Diagnosing and resolving parallelism-related latching and blocking in SQL Server using DMV’s, the activity monitor, procedure execution plans, and index tuning.

**Background:** Our IT department recently made a change to a web-based survey that caused a stored procedure to execute about 150 times more than usual.  The change made the procedure fire every 6 seconds while the webpage was open instead of firing once on a webpage hyperlink click. This increased the daily execution count of this one particular procedure from 800-1,000 times per day to over 150,000 times per day.

**Issue:** Once the change was made, a substantial amount of latching and blocking started showing up in the activity monitor.  The wait statistic messages showing in the activity monitor

were:

Figure 1-1 Session running with parallelism waiting for exclusive latch

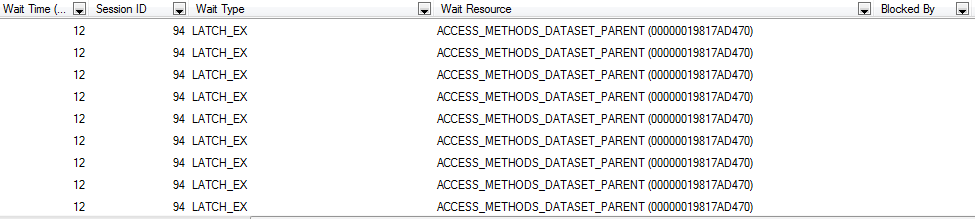
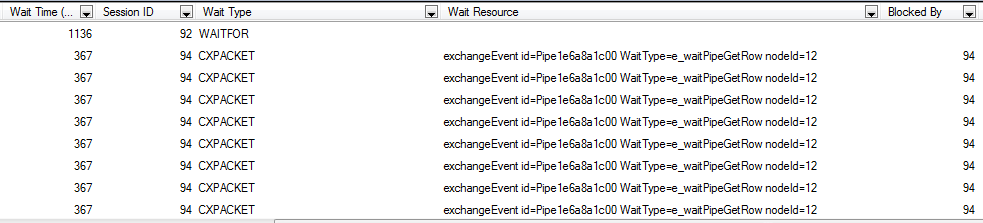


Figure 1-2 Session running with parallelism waiting for all threads to finish



What I saw were two wait types associated normally with parallelism in SQL Server. We had sessions waiting to acquire exclusive latches (LATCH\_EX) in order to wrap up the parallel processes running and we were also seeing CXPACKET waits which are a result of multiple threads from one process waiting for a another thread to finish a task. A humorous analogy made by [Tim Ford](http://sqlmag.com/database-performance-tuning/my-query-not-waiting-because-parallelism) is a bunch of construction workers standing around a hole and watching one guy dig. All of the resources are tied up in the one task, but only one resource is doing anything. This is evident by looking at the session ID of the CXPACKET rows and then looking at the session ID that is causing the blocking. Each row represents a thread that is waiting for another resource in the same session to finish up.

The built in DMV’s for wait stats also showed an extreme jump in waiting tasks and in the wait stats for these specific wait messages.

sys.dm\_os\_wait\_stats

sys.dm\_os\_waiting\_tasks

**Investigation:** I executed the stored procedure to confirm it was executing at the same speed we were used to.  It ran as expected and executed in a little under 2 seconds so there did not appear to be anything different with the procedure.  I opened the execution plan and found the procedure was doing clustered index scans using parallelism on a million row table.  Each scan was taking 99% of the resources from the batch and running 32 threads since we had no MAXDOP set on our server.

Figure 2-1 Execution plan using a Clustered Index Scan and parallelism

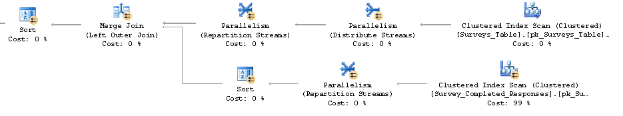
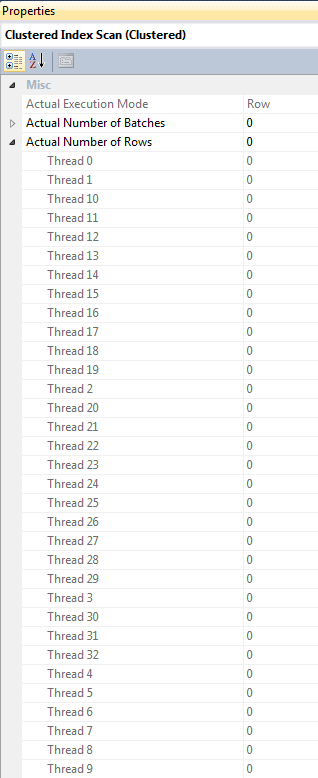


