLTRACE_XX Waits

- **What does it mean:**
  - Threads are waiting to be able to write SQL Trace information
- **Avoid knee-jerk response:**
  - Do not necessarily stop tracing
- **Further analysis:**
  - Analyze the existing traces to see if high frequency events are being traced using sys.traces and sys.fn_trace_geteventinfo
  - Analyze the I/O subsystem where the trace data is being stored
- **Possible root-causes:**
  - Traces capturing more information than can be processed by SQL Trace
  - Rowset or File Providers not consuming trace data quickly enough
  - Third-party products spawning traces

LATCH_XX Wait

- **What does it mean:**
  - 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
    - Demonstrated in Module 4
  - 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 use a search engine to look for any information
    - My blog category on latches has information (http://bit.ly/Lc3J35)
    - The following slides discuss the most common latches to see

ACCESS_METHODS_XX Latches

- **Common examples:**
  - ACCESS_METHODS_DATASET_PARENT
  - ACCESS_METHODS_SCAN_RANGE_GENERATOR
  - ACCESS_METHODS_HOBT_VIRTUAL_ROOT
- **What do these mean:**
  - The first two latches are used to coordinate parallel scans and are the most common latches to see at the top of the sys.dm_os_latch_stats output
  - The third latch is used when traversing an index, and in EX mode when performing a B-tree root page split operation
- **Further analysis:**
  - Look for query plans with large scan or hash operations
  - Look for indexes with large numbers of page splits occurring
- **Possible root-causes:**
  - Unordered parallel scans have been known to not scale
  - Take corrective action to reduce page splits

FGCB_ADD_REMOVE Latch

- **What does it mean:**
  - 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
    - In SQL Server 2012, 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
    - In SQL Server 2008 and 2008 R2, the FGCB_ADD_REMOVE latch class number was 54
- **Possible root-causes:**
  - Auto-growth settings for a file are very low, requiring frequent growth, coupled with heavy, concurrent use of the filegroup
  - File sizes set too low for the rate of data entry into the database, requiring frequent auto-growth

DBCC_XX Latches

- **Examples:**
  - DBCC_MULTIOBJECTSCANNER (the most common to see)
  - DBCC_FILE_CHECK_OBJECT
  - DBCC_PFS_STATUS
  - DBCC_CHECK_AGGREGATE
  - DBCC_OBJECT_METADATA
- **Avoid knee-jerk response:**
  - Do not stop running consistency checks
- **Further analysis:**
  - None necessary as these are all internal to DBCC CHECKDB
- **Possible root-causes:**
  - The DBCC_MULTIOBJECTSCANNER latch was identified as a contention point and fixed in SQL Server 2012 and under a trace flag in SQL Server 2008 R2
  - See Bob Ward's blog post (http://bit.ly/yAFY7W) and KB article 2634571 (http://bit.ly/tK83Dk)

Miscellaneous Wait Types

- **EXECSYNC**
  - Parallel threads exchanging information during execution
- **ASYNC_IO_COMPLETION**
  - Waiting for a non-data-file I/O to complete, for example when zeroing a transaction log file or writing to backup media (especially over a network)
- **IO_COMPLETION**
  - Waiting for a non-data-file I/O to complete, for example when reading from a transaction log file during transaction rollback or with transactional replication
- **WRITE_COMPLETION**
  - Waiting for a non-buffered I/O to complete, for example when creating certain allocation bitmap pages, such as PFS pages
- **THREADPOOL**
  - Waiting for a worker thread to become available, for example on a heavily-loaded system with a lot of parallel queries running

Miscellaneous Wait Types (2)

- **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
- **LOGBUFFER**
  - Waiting for a log buffer to become available when flushing log contents

Miscellaneous Latch Classes

- **BUFFER**
  - Waiting for access to structures that map data file pages in the buffer pool
- **APPEND_ONLY_STORAGE_INSERT_POINT**
  - Waiting for access to create a row version in the tempdb version store
- **NESTING_TRANSACTION_FULL**
  - Synchronizing parallel reads using READ COMMITTED ISOLATION
- **DATABASE_MIRRORING_CONNECTION**
  - Waiting for access to the Service Broker endpoint used for database mirroring

Miscellaneous Spinlocks

- **OPT_IDX_STATS**
  - This is the Query Optimizer updating the structures used for sys.dm_db_index_usage_stats
  - This could be from many concurrent updates to table with lots of indexes
  - See KB article 2003031 (http://bit.ly/OgG7BE)
- **LOCK_HASH**
  - This is the Lock Manager looking in the list of hash buckets for lock hash collisions
  - See SQLCAT article on potential issues with DTC (http://bit.ly/McPz72)
- **LOGFLUSH_ACCESS and LOGFLUSHQ**
  - Involved with writing completed transaction log buffers to disk
  - Contention could be from very heavy load of very small transactions
  - See the WRITELOG wait for more details

Summary

- **There are many wait types and latch classes that might become prevalent in the environment**
- **Understanding their real meaning and how to do further analysis will save a lot of time when performance troubleshooting**

- **There are many more wait types than we can cover here**
  - Search in the Performance Tuning with Waits and Queues whitepaper and other information repositories for help (see the resource section of the final module)

- **In the next module we'll summarize the methodology and walk through a real-life client example**