

# Waits - collect and description of common wait types

Last updated by | Ricardo Marques | Mar 24, 2023 at 8:53 AM PDT

---

## Contents

- [How to check waits](#)
- List of some common wait types:
  - CXPACKET
  - PAGEIOLATCH XX
  - PAGELATCH XX
  - SOS SCHEDULER YIELD
  - ASYNC NETWORK IO
  - LCK XX XX
  - WRITELOG
  - HADR SYNC COMMIT
  - WAIT ON SYNC STATISTICS REFRESH
- [Internal reference](#)

## How to check waits

The waits can be collected in different ways

On the telemetry:

1. ASC report
2. Kusto. Check [Wait stats analysis](#)

On customer side:

1. Using the query below (overall wait stats):

```

WITH [Waits]
AS (SELECT wait_type, wait_time_ms/ 1000.0 AS [WaitS],
    (wait_time_ms - signal_wait_time_ms) / 1000.0 AS [ResourceS],
    signal_wait_time_ms / 1000.0 AS [SignalS],
    waiting_tasks_count AS [WaitCount],
    100.0 * wait_time_ms / SUM (wait_time_ms) OVER() AS [Percentage],
    ROW_NUMBER() OVER(ORDER BY wait_time_ms DESC) AS [RowNum]
FROM sys.dm_os_wait_stats WITH (NOLOCK)
WHERE [wait_type] NOT IN (
    N'BROKER_EVENTHANDLER', N'BROKER_RECEIVE_WAITFOR', N'BROKER_TASK_STOP',
    N'BROKER_TO_FLUSH', N'BROKER_TRANSMITTER', N'CHECKPOINT_QUEUE',
    N'CHKPT', N'CLR_AUTO_EVENT', N'CLR_MANUAL_EVENT', N'CLR_SEMAPHORE', N'CXCONSUMER',
    N'DBMIRROR_DBM_EVENT', N'DBMIRROR_EVENTS_QUEUE', N'DBMIRROR_WORKER_QUEUE',
    N'DBMIRRORING_CMD', N'DIRTY_PAGE_POLL', N'DISPATCHER_QUEUE_SEMAPHORE',
    N'EXECSYNC', N'FSAGENT', N'FT_IFTS_SCHEDULER_IDLE_WAIT', N'FT_IFTSHC_MUTEX',
    N'HADR_CLUSAPI_CALL', N'HADR_FABRIC_CALLBACK', N'HADR_FILESTREAM_IOMGR_IOCOMPLETION', N'HADR_LOGCAPTURE_WAIT',
    N'HADR_NOTIFICATION_DEQUEUE', N'HADR_TIMER_TASK', N'HADR_WORK_QUEUE',
    N'KSOURCE_WAKEUP', N'LAZYWRITER_SLEEP', N'LOGMGR_QUEUE',
    N'MEMORY_ALLOCATION_EXT', N'ONDEMAND_TASK_QUEUE',
    N'PARALLEL_REDO_DRAIN_WORKER', N'PARALLEL_REDO_LOG_CACHE', N'PARALLEL_REDO_TRAN_LIST',
    N'PARALLEL_REDO_WORKER_SYNC', N'PARALLEL_REDO_WORKER_WAIT_WORK',
    N'PREEMPTIVE_HADR_LEASE_MECHANISM', N'PREEMPTIVE_SP_SERVER_DIAGNOSTICS',
    N'PREEMPTIVE_OS_LIBRARYOPS', N'PREEMPTIVE_OS_COMOPS', N'PREEMPTIVE_OS_CRYPTOPS',
    N'PREEMPTIVE_OS_PIPEOPS', N'PREEMPTIVE_OS_AUTHENTICATIONOPS',
    N'PREEMPTIVE_OS_GENERICOPS', N'PREEMPTIVE_OS_VERIFYTRUST',
    N'PREEMPTIVE_OS_FILEOPS', N'PREEMPTIVE_OS_DEVICEOPS', N'PREEMPTIVE_OS_QUERYREGISTRY',
    N'PREEMPTIVE_OS_WRITEFILE',
    N'PREEMPTIVE_XE_CALLBACKEXECUTE', N'PREEMPTIVE_XE_DISPATCHER',
    N'PREEMPTIVE_XE_GETTARGETSTATE', N'PREEMPTIVE_XE_SESSIONCOMMIT',
    N'PREEMPTIVE_XE_TARGETINIT', N'PREEMPTIVE_XE_TARGETFINALIZE',
    N'PWAIT_ALL_COMPONENTS_INITIALIZED', N'PWAIT_DIRECTLOGCONSUMER_GETNEXT',
    N'PWAIT_EXTENSIBILITY_CLEANUP_TASK',
    N'QDS_PERSIST_TASK_MAIN_LOOP_SLEEP', N'QDS_ASYNC_QUEUE',
    N'QDS_CLEANUP_STALE_QUERIES_TASK_MAIN_LOOP_SLEEP', N'REQUEST_FOR_DEADLOCK_SEARCH',
    N'RESOURCE_QUEUE', N'SERVER_IDLE_CHECK', N'SLEEP_BPOOL_FLUSH', N'SLEEP_DBSTARTUP',
    N'SLEEP_DCOMSTARTUP', N'SLEEP_MASTERDBREADY', N'SLEEP_MASTERMDREADY',
    N'SLEEP_MASTERUPGRADED', N'SLEEP_MSDBSTARTUP', N'SLEEP_SYSTEMTASK', N'SLEEP_TASK',
    N'SLEEP_TEMPDBSTARTUP', N'SNI_HTTP_ACCEPT', N'SOS_WORK_DISPATCHER',
    N'SP_SERVER_DIAGNOSTICS_SLEEP',
    N'SQLTRACE_BUFFER_FLUSH', N'SQLTRACE_INCREMENTAL_FLUSH_SLEEP', N'SQLTRACE_WAIT_ENTRIES',
    N'STARTUP_DEPENDENCY_MANAGER',
    N'WAIT_FOR_RESULTS', N'WAITFOR', N'WAITFOR_TASKSHUTDOWN', N'WAIT_XTP_HOST_WAIT',
    N'WAIT_XTP_OFFLINE_CKPT_NEW_LOG', N'WAIT_XTP_CKPT_CLOSE', N'WAIT_XTP_RECOVERY',
    N'XE_BUFFERMGR_ALLPROCESSED_EVENT', N'XE_DISPATCHER_JOIN',
    N'XE_DISPATCHER_WAIT', N'XE_LIVE_TARGET_TVF', N'XE_TIMER_EVENT')
AND waiting_tasks_count > 0)
SELECT
    MAX (W1.wait_type) AS [WaitType],
    CAST (MAX (W1.Percentage) AS DECIMAL (5,2)) AS [Wait Percentage],
    CAST ((MAX (W1.WaitS) / MAX (W1.WaitCount)) AS DECIMAL (16,4)) AS [AvgWait_Sec],
    CAST ((MAX (W1.ResourceS) / MAX (W1.WaitCount)) AS DECIMAL (16,4)) AS [AvgRes_Sec],
    CAST ((MAX (W1.SignalS) / MAX (W1.WaitCount)) AS DECIMAL (16,4)) AS [AvgSig_Sec],
    CAST (MAX (W1.WaitS) AS DECIMAL (16,2)) AS [Wait_Sec],
    CAST (MAX (W1.ResourceS) AS DECIMAL (16,2)) AS [Resource_Sec],
    CAST (MAX (W1.SignalS) AS DECIMAL (16,2)) AS [Signal_Sec],
    MAX (W1.WaitCount) AS [Wait Count]
FROM Waits AS W1
INNER JOIN Waits AS W2
ON W2.RowNum <= W1.RowNum
GROUP BY W1.RowNum, W1.wait_type
HAVING SUM (W2.Percentage) - MAX (W1.Percentage) < 99 -- percentage threshold
OPTION (RECOMPILE);