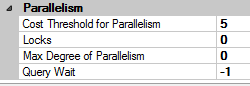
Figure 2-2 One plan running 32 threads in parallel with MAXDOP as 0



I used sp\_HelpText (sometimes defaulted as ALT+F1 shortcut in SSMS) and found the table only had a clustered index on an identity ID column and no non-clustered indexes designed for this procedure.

As mentioned above, the properties on the plan showed the Clustered Index Scan using 32 threads since our MAXDOP on the server was set to a default 0 (can be found in by right-clicking on your server instance in the object explorer and selecting Properties > Advanced).

Figure 3-1 Server Instance Properties



Since parallelism isn’t necessarily a bad thing, I’m sure the query was fine at the time it was written for serving the purpose of getting executed 800-1000 times daily. Once the environment changed and the procedure was now being executed every 6 seconds by potentially hundreds of concurrent sessions, it caused a problem.

**Solution:** I tuned the query and added a covering index for this specific procedure.  Once recompiled, all parallelism was removed from the execution plan and the latching and blocking essentially disappeared in a matter of minutes.

Figure 3-2 Execution plan with covering index and no parallelism

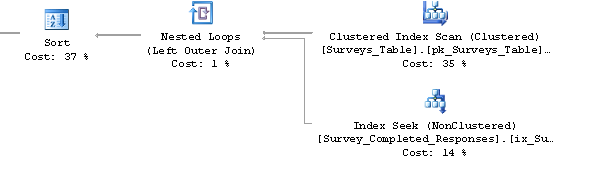


Figure 3-3 Index Seek Properties

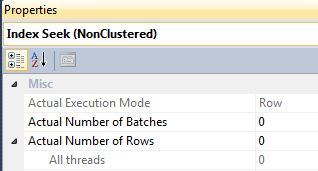
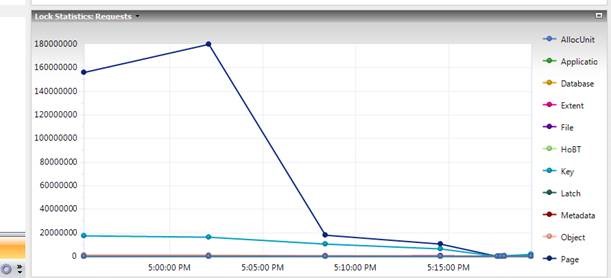
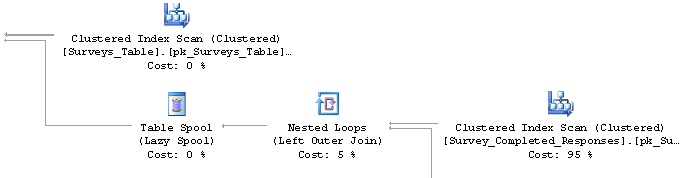


Figure 4-1 Decrease in Wait statistics after recompiling procedure with new index



An alternative method would be to use a query hint in the procedure to force a specific MAXDOP of 1 to remove the parallelism, but that method is more masking the issue than addressing it. Without a covering index, the query would still be performing the large scans instead of seeking and performance would still suffer.

Figure 4-2 Original execution plan using Clustered Index and a MAXDOP of 1



Microsoft has published [recommendations for MAXDOP](https://support.microsoft.com/en-us/kb/2806535) that cover guidelines for how to determine best levels or parallelism at a server level. Other articles that are extremely helpful regarding parallelism and latching include:

[Understanding and Using Parallelism in SQL Server](https://www.simple-talk.com/sql/learn-sql-server/understanding-and-using-parallelism-in-sql-server/) by Paul White

[My Query is (NOT) Waiting Because of Parallelism?](http://sqlmag.com/database-performance-tuning/my-query-not-waiting-because-parallelism) by Tim Ford

[Parallel Execution Plans – Branches and Threads](http://sqlperformance.com/2013/10/sql-plan/parallel-plans-branches-threads) by Paul White

David Fundakowski