Tuning the Performance of Change Data Capture in SQL Server 2008

SQL Server Best Practices Article

Writer: Steffen Krause

Contributors: Sanjay Mishra, Gopal Ashok, Greg Yvkoff, Rui Wang

Technical Reviewers: Burzin Patel, Denny Lee, Glenn Berry (SQL Server MVP), Joseph Sack, Lindsey Allen, Michael Redman, Mike Ruthruff, Paul S. Randal (SQLskills.com)

Published: November 2008

Applies to: SQL Server 2008

Summary: Change data capture is a new feature in SQL Server 2008 that provides an easy way to capture changes to data in a set of database tables so these changes can be transferred to a second system like a data warehouse. This document provides guidance on how to configure change data capture parameters to maximize data capture performance while minimizing the performance impact on the production workload. The scope of this document is limited to the capture of change data and the cleanup process. Querying the changed data is out of scope for this white paper.

Introduction

Change data capture is a new feature in Microsoft® SQL Server® 2008 that provides an easy way to capture changes to data in a set of database tables so these changes can be transferred to a second system such as a data warehouse. This document provides guidance on how to configure change data capture parameters to maximize data capture performance while minimizing the performance impact on the production workload. The scope of this document is limited to the capture of change data and the cleanup process. Querying the changed data is out of scope for this white paper. For an introduction to the change data capture feature, see Change Data Capture [http://msdn.microsoft.com/en-us/library/bb522489.aspx] in SQL Server Books Online.

Notice that there is another new feature named change tracking in SQL Server 2008 that also enables tracking of table rows changed by DML commands (insert, update, delete, merge). Change tracking is aimed at the synchronization of loosely coupled database applications and tracks only the primary keys of changed rows and optionally which columns changed, but not the changed data itself. Change Tracking is also out of scope for this document. For more information about these two features, see Comparing Change Data Capture and Change Tracking [http://msdn.microsoft.com/en-us/library/cc280519.aspx] in SQL Server Books Online.

Benefits of Change Data Capture

Traditionally, detecting changes in a source database to transfer these changes to a data warehouse required either special columns in the source tables (time stamps, row versions), triggers that capture changes, or comparison of the source and the destination system. These methods can have significant disadvantages: special columns require a change in the source database schema and in many cases a change in the application logic. Triggers need to be implemented manually and can put significant additional overhead on DML commands to the source system. Comparing source and target databases can put a heavy load on both systems.

Change data capture enables the capture of changes in the source system by asynchronously reading the transaction log of the source database. For this, change data capture uses the same log reader that is used in transactional replication. Because change data capture works on existing table schemas, the source database or application doesn’t need to be changed to enable change data capture. Because the log reader job works asynchronously, DML transactions are far less impacted than with synchronous solutions like triggers. All changes to the source tables are recorded in special change tables, so no comparison between source and target system for changes is needed.

Basic Change Data Capture Method of Operation

When change data capture is enabled on a database, the schema “cdc” and a set of metadata tables in this schema are created. Most of these change data capture tables contain metadata and have very little data volume and transactional load. The only metadata table that is frequently written to during change data capture operation and that can grow to a significant size is the table **cdc.lsn_time_mapping**, which records the mapping between log sequence numbers (LSNs) and the date and time when the transaction happened.

When subsequently a table is enabled for change data capture, a capture instance is created. A capture instance consists of a change table (cdc.<capture_instance_name>_CT) and up to two table-valued functions for querying this change table. There can be up to two capture instances per table. If there is no log scan job for either transactional replication or another capture instance on the database, a capture job and a cleanup job are created in SQL Server Agent with default parameters (For more information about changing these parameters, see Configuration of the Capture Job). The capture job is automatically started.

After the capture job is enabled, it will write one row to the change table for every row inserted or deleted in the source table, and two rows for every row updated in the source table (one row containing the values before the update, one row containing the values after the update). The change table always contains the values for all captured columns in the source table, not only for columns that changed their value during the update. Therefore, data from only the change table is necessary to propagate the changes to a data warehouse. In addition to the data from the source table, the change table contains at least 37 bytes (or more, depending on the number of captured columns; for more information, see cdc.<capture_instance>_CT [http://msdn.microsoft.com/en-us/library/bb500305.aspx] in SQL Server Books Online) of additional data for each row of change data.

Configuration of sys.sp_cdc_enable_table Parameters

The stored procedure sys.sp_cdc_enable_table [http://msdn.microsoft.com/en-us/library/bb522475.aspx] has several parameters that we found important for the performance of change data capture. Parameters that are not relevant for performance are not discussed here. For more information about the influence of **sys.sp_cdc_enable_table** on the workloads we tested, see Influence of sys.sp_cdc_enable_table Parameters on Change Data Capture Performance.

The parameter *@capture_instance* determines the name of the capture instance for this table. The name itself has no impact on performance. However, this parameter can be used to create a second capture instance for the same table. The feature of having a second capture instance is only intended for schema upgrade scenarios where a change to the schema of the source table is performed. The old capture instance should be disabled as soon as it is no longer needed, because having two capture instances active on one table means that twice as much change data needs to be written. This can significantly impact performance.

When you have a schema change on your source table, you should do the following:

1. Change the schema of your source table.
2. Create a new capture instance. That new capture instance is created with the new (changed) schema of the source table.
3. Wait for a change to occur, and then read the minimum LSN from the new capture instance (using sys.fn_cdc_get_min_lsn(new capture instance name)).
4. Read and process all changes from the old capture instance up to but not including the first LSN from the new capture instance.
5. Disable the old capture instance.
6. Update all references to the old capture instance name with the new name.
7. Continue reading from the new capture instance.

The parameter *@captured_column_list*determines which columns of the source table are included in the change table and captured by change data capture. If this parameter is not specified or NULL, all columns of the source table are included in the change table. We found that the number and size of captured columns has a significant impact on change data capture performance and required disk space. Change data capture performance is generally better when the number of captured columns is smaller because the amount of data that needs to be written to the change tables is smaller.

