**A closer look at CXPACKET wait type in SQL Server**

By: [Matteo Lorini](https://www.mssqltips.com/sqlserverauthor/42/matteo-lorini/)

FROM: <https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/>

**Problem**

If you have a server that has more than one CPU core, you may experience high values of CXPACKET wait types.  This is typically due to queries that run in parallel and the real issue is to understand how different versions of a query can impact CXPACKET waits.  In this tip we examine how changing a query can impact CXPACKET waits.

**Solution**

This goal of this tip is to increase query performance, decrease high CXPACKET waits without reducing MAXDOP.  We will look at a few examples of a query and the impact on CXPACKET waits.

If you are not familiar with CXPACKET, you can learn more here <http://msdn.microsoft.com/en-us/library/ms179984.aspx>.

These waittypes are all involved in parallel query execution. These waittypes indicate that the SPID is waiting on a parallel process to complete or start. Let's see it with the following examples. Please note that all the examples below have been provided by [http://www.sqlworkshops.com](http://www.sqlworkshops.com/).

First we will create a new table.

SET nocount ON   
CREATE TABLE orders (d\_id INT, o\_id INT, o\_amount INT, o\_description CHAR(2000))   
CREATE UNIQUE CLUSTERED INDEX test ON orders(d\_id, o\_id)

Then insert 800,000 rows.

BEGIN TRAN   
DECLARE @i INT   
SET @i = 1   
WHILE @i <= 800000   
BEGIN   
INSERT INTO orders VALUES (@i % 8, @i, RAND() \* 800000, REPLICATE('a', 2000))   
SET @i = @i + 1   
END   
COMMIT TRAN

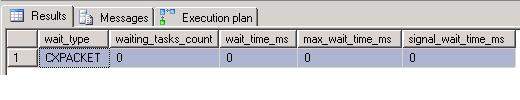
Then update the statistics with a full scan, so the optimizer can work easier.

UPDATE STATISTICS orders WITH fullscan   
GO   
CREATE TABLE #department (d\_id INT)   
INSERT INTO #department VALUES(0)   
INSERT INTO #department VALUES(1)   
INSERT INTO #department VALUES(2)   
INSERT INTO #department VALUES(3)   
INSERT INTO #department VALUES(4)   
INSERT INTO #department VALUES(5)   
INSERT INTO #department VALUES(6)   
INSERT INTO #department VALUES(7)   
GO

TThen before we begin, enable statistics time to observe CPU time and elapsed time. Include the actual execution plan, clear the wait stats and make sure the system you are using is idle otherwise, the CXPACKET will not reflect our query waits alone. The server I tested on has 2 quad core CPUs for a total of 8 cores.

Test 1 - Run query using 1 processor (MAXDOP 1)

SET STATISTICS time ON   
GO   
DBCC sqlperf('sys.dm\_os\_wait\_stats', clear)  
   
DECLARE @order\_amount INT   
  
SELECT @order\_amount = MAX(o\_amount)   
      FROM orders o INNER JOIN #department d ON (o.d\_id = d.d\_id)   
      OPTION (maxdop 1)  
   
SELECT \* FROM sys.dm\_os\_wait\_stats   
      WHERE wait\_type = 'CXPACKET'  
  
SQL Server Execution Times:    
   CPU time = 688 ms,  elapsed time = 689



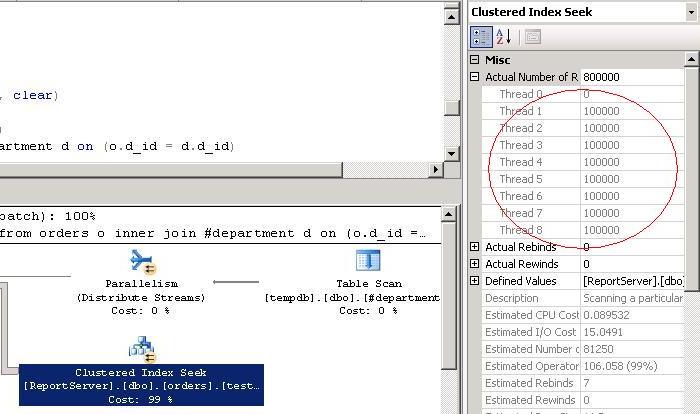
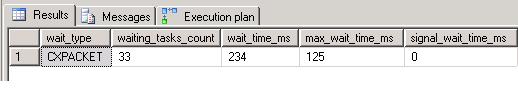
As we might notice the CPU time is very close to elapsed time. The query is CPU bound; if not execute the query again to make sure we are reading the data from cache. This example is to understand parallel query execution, not disk IO, so we need to get the data in the cache before we can continue.

We executed the query with MAXDOP 1 that is why the CXPACKET wait time is 0 because the query executes serially.