```

2. For queries that are currently running (look at the columns QUERY and INDIVIDUAL QUERY to identify the query that you are looking for. After this look at Wait type - this will show you what is the current wait type

for that query):

```

SET NOCOUNT ON ; SELECT      @@SERVERNAME Server,
                                [SPID] = SESSION_ID ,
                                percent_complete [%] ,
                                [DATABASE] = DB_NAME(SP.DBID) ,
                                [STATUS] = ER.STATUS ,
                                [WAIT] = WAIT_TYPE ,
                                wait_resource ,
                                reads ,
                                writes ,
                                logical_reads ,
                                command ,
                                [INDIVIDUAL QUERY] = SUBSTRING(QT.TEXT,
                                                                ER.STATEMENT_START_OFFSET
                                                                / 2,
                                                                ( CASE WHEN ER.STATEMENT_END_OFFSET = -1
                                                                  THEN LEN(CONVERT(NVARCHAR(MAX), QT.TEXT))
                                                                  * 2
                                                                  ELSE ER.STATEMENT_END_OFFSET
                                                                END
                                                                - ER.STATEMENT_START_OFFSET )
                                                                / 2) ,
                                [QUERY] = QT.TEXT ,
                                PROGRAM = PROGRAM_NAME ,
                                [USER] = NT_USERNAME ,
                                HOSTNAME ,
                                NT_DOMAIN ,
                                START_TIME ,
                                QP.query_plan AS xml_batch_query_plan
FROM      sys.dm_exec_requests ER WITH (NOLOCK)
          INNER JOIN sys.sysprocesses SP WITH (NOLOCK) ON ER.SESSION_ID = SP.SPID
          CROSS APPLY sys.dm_exec_sql_text(ER.SQL_HANDLE) AS QT
          CROSS APPLY sys.dm_exec_query_plan(ER.plan_handle) QP
WHERE     SESSION_ID > 50
          AND SESSION_ID NOT IN ( @@SPID ) ORDER BY 1 ,2