The parameter *@supports_net_changes* determines whether net change queries to the change table (using cdc.fn_cdc_get_net_changes_<capture_instance>) are possible. Net change queries result in only one change row for each source row that was changed, independent of the number of changes. For instance, if stock market prices are tracked per stock and the price column is updated several times a day, an all changes query would return every change of every stock price in the queried LSN interval. A net change query would result in only one change row per stock with the final price in the data.

When *@supports_net_changes* is set to 1, an additional nonclustered index is created on the change table and the net changes query function is created. Because this index needs to be maintained, we found that enabling net changes can have negative impact on change data capture performance.

The parameter *@filegroup_name* determines the name of the filegroup where the change tables are created. If this parameter is not specified, the default filegroup will be used. Because in most cases the default filegroup is PRIMARY and it is a best practice to keep the PRIMARY filegroup small, a filegroup name should always be specified.

Configuration of the Capture Job

When change data capture is enabled on a database that is already configured for transactional replication, the replication capture job is also used by change data capture. For this reason, the discussion in this topic does not apply for databases that have both transactional replication and change data capture enabled.

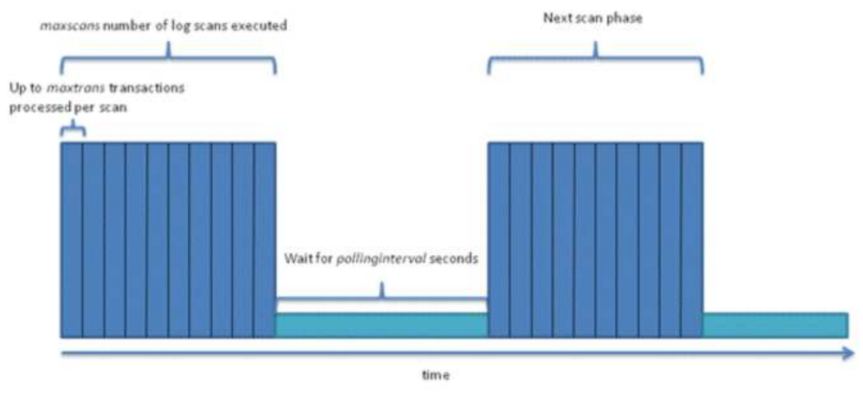
The capture job created in SQL Server Agent for change data capture (typically named cdc.database name_capture) contains only a call to the procedure **sys.sp_MScdc_capture_job** without parameters. This procedure determines the scan job parameters and calls **sys.sp_cdc_scan** with these parameters. The procedure **sys.sp_cdc_scan** does the actual work by scanning the log (using **sys.sp_replcmds**) and inserting the data that needs to be captured into the change tables.

There are four parameters in **sys.sp_cdc_scan** that determine the behavior of the capture job.

The first parameter is continuous (default value 1). It determines whether the capture job runs continuously (value 1) or exits after one scan phase (one shot mode, value 0). One shot mode is recommended for testing only, not for production use. The main reason for this is that the log records stay active until change data capture has processed them. So, the log will keep growing when the scan job is not running.

The other three parameters determine how often and how many transactions are read from the log and inserted to the change tables.

The parameter *maxtrans* (default value 500) determines how many transactions are read from the log and written to the change tables. This write is done in one transaction. The parameter *maxscans* (the default value is 10) determines how many of these scan cycles are attempted before the job is ended (continuous = 0) or before pausing for an interval (continuous=1). The length of this pause interval is set by the parameter *pollinginterval* (in seconds, with a default value of 5 seconds). WAITFOR is executed when a scan cycle drains the log completely or when *maxscans* scan cycles are completed.



maxscans number of log scans executed
Next scan phase
Up to maxtrans transactions processed per scan
Wait for pollinginterval seconds
time

Figure 1: Influence of capture job parameters

The scan job parameters can be changed by using **sys.sp_cdc_change_job**. Because these parameters are only read during the initialization of the capture job, the capture job needs to be stopped using EXEC sys.sp_cdc_stop_job @job_type = 'capture' and subsequently restarted using EXEC sys.sp_cdc_start_job @job_type = 'capture' so the changed parameters are applied.

Test Workload and Test Environment

To test change data capture performance and influence on workloads, we tested against an ISV application workload. The tests were done on a HP DL 580 (4 dual-core, 32 GB RAM, x64) connected to a HP EVA SAN with separate disk volumes for data, log, backup and change data capture filegroups. The application server was a HP BL460 (2 quad-core, 32 GB RAM, 32bit) connected over Gigabit Ethernet. For more information about hardware and software environment used in the test, see Appendix A: Test Hardware and Software.

In the application's main database, seven tables with a total of 603 columns were enabled for change data capture. It should be noted that with this workload, change data capture was used to capture transaction data (such as sales, shipment, audit data), not master reference data (such as customer, product, sites data). Many customers will use change data capture only to capture changes to master reference data, not to transaction data. This would result in significantly lower transaction rates that need to be captured by change data capture than with this workload.

The application comprised of a 'batch' type workload with many discrete phases. With the exception of phase 3 (see Table 1), nearly all transactions to the application database wrote to the tables that were tracked by change data capture.

Phase	**Duration (min)**	**Average application database transactions per second (tps)**	**Average tempdb transactions per second (tps)**	**Average CPU usage % (Database server)**
Begin	A few seconds	2 big transactions (200,000 and 20,000 rows updated)	Not applicable	Not applicable
1	7	5	780	6
2	10	10, spikes up to 200	40	13
3	17	3,000 gradually decreasing to 1800. Of these, about 1,000 writes to tables are covered by change data capture	5,000 decreasing to 3,200	60
4	3	1,800 – 2,500	1,000	37
5	7	850	2,000	62

6	6	3,000 – 4,000	12,000 - 1,200	45
7	3	Varying, spikes up to 14,500	Varying, low to medium	Varying, low to medium

Table 1: Application load behavior

In addition to the application workload, we also wanted to test synthetic workloads (simpler, artificially scaled workloads for testing specific types of DML at constant levels) that push change data capture to its limits and determine the behavior of change data capture with different parameter sets under extremely high load. Each of these workloads was designed to produce a load as high as possible on our test system. All workloads ran from a test client with 100 parallel threads and ran for 500 seconds on the same database that the application workload used.