Test 2  - Let's run it again in parallel (MAXDOP 0)

DBCC sqlperf('sys.dm\_os\_wait\_stats', clear)  
  
DECLARE @order\_amount INT   
  
SELECT @order\_amount = MAX(o\_amount)   
FROM orders o INNER JOIN #department d ON (o.d\_id = d.d\_id)   
OPTION (maxdop 0)   
  
SELECT \* FROM sys.dm\_os\_wait\_stats   
WHERE wait\_type = 'CXPACKET'   
GO   
  
SQL Server Execution Times:   
   CPU time = 874 ms,  elapsed time = 221 ms.

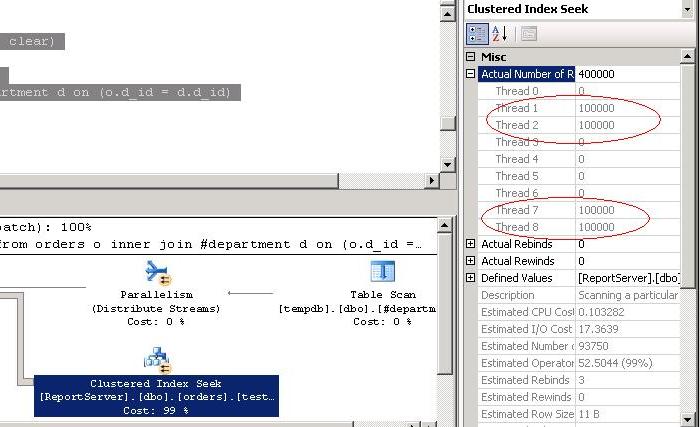
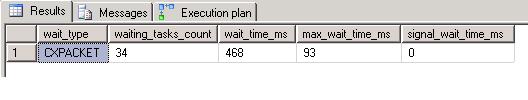
On the execution plan, right click on Cluster Index Seek and click on Properties as shown below. We can see that SQL Server had distributed 100,000 rows across each of my 8 CPUs. Notice the CPU time and elapsed time. Elapsed time (221ms) is similar to CXPACKET wait time (234 ms) as no child threads are entirely idle and the coordinator in this case waits for all child threads to complete processing the rows with CXPACKET wait type.

Test 3 - Let's drop and recreate our #department table and only insert 4 rows this time.

DROP TABLE #department   
--insert into #department values (0)   
INSERT INTO #department VALUES (1)   
--insert into #department values (2)   
INSERT INTO #department VALUES (3)   
--insert into #department values (4)   
INSERT INTO #department VALUES (5)   
--insert into #department values (6)   
INSERT INTO #department VALUES (7)   
  
DBCC sqlperf('sys.dm\_os\_wait\_stats', clear)  
   
DECLARE @order\_amount INT   
  
SELECT @order\_amount = MAX(o\_amount)   
FROM orders o INNER JOIN #department d ON (o.d\_id = d.d\_id)   
OPTION (maxdop 0)   
  
SELECT \* FROM sys.dm\_os\_wait\_stats   
WHERE wait\_type = 'CXPACKET'   
GO   
  
SQL Server Execution Times:   
   CPU time = 454 ms,  elapsed time = 105 ms.

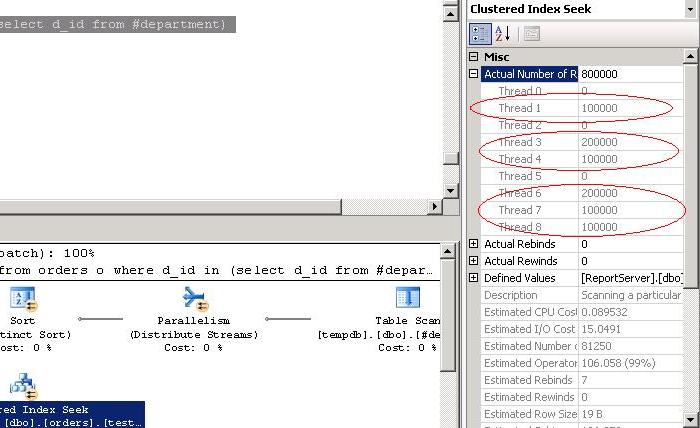
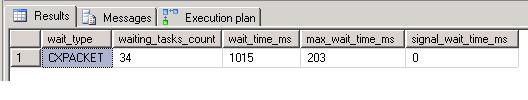
The CXPACKET wait time should be approximately 4 times the elapsed time and CPU time should be approximately 4 times the elapsed time. The reason is only 4 threads are processing the rows, 100,000 rows each (4 threads X elapsed time = CPU time), the other 5 threads (4 child threads and the coordinator thread) wait with CXPACKET wait time (5 threads X elapsed time = CXPACKET wait time).