```

3. On the Actual Execution Plan. Check [Execution Plans](#)

## List of some common wait types:

### CXPACKET

Description:

According with books online: "Occurs with parallel query plans when trying to synchronize the query processor exchange iterator. If waiting is excessive and cannot be reduced by tuning the query (such as adding indexes), consider adjusting the cost threshold for parallelism or lowering the degree of parallelism."

In a nutshell, means that a parallel plan is running. Parallelism occurs when the cost threshold for parallelism is passed, or if the query contains a query hint (for example, "OPTION (MAXDOP 8)").

A friendly picture would be: imagine that you have to find a specific value on a set of 1000 pages. This task would be faster if you split 1000 pages between your colleagues. After splitting the "workload", each colleague would find the results and give them to you. Then you would process the final result.

If the number of total pages was 4, most probably you would choose to do it by yourself.

When parallelism occurs on SQL Server the logic is the same - spread the work between multiple threads to get the result faster.

What not to do:

- Reduce the database or instance MAXDOP to 1. Decreasing this value can result on poor performance of other tasks that requires parallelism - for example, a large report.

What to look:

The occurrence of parallelism, like mentioned before, means that the cost threshold was passed. Like so, try first to understand if parallelism is in fact a problem. In some cases parallelism can be beneficial (think about the example mentioned before).

The presence of parallelism can sometimes mean that the query is not optimized (reason why the parallelism threshold is passed), or an non optimal plan is compiled.

Like so a good place to start is to:

- Check the execution plan and look for scans caused by implicit conversions
- Check for tuning opportunities. Sometimes it's only missing an index.
- Check if statistics are updated.
- Other tuning opportunities (query rewrite, etc.)

You can also force the query to run with less threads.

Note: always double check the instance MAXDOP, if it's set to 0.

## **PAGEIOLATCH XX**

Description: This wait types can change depending on the operation that is being done. For example, we have PAGEIOLATCH\_EX (Exclusive), PAGEIOLATCH\_SH (Shared), etc.

According with books online this wait type is "Occurs when a task is waiting on a latch for a buffer that is in an I/O request."

In short this means that a thread is waiting for a page to be retrieved from disk. For example, if the wait is PAGEIOLATCH\_EX, it means that the thread is waiting for a page to be read from disk. This page will be changed when it's in memory.

PAGEIOLATCH\_SH is the most common wait type.

What not to do:

Assume that there is a problem with the storage. This can be case, but most of the times the problem is located elsewhere.

What to look:

Usually the problem is related with an high number of requests, not with a problem on SQL Server. Like so the problem can looked from different perspectives:

- Memory pressure. Since pages don't live long on the buffer cache. So more requests are done to the IO subsystem. On this case, look for queries with high memory grants, outdated statistics, missing indexes or other tuning opportunities. Check [Page Life Expectancy](#). To check queries with high memory grants on Kusto:

```
MonWidsExecStats
| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}" and AppName
| where is_primary == 1
| extend MaxQueryMemoryMB=round(max_max_query_memory_pages*8.0/1024,1)
| summarize max(MaxQueryMemoryMB) by query_hash
| top 20 by max_MaxQueryMemoryMB desc nulls last
```

- Large scans caused by a missing index or by an implicit conversion.
- When an instance is restarted the buffer cache is empty. In some cases, when an instance comes online an increase can be observed since the pages need to be retrieved from disk.

## PAGELATCH XX

Description:



From books online: "Occurs when a task is waiting on a latch for a buffer that is in an I/O request".

It's very similar to PAGEIOLATCH, but on this case it's not related with IO. The page is already in memory.

What not to do:

Always keep in mind that this is not an IO related

What to look: When the wait type is PAGELATCH\_EX, look last-page insert problems -


<https://learn.microsoft.com/en-us/troubleshoot/sql/performance/resolve-pagelatch-ex-contention> . There are multiple ways on how to solve this type of issue, but note that require testing from customer side - <https://learn.microsoft.com/en-us/troubleshoot/sql/performance/resolve-pagelatch-ex-contention#2-choose-a-method-to-resolve-the-issue> 

When the wait type is PAGELATCH\_SH or PAGELATCH\_UP, look for tempdb contention.