Specifics of synthetic workloads:

1. Insert test 1: Insert into a wide table (82 columns) with 8 columns filled and 390,000 rows prefilled in the table, one row inserted per transaction.
2. Insert test 2: Insert into a narrower table (21 columns) with 10 columns filled and 666,000 rows prefilled in the table, one row inserted per transaction.
3. Small update test: Update 3 columns in the 82 column table, one row per transaction.
4. Insert with trigger: The same insert as in test 1. The only difference is that the INSERT statement fires a trigger that updates one column of the same table after the insert. This results in three rows in the change table for every insert (one insert row, one row that contains the values as they exist before the update, and one row that contains the values as they exist after the update).
5. Mix of insert, update, and select statements.
6. Large update: Updates large numbers of rows (random between 10 and 2,000 rows) per transaction.
7. Large insert: 1,000 rows per transaction inserted into a wide table (82 columns) with 8 columns filled and 390,000 rows prefilled in the table. This workload ran for 300 seconds.

Determining Performance and Characteristics of Change Data Capture

To determine the performance characteristics of change data capture, three main questions needed to be answered:

- What is the performance of change data capture itself?
- How does enabling change data capture influence the performance of the original workload?
- Which parts of the system (CPU, I/O, and so on) are impacted by change data capture at what amount?

The **performance of change data capture** itself is determined by the difference between the time when the original transaction happened in the database and the time the change record appears in the change table. This time is called *latency*. On a system where change data capture can fully keep up with the workload, latency should not be significantly higher than the polling interval, and it should not increase over time.

The latency can be determined using the dynamic management view sys.dm_cdc_log_scan_sessions [http://msdn.microsoft.com/en-us/library/bb510694.aspx] . For this white paper, this dynamic management view was used to determine how long the scan sessions took, how many commands and transactions were processed, and how big the latency is. The sys.dm_cdc_log_scan_sessions dynamic management view keeps track of only the last few scan sessions. Because we wanted to monitor the buildup of latency over the full runtime of the workload, we captured the extended event sqlserver.cdc_session to a file instead of querying sys.dm_cdc_log_scan_sessions.

Extended Events is a new, high-performance event capturing mechanism in SQL Server 2008 that allows you to register targets (in this case, a file) with events (in this case, the sqlserver.cdc_session event). Whenever the event fires (in this case, for each log scan session) the event data is written to the target. A big advantage of Extended Events is that they incur a very low overhead on the server.

The extended event sqlserver.cdc_session contains the same information as the sys.dm_cdc_log_scan_sessions dynamic management view, but it enabled us to record every scan session that happened during the workload.

To capture the event, you first have to create an event session that determines the event(s) to be captured and the target that the event will be written to:

```
CREATE EVENT SESSION cdc_session ON SERVER

ADD EVENT sqlserver.cdc_session

ADD TARGET package0.asynchronous_file_target

(SET filename='c:\cdc_session.xel',

 metadatafile='c:\cdc_session.xem', max_file_size=10)
```

Next, before each run of the workload, enable the event session:

```
ALTER EVENT SESSION cdc_session ON SERVER STATE = start
```

To record the time it takes for change data capture to catch up with all transactions of a given workload and thus to find the maximum latency, we also needed a way to find out whether change data capture was finished reading the log after the workload finished. This can be determined by querying the dynamic management view periodically (in our test, every minute) to see whether the last scan was an empty scan (that is, a scan that didn't find any log records that needed to be inserted into the change tables). The dynamic management view does not list empty scans individually; instead it lists them as one row with the column **empty_scan_count** filled with the number of empty scans. In the case of our workloads, change data capture was finished catching up as soon as the last scan had a nonzero empty scan count:

```
SELECT empty_scan_count FROM appdb.sys.dm_cdc_log_scan_sessionsWHERE start_time  =(selectMAX(start_time)from appdb.sys.dm_cdc_log_scan_sessions)
```

After both the workload and change data capture finish, the event session is disabled and the results are written to a table. Because extended events are recorded in XML format, the result XML needs to be parsed:

```
ALTER EVENT SESSION cdc_session ON SERVER STATE=stop

SELECT

CAST (event_data asxml).value('(/event/data[@name="start_time"]/text/text())[1]','datetime2(2)') start_time,

CAST (event_data asxml).value('(/event/data[@name="end_time"]/text/text())[1]','datetime2(2)') end_time,

CAST (event_data asxml).value('(/event/data[@name="last_commit_cdc_time"]/text/text())[1]','datetime2(2)') last_commit_cdc_time,

CAST (event_data asxml).value('(/event/data[@name="duration"]/value/text())[1]','int') duration,

CAST (event_data asxml).value('(/event/data[@name="tran_count"]/value/text())[1]','int') tran_count,

CAST (event_data asxml).value('(/event/data[@name="command_count"]/value/text())[1]','int') command_count,

CAST (event_data asxml).value('(/event/data[@name="latency"]/value/text())[1]','int') latency

FROM sys.fn_xe_file_target_read_file

('C:\cdc_session_*.xel','C:\cdc_session_*.xem',null,null)
```

This capture results in a complete table of all log scan sessions that happened during a workload. It also contains the maximum latency. We also calculated the maximum throughput from these numbers. (For throughput, we measured commands per seconds, not transactions per seconds. To calculate change data capture throughput, we only considered the time period when change data capture ran under full load. This is the period where latency was > 5 seconds. We calculated the number of commands processed and divided it by the time from the start of the first scan that had a latency > 5s to the end of the last scan. For instance, if change data capture finished exactly 1000 seconds after it reached a significant latency and processed 1,000,000 commands in that time, the throughput would be 1,000 commands per second. This number is a good indicator on how many commands change data capture can keep up with a given workload.)

To determine the **influence of change data capture on the workload performance**, we simply used the run time of the workload for the ISV application workload (which is recorded by the application itself in a table). For the synthetic workloads that have a fixed run time of 500 seconds, we recorded the number of completed transactions in the workload.