Test 4 - Let's now re-write our query using an IN instead of an INNER JOIN

DROP TABLE #department   
CREATE TABLE #department (d\_id INT)   
INSERT INTO #department VALUES (0)   
INSERT INTO #department VALUES (1)   
INSERT INTO #department VALUES (2)   
INSERT INTO #department VALUES (3)   
INSERT INTO #department VALUES (4)   
INSERT INTO #department VALUES (5)   
INSERT INTO #department VALUES (6)   
INSERT INTO #department VALUES (7)   
GO   
  
DBCC sqlperf('sys.dm\_os\_wait\_stats', clear)  
   
DECLARE @order\_amount INT   
  
SELECT @order\_amount = MAX(o\_amount)   
FROM orders o WHERE d\_id IN (SELECT d\_id FROM #department)   
OPTION (maxdop 0)   
  
SELECT \* FROM sys.dm\_os\_wait\_stats   
WHERE wait\_type = 'CXPACKET'   
GO   
  
SQL Server Execution Times:   
   CPU time = 860 ms,  elapsed time = 212 ms.

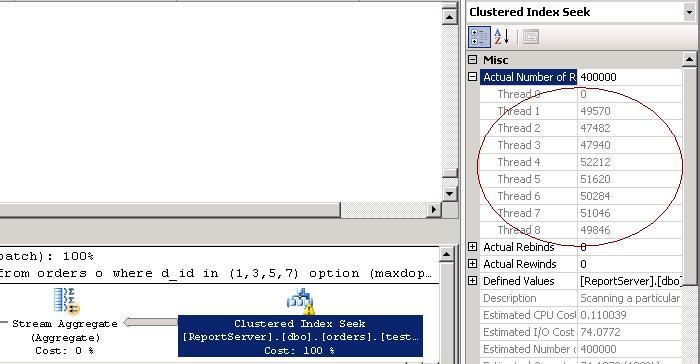
If you take a look at the execution plan, we can see that not all threads processed 100,000 rows. Some threads processed 200,000 rows, some 100,000 rows, and some none. This is the reason for high CXPACKET wait time (due to idle threads waiting on CXPACKET) and higher elapsed time (uneven distribution of rows).

Test 5 - Let's try this final example without using the temporary table

DBCC sqlperf('sys.dm\_os\_wait\_stats', clear)  
   
DECLARE @order\_amount INT   
  
SELECT @order\_amount = MAX(o\_amount)   
FROM orders o WHERE d\_id IN (0,3,5,7)   
OPTION (maxdop 0)   
  
SELECT \* FROM sys.dm\_os\_wait\_stats   
      WHERE wait\_type = 'CXPACKET'   
GO   
  
SQL Server Execution Times:   
   CPU time = 578 ms,  elapsed time = 72 ms.

Finally we can see that in the above query, all threads are processing rows, leading to low elapsed time. Since no child threads were entirely idle, elapsed time is similar to CXPACKET wait time as only the coordinator thread waited with CXPACKET wait type.



Summary

In all the above examples, join, subquery and IN, when handling 4 rows from a temporary table you will see the elapsed time to be less than with an IN clause. This does not mean that an IN clause is always better.  This is not a discussion about efficiency of IN clause or joins or subqueries, but about how to tune certain parallel queries for shortest elapsed time and to see and understand how to interpret CXPACKET wait type.

**Next Steps**

* Review these other SQL Server [tuning tips](https://www.mssqltips.com/category.asp?catid=9)
* Check out [www.sqlworkshops.com](http://www.sqlworkshops.com/) for webcasts related to performance tuning.

Last Update: 2010-06-18   
  
[**View all my tips**](https://www.mssqltips.com/sqlserverauthor/42/matteo-lorini/)

|  |  |
| --- | --- |
| **Monday, June 13, 2016 - 6:34:32 AM - Erez Ben Simon** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| Great tip!    Thanks a lot!  Erez | |

|  |  |
| --- | --- |
| **Tuesday, June 18, 2013 - 9:29:04 AM - Sudhakar** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| Thanks for sharing this wonderfull article. This article helped me to emerge the reason for high utilization. | |

|  |  |
| --- | --- |
| **Tuesday, June 11, 2013 - 6:08:51 PM - Srinath** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| Excellent post !! | |

|  |  |
| --- | --- |
| **Friday, December 16, 2011 - 9:02:33 AM - Matteo** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| Question: when you see CXPACKET acting up on production server, what do you do?  You will have to review your MAX DOP setting, please check this post <http://blogs.msdn.com/b/cindygross/archive/2011/01/28/the-ins-and-outs-of-maxdop.aspx>; and take a closer look to your queries. | |

|  |  |
| --- | --- |
| **Wednesday, December 14, 2011 - 6:29:05 PM - Jason** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| Bill, Cranfield, when you see CXPACKET acting up on production server, what do you do? | |

|  |  |
| --- | --- |
| **Wednesday, September 01, 2010 - 1:20:04 PM - Bill** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| Good stuff, thanks for the article. | |

|  |  |
| --- | --- |
| **Friday, June 18, 2010 - 4:52:51 AM - --cranfield** | [Back To Top](https://www.mssqltips.com/sqlservertip/2027/a-closer-look-at-cxpacket-wait-type-in-sql-server/#top) |
| this is is a great tip. Thanks for posting.  I have learned something new today. | |