## SOS SCHEDULER YIELD

Description: From books online: "Occurs when a task voluntarily yields the scheduler for other tasks to execute. During this wait the task is waiting for its quantum to be renewed."

This means that a thread moved to the Runnable queue.

What to look: When this wait type is high look for tuning opportunities, especially large scans Knee-Jerk Wait Statistics : [SOS SCHEDULER YIELD](#) 

## ASYNC NETWORK IO

Description: Books online "Occurs on network writes when the task is blocked waiting for the client application to acknowledge it has processed all the data sent to it. Verify that the client application is processing data from

the server as fast as possible or that no network delays exist."

In short, the thread is waiting for something outside of SQL - client or network.

What not to do: Assume that there is a network problem. Can be the cause, but usually the problem is elsewhere

What to look: Since the thread is waiting on something outside of SQL, there's not much cannot be done. The problem can have multiple reasons, like for example:

- Network issue (not usual, but still a possibility)
- Number of rows being returned (when a customer is querying millions of rows)
- Slow client

## LCK XX XX

Description: A thread is waiting to acquire a lock (can be Exclusive, Update, Shared, etc).

What not to do: When finding the lead blocker, just kill the transaction. Look carefully first.

What to look: Find the lead blocker first - [Blocking](#)

If the lead blocker is still running, try to tune the query.

If it's on sleeping state, it can mean different things:


- Someone opened a transaction and didn't committed (look at the client name).
- Problems with the application. Could be that is not committing the transaction because is waiting for another process to finish.
- Code error - for example, a stored procedure where a COMMIT wasn't included.

Eventually the solution would be to kill the transaction, but if it's included on the last two hypothesis will not help on a reoccurrence of the problem.

Knowing what is the lead blocker, you can dbcc inputbuffer(spид\_number) to have an idea of the code on the transaction (note that will only contain the first command executed inside the transaction).

## WRITELOG

Description: Books online: "Occurs while waiting for a log flush to complete. Common operations that cause log flushes are transaction commits and checkpoints."

What to look: Identify the query and look for opportunities on reducing the transaction log generated. Look also at Managed Instance Max log throughput - <https://learn.microsoft.com/en-us/azure/azure-sql/managed-instance/resource-limits?view=azuresql#service-tier-characteristics> 

## HADR SYNC COMMIT

Description: Waiting for log block to be hardened on remote replica.

[https://learn.microsoft.com/en-us/archive/blogs/psssql/alwayson-hadr-on-learning-series-hadr\\_sync\\_commit-vs-writelog-wait](https://learn.microsoft.com/en-us/archive/blogs/psssql/alwayson-hadr-on-learning-series-hadr_sync_commit-vs-writelog-wait) 

What to look: When you see this wait happening look for network issues between primary and secondary, and for performance problems on the secondary.

Look especially for an increase on write activity - increase on DML operations.




```
MonWiQdsExecStats
| where TIMESTAMP > datetime(2023-03-02 23:00)-3d and TIMESTAMP <datetime(2023-03-02 23:00)+3d
| where LogicalServerName =~ 'server_name' and database_name =~ 'database_name'
| extend statement = iif(statement_type in ("x_estypDelete", "x_estypInsert", "x_estypSelectInto", "x_estypMer
| summarize sum(execution_count) by bin(TIMESTAMP,15m), statement
```

## WAIT ON SYNC STATISTICS REFRESH

Description: Related with Auto Statistics Update. When a query is submitted and data changes reached the data changes threshold, statistics are updated before compiling the query.

The query will only resume after the statistics update is finished.

What to look: If this wait type is becoming a problem for the customer workload, check one of the following possibilities:

- Increase the frequency of UPDATE STATISTICS jobs (assuming that the customer has one). It might be a tricky solution if we are dealing with tables where changes are very frequent.
- Change auto update stats to asynchronous (<https://techcommunity.microsoft.com/t5/azure-sql-blog/improving-concurrency-of-asynchronous-statistics-update/ba-p/1441687> ). This needs to be tested by the customer since the query will execute with the last available execution plan, that might not be optimal.
- Disable auto update statistics (<https://blog.sqlauthority.com/2020/06/25/sql-server-disable-statistics-update-on-a-specific-table/>  and <https://www.mssqltips.com/sqlservertip/2766/sql-server-auto-update-and-auto-create-statistics-options/> ) and rely only on update statistics maintenance jobs that the customer might have. Can also be not optimal for the same reason as the first point mentioned.

## Internal reference

[372218827](#) 

## How good have you found this content?