To determine the **additional load change data capture incurs on the system**, we recorded a number of performance counters in the Windows® operating system and SQL Server dynamic management views. The following performance counters proved useful for measuring the influence of change data capture on the system:

- Logical Disk: Average Disk Queue Length for Data Disk (M:), Log Disk (L:) and CDC Filegroup Disk (P:)
- Databases: Transactions/second for the application database and for **tempdb**
- Processor: % Processor time (Total)

To gather data about how big the additional load on data and log files was ,we used the dynamic management view sys.dm_io_virtual_file_stats. For information about whether change data capture incurs additional waits to the system, we used the dynamic management view sys.dm_os_wait_stats, which was reset before each workload run by using the command DBCC SQLPERF("sys.dm_os_wait_stats" , CLEAR). We also restored the application database and used DBCC FREEPROCCACHE and DBCC DROPCLEANBUFFERS to start each test run with the same baseline (by ensuring all caches were cold).

Considerations for Change Data Capture Performance

Influence of Scan Job Parameters on Change Data Capture Performance

As explained in Configuration of the Capture Job, the capture job with default parameters (*maxscans*=10, *maxtrans*=500, and *pollinginterval*=5) cannot, even in theory, process more than 1,000 transactions per second, on average.

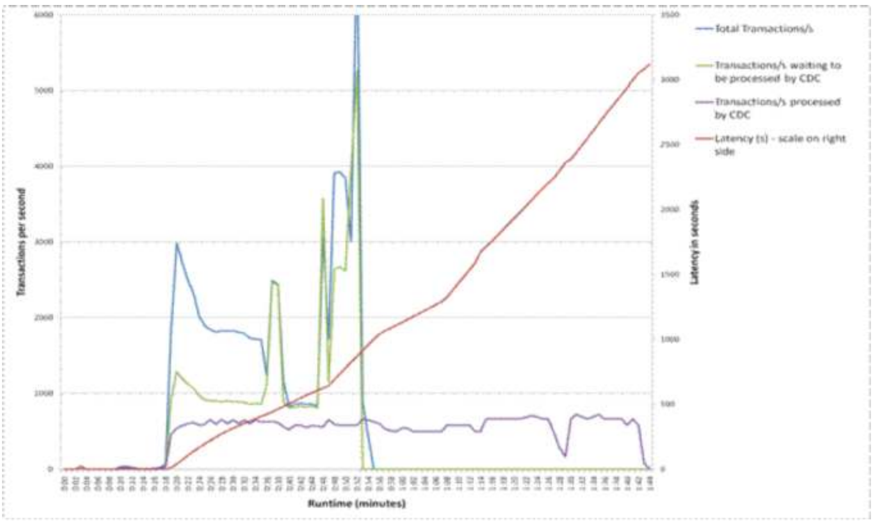


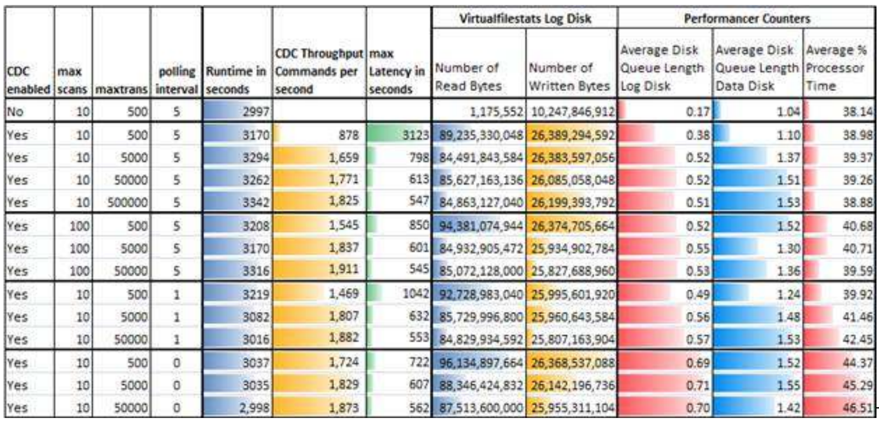
Figure 2: Latency buildup with default scan job parameters (*maxscans*=10, *maxtrans*=500, and *pollinginterval*=5)

In practice, we found that with the ISV application workload about 670 transactions per second were processed, with drops when very big transactions needed to be processed. As a result, the total latency until all transactions were processed by change data capture was 3123 seconds. The latency started to build up as soon as the workload reached more than 670 transactions per second, as shown in Figure 2.

To increase the number of transactions per second change data capture can handle and to reduce latency you can do one of the following three things:

- Increase the number of transactions per scan by increasing .
- Increase the number of scans before a pause by increasing .
- Decrease the pause between scanning cycles by decreasing .

Table 2 shows the results achieved by changing these parameters.



							Virtualfilestats Log Disk		Performancer Counters		
CDC enabled	max scans	maxtrans	polling interval	Runtime in seconds	CDC Throughput Commands per second	max Latency in seconds	Number of Read Bytes	Number of Written Bytes	Average Disk Queue Length Log Disk	Average Disk Queue Length Data Disk	Average % Processor Time
No	10	500	5	2997			1,175,552	10,247,846,912	0.17	1.04	38.14
Yes	10	500	5	3170	878	3123	89,235,330,048	26,389,294,592	0.38	1.10	38.98
Yes	10	5000	5	3294	1,659	798	84,491,843,584	26,383,597,056	0.52	1.37	39.37
Yes	10	50000	5	3262	1,771	613	85,627,163,136	26,085,058,048	0.52	1.51	39.26
Yes	10	500000	5	3342	1,825	547	84,863,127,040	26,199,393,792	0.51	1.53	38.88
Yes	100	500	5	3208	1,545	850	94,381,074,944	26,374,705,664	0.52	1.52	40.68
Yes	100	5000	5	3170	1,837	601	84,932,905,472	25,934,902,784	0.55	1.30	40.71
Yes	100	50000	5	3316	1,911	545	85,072,128,000	25,827,688,960	0.53	1.36	39.59
Yes	10	500	1	3219	1,469	1042	92,728,983,040	25,995,601,920	0.49	1.24	39.92
Yes	10	5000	1	3082	1,807	632	85,729,996,800	25,960,643,584	0.56	1.48	41.46
Yes	10	50000	1	3016	1,882	553	84,829,934,592	25,807,163,904	0.57	1.53	42.45
Yes	10	500	0	3037	1,724	722	96,134,897,664	26,368,537,088	0.69	1.52	44.37
Yes	10	5000	0	3035	1,829	607	88,346,424,832	26,142,196,736	0.71	1.55	45.29
Yes	10	50000	0	2,998	1,873	562	87,513,600,000	25,955,311,104	0.70	1.42	46.51

Table 2: Change data capture performance with different scan job parameters (ISV workload)

All results shown in Table 2 were taken using the ISV application workload with all 603 columns in seven tables captured by change data capture, net changes disabled, and change tables on the same filegroup as the application data. We also measured with change tables on a different filegroup on a different LUN and found no significant difference with this workload and hardware.

By only modifying scan job parameters, we were able to reduce the latency from above 52 minutes to about 9 minutes (first green column) and the throughput to 1,900 commands per second (dark yellow column). The influence on workload run time was below 10% (first blue column). Notice that workload run time without change data capture already has differences of ±5%. This can also explain the unexpected good workload run time results in the tests with a low polling interval, which did not exist in a second test run.

Figure 3 shows Transactions/s and latency for a test with *maxtrans*=5000, *maxscans*=100, and *pollinginterval*=5. Notice that both the time axis (bottom) and the latency axis (right) have a different scale than in Figure 2. With these changed parameters, the latency is much smaller, and latency does not build up until after 35 minutes of workload run time, because change data capture can now keep up with higher transaction loads.

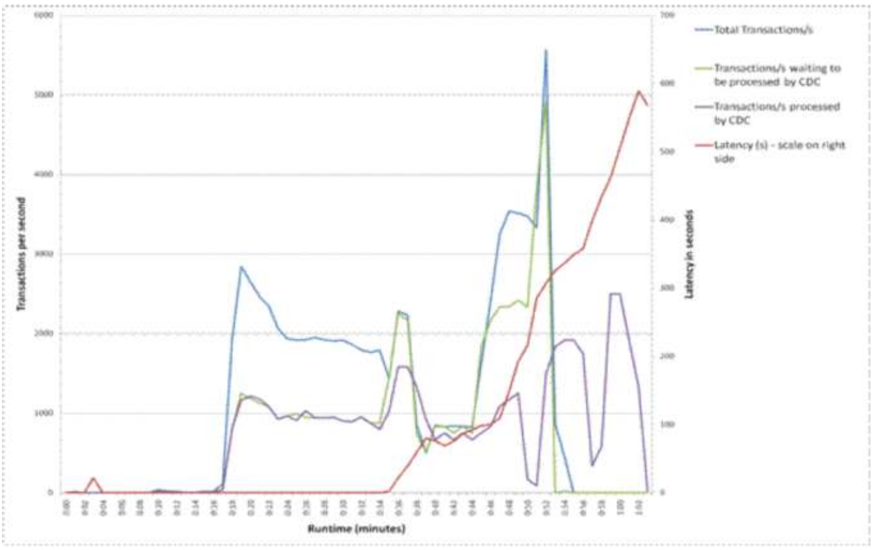


Figure 3: Latency buildup with changed scan job parameters (*maxscans*=100, *maxtrans*=5000, and *pollinginterval*=5)

Our results show that change data capture has a very low impact (below 10% change in workload run time) on the application workload and can therefore be used without negatively impacting application performance on a system with sufficient I/O capacity. On systems with a less optimal I/O subsystem, especially for the log disk, results can be different. In our tests, average disk queue length for the log disk tripled for most tests, the number of log bytes written grew to 250% of normal, and the log disk had significant read activity with change data capture. Other disk-related counters not shown in the table also grew significantly, such as stall read and write time and data disk writes.

In most test cases, change data capture incurred only a very small increase in processor usage. The only exceptions were the tests where *pollinginterval* was reduced to 0 (so no waits between scan cycles happened).

The tests prove that even a very big increase of *maxtrans* has no a significant negative impact on application performance. We also found that from a performance perspective, there was little difference between increasing *maxscans* or *maxtrans* and decreasing *pollinginterval*. Other test series on the application workload showed a slight advantage in increasing *maxtrans* over *maxscans*. But setting *maxtrans* very high leads to long scan cycle durations (we measured scan cycles up to 390 seconds for *maxtrans* = 500,000) and huge numbers of changed rows written to the change table in one transaction, which may hold a large number of locks and thus negatively impact reads from the change tables.

On the synthetic workloads, the workload 2 (small inserts to narrower table) showed that setting *maxtrans* too high (50,000 and 500,000) or setting *pollinginterval* too low (0) can decrease performance (lower transactions/s for the workload, and higher latency for change data capture).

CDC enabled	maxscans	maxtrans	pollinginterval	Transactions per second	Latency (sec)
No	10	500	5	3,367	
Yes	10	500	5	3,190	1,582
Yes	10	50000	5	2,588	86
Yes	10	500000	5	2,402	155
Yes	100	5000	5	2,783	5
Yes	10	5000	1	2,753	2
Yes	10	500	0	2,781	42

Table 3: Change data capture performance for synthetic workload 2

The other synthetic workloads didn't show similar behavior.

Recommendation: If change data capture with default parameters cannot keep up with the workload and latency becomes too high, you can increase *maxscans* and/or *maxtrans* by a factor of 10, or you can reduce *pollinginterval* to 1. If your latency decreases but is still too high, you can further increase *maxtrans*, but monitor the performance of your workload, latency and performance of queries to the change tables closely.

Influence of Workload Characteristics on Change Data Capture Performance

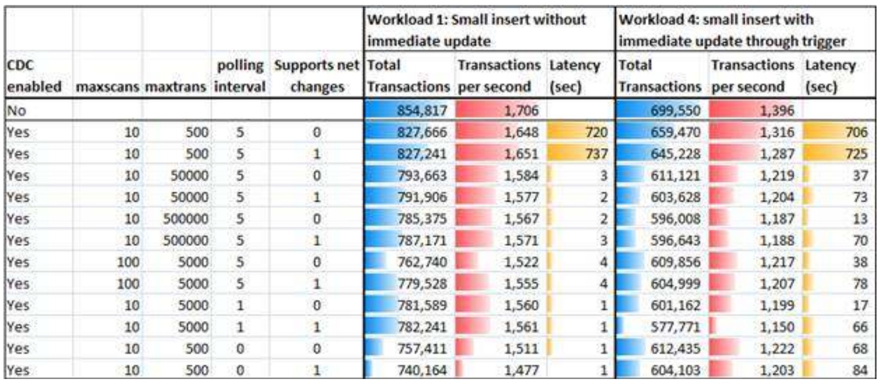
For planning a change data capture solution, workloads characteristics are very important. The main factors to consider are INSERT/DELETE vs. UPDATE, and whether the DML operations impact one row per transaction vs. many. To compare INSERT and UPDATE operations, we compared workload 1, which inserts one row per transaction into a table, with workload 3, which updates one row per transaction in the same table. For more information about these workloads, see Test Workload and Test Environment.

					Workload 1: Small insert			Workload 3: small update		
CDC enabled	maxscans	maxtrans	polling interval	Supports net changes	Total Transactions	Transactions per second	Latency (sec)	Total Transactions	Transactions per second	Latency (sec)
No					854,817	1,706		4,557,067	9,095	
Yes	10	500	5	0	827,666	1,648	720	4,086,608	8,156	5,689
Yes	10	500	5	1	827,241	1,651	737	4,033,443	8,050	5,789
Yes	10	50000	5	0	793,663	1,584	3	3,907,219	7,798	1,416
Yes	10	50000	5	1	791,906	1,577	2	3,907,787	7,799	1,581
Yes	10	500000	5	0	785,375	1,567	2	3,868,187	7,720	1,406
Yes	10	500000	5	1	787,171	1,571	3	3,866,295	7,717	1,575
Yes	100	5000	5	0	762,740	1,522	4	3,904,773	7,793	1,458
Yes	100	5000	5	1	779,528	1,555	4	3,863,317	7,711	1,609
Yes	10	5000	1	0	781,589	1,560	1	3,933,299	7,850	1,503
Yes	10	5000	1	1	782,241	1,561	1	3,832,475	7,634	1,518
Yes	10	500	0	0	757,411	1,511	1	3,823,023	7,630	1,478
Yes	10	500	0	1	740,164	1,477	1	3,784,376	7,553	1,621

Table 4: Comparison between INSERT and UPDATE transactions

SQL Server is able to maintain a much higher rate of UPDATE transactions than INSERT transactions (about 5x transactions per second). For change data capture, one INSERT only creates one row in the change table. One UPDATE, however, inserts two rows in the change table. In total this means that the UPDATE workload has to insert about 10 times the number of rows into the change table that the INSERT workload does, which leads to the significant latency that shows in Table 3. Workload performance (number of transactions in 500 seconds) reduction with change data capture was between 3% and 14% for the INSERT workload and between 10% and 17% for the UPDATE workload. This is a very low overhead for the amount of work that needs to be done because of change data capture.

A common scenario in applications is that a row is inserted to a table and the immediately updated to fill missing fields. This can happen in the application or by utilizing an INSERT trigger that looks for missing fields and then updates them if necessary. We tested change data capture in this scenario. The table that workloads 1 and 4 insert into has an INSERT trigger that checks if one column is filled and updates it with another column's value if it is NULL.



					Workload 1: Small insert without immediate update			Workload 4: small insert with immediate update through trigger		
CDC enabled	maxscans	maxtrans	polling interval	Supports net changes	Total Transactions	Transactions per second	Latency (sec)	Total Transactions	Transactions per second	Latency (sec)
No					854,817	1,706		699,550	1,396	
Yes	10	500	5	0	827,666	1,648	720	659,470	1,316	706
Yes	10	500	5	1	827,241	1,651	737	645,228	1,287	725
Yes	10	50000	5	0	793,663	1,584	3	611,121	1,219	37
Yes	10	50000	5	1	791,906	1,577	2	603,628	1,204	73
Yes	10	500000	5	0	785,375	1,567	2	596,008	1,187	13
Yes	10	500000	5	1	787,171	1,571	3	596,643	1,188	70
Yes	100	5000	5	0	762,740	1,522	4	609,856	1,217	38
Yes	100	5000	5	1	779,528	1,555	4	604,999	1,207	78
Yes	10	5000	1	0	781,589	1,560	1	601,162	1,199	17
Yes	10	5000	1	1	782,241	1,561	1	577,771	1,150	66
Yes	10	500	0	0	757,411	1,511	1	612,435	1,222	68
Yes	10	500	0	1	740,164	1,477	1	604,103	1,203	84

Table 5: Change data capture performance when inserted rows are immediately updated

In synthetic workload 1, we filled this column in the INSERT command so no update would happen in the trigger. In synthetic workload 4, the column value was not filled, so the trigger updated the row immediately after insert (within the same transaction). This resulted in three rows instead of one row that needed to be written to the change table for each insert to the original table. The results show that both workload and change data capture performance are better when no update happens immediately after insert.

Recommendation: Try to avoid scenarios where a row needs to be updated immediately after insert.

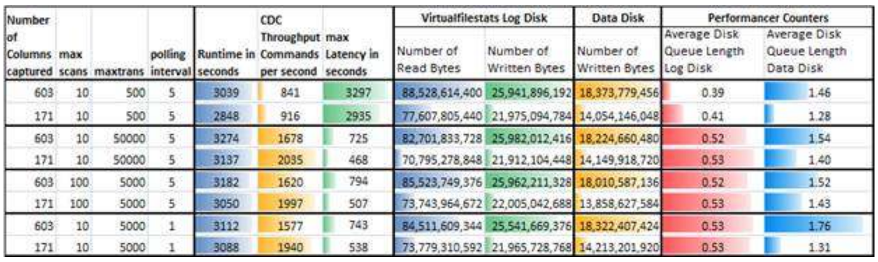
Workload 6 (large updates, 10-2,000 rows updated per transaction) showed no significant difference in performance or latency independent of change data capture parameters. The reason for this behavior is that with small amounts of big update transactions, change data capture has to insert two rows to the change table for every row that was updated in the source table. So with this workload, one transaction leads on average to 2,000 inserts to the change table. The time this large insert takes is so high that the scan job parameters do not have a significant impact any more. Even with default parameters, change data capture has to insert 1 million rows to the change table in one scan cycle. The latency here was only determined by the amount of data that needed to be inserted to the change table in total. Latency grew up to 13,000 seconds (with all columns captured) or 7,000 seconds (with reduced number of columns captured) for an 800-second workload that updated about 18 million rows.

Recommendation: Try to avoid using change data capture to capture changes to tables that have frequent large update transactions.

Workload 7 (large inserts, 1,000 rows inserted per transaction, about 1.7 million rows inserted in 300 seconds) also showed no significant difference in performance or latency independent of change data capture parameters. The latency was at about 250-399 seconds, depending on net change support, the number of columns captures with change data capture, and scan job parameters. This means that change data capture took about the same time to insert the rows into the change table after the workload finished as the original workload took.

Influence of sys.sp_cdc_enable_table Parameters on Change Data Capture Performance

The application workload has change data capture enabled on seven tables with 603 columns total. When studying the change tables we found that only 171 columns had more than one distinct value. The other columns where either NULL or had always the same value. In a second test run, we decided to capture only the 171 columns that had different values. We did this by specifying the *@captured_column_list* parameter in **sys.sp_cdc_enable_table**. We found that the decreased number of columns captured made a huge difference in overall change data capture performance, especially when specifying nondefault scan job parameters.



							Virtualfilestats Log Disk		Data Disk	Performancer Counters	
Number of Columns captured	max scans	maxtrans	polling interval	Runtime in seconds	CDC Throughput Commands per second	max Latency in seconds	Number of Read Bytes	Number of Written Bytes	Number of Written Bytes	Average Disk Queue Length Log Disk	Average Disk Queue Length Data Disk
603	10	500	5	3039	841	3297	88,528,614,400	25,941,896,192	18,373,779,456	0.39	1.46
171	10	500	5	2848	916	2935	77,607,805,440	21,975,094,784	14,054,146,048	0.41	1.28
603	10	50000	5	3274	1678	725	82,701,833,728	25,982,012,416	18,224,660,480	0.52	1.54
171	10	50000	5	3137	2035	468	70,795,278,848	21,912,104,448	14,149,918,720	0.53	1.40
603	100	5000	5	3182	1620	794	85,523,749,376	25,962,211,328	18,010,587,136	0.52	1.52
171	100	5000	5	3050	1997	507	73,743,964,672	22,005,042,688	13,858,627,584	0.53	1.43
603	10	5000	1	3112	1577	743	84,511,609,344	25,541,669,376	18,322,407,424	0.53	1.76
171	10	5000	1	3088	1940	538	73,779,310,592	21,965,728,768	14,213,201,920	0.53	1.31

Table 6: Change data capture performance in dependency of data amount captured

By reducing the number of columns captured to the ones that have relevant insertions or updates, we were able to reduce latency to below 8 minutes for the overall workload and at the same time reduce the workload runtime and reduce the load on the I/O subsystem incurred through change data capture. For the synthetic workloads, we saw similar improvements in Transactions/s and reductions in latency when the number of captured columns was smaller.

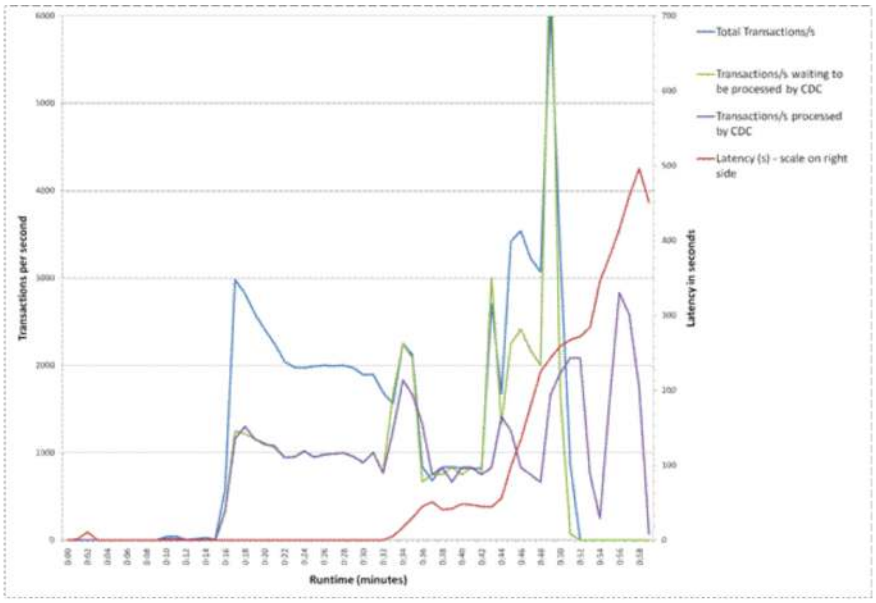


Figure 4: Latency buildup with reduced number of columns captured (*maxscans*=100, *maxtrans*=5000, and *pollinginterval*=5)

Figure 4, which uses the same scan job parameters as Figure 3, shows that with a reduced number of columns captured, not only is the latency lower and the performance is better, but the latency also shrinks between minute 36 and minute 43.

Recommendation: Always limit the list of columns captured by change data capture to only the columns you really need to track by specifying the *@captured_column_list* parameter in **sys.sp_cdc_enable_table**.

The @*supports_net_changes* parameter can have significant influence on change data capture performance. Especially if change data capture is just able to keep up with a workload, the additional load that is incurred by maintaining the additional index to support net changes queries can be enough to prevent change data capture from keeping up with the workload. The following table shows performance with and without supporting net changes queries for the synthetic workload 4 (Insert with trigger that updates the row immediately after insert).

maxscans	maxtrans	polling interval	Supports net changes	Total Transactions	Transactions per second	Latency (sec)
10	500	5	0	659,470	1,316	706
10	500	5	1	645,228	1,287	725
10	50000	5	0	611,121	1,219	37
10	50000	5	1	603,628	1,204	73
10	500000	5	0	596,008	1,187	13
10	500000	5	1	596,643	1,188	70
100	5000	5	0	609,856	1,217	38
100	5000	5	1	604,999	1,207	78
10	5000	1	0	601,162	1,199	17
10	5000	1	1	577,771	1,150	66
10	500	0	0	612,435	1,222	68
10	500	0	1	604,103	1,203	84

Table 7: Influence of the @*supports_net_changes* parameter

Recommendation: If you do not require support for net changes, set @*supports_net_changes* to 0. If you do require querying for net changes but change data capture latency grows too big, it can be worthwhile to turn support for net changes off and do the net change detection later in a staging database.

Cleanup Job Considerations

To test cleanup performance, cleanup was run manually after the application workload finished. In the test, one change table that contained 4,147,855 rows was cleaned completely with varying threshold parameters by running **sys.sp_cdc_cleanup_change_table**. The configurable threshold value limits how many entries are deleted in any single statement. The test was done after querying the change table (by running SELECT count(*) FROM change_table_name) to simulate the scenario when a change table is first read and then cleared.

When the cleanup job was run without additional workload, it showed that an increase of threshold up to 500,000 could improve overall cleanup performance. Improving it further to 5,000,000 (so the whole table would be cleaned up in one transaction) lowered the performance, as shown in Figure 4.

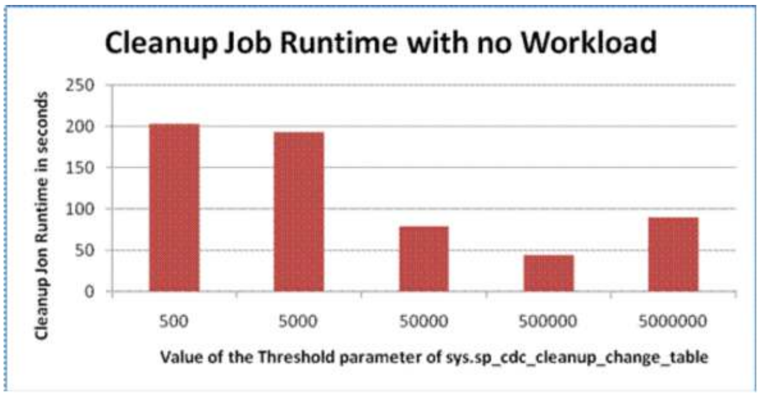


Figure 4: Cleanup job run time depending on threshold

When the cleanup job was run in parallel with a workload (workload 5: mix of insert, update, and select transactions), the time to run cleanup grew significantly. This test was done with the following change data capture job parameters: *maxtrans*=5000, *maxscans*=100, and *pollinginterval*=5.

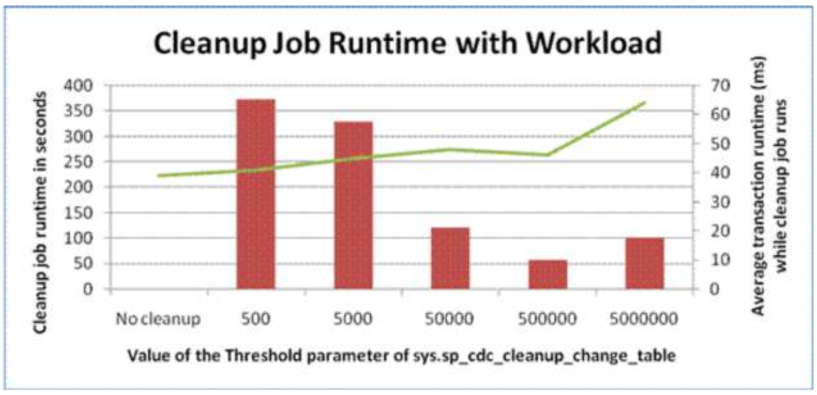


Figure 5: Cleanup job run time with concurrent workload

Again, a threshold of 500,000 is a sweet spot for this workload. Figure 5 also illustrates the average transaction run time while the cleanup job runs. Notice that especially with big thresholds, a lock escalation on change tables can happen. This could degrade the application response time, and it could increase the latency of the change data capture scan job.

Recommendation: If possible, run cleanup when there is no other workload active. Test increasing the threshold parameter until you find a sweet spot for your workload.

Transaction Log File Considerations

One of the most important things to watch out for with change data capture is the transaction log I/O subsystem. As stated earlier, log file I/O significantly grows when change data capture is enabled in a database. In addition to that, log records stay active until change data capture has processed them. This means that especially in environments where a large latency builds up, the log file can grow significantly because the log space cannot be reused as long as the change data capture scan job has not processed the log records, even in simple recovery model, or even after a log backup in full recovery model.

It should be noted that change data capture works with all recovery models. But when change data capture is enabled, operations that would normally be minimally logged in simple or bulk-logged recovery models are fully logged to enable change data capture to capture all changes.

Be aware also that when a log disk becomes full, you cannot shrink the log file by backing it up and manually shrinking it until change data capture has processed all transactions. But change data capture cannot process the transactions when the log disk is full, because change data capture writes to change tables are logged operations. In this case, the easiest way to recover from this situation is to temporarily add another log file on a different disk.

Recommendation: Whenplanning change data capture architecture, take a significant increase in log size and log volume I/O operations into account. Depending on the amount of data captured by change data capture and the time change data capture needs to catch up with changes, the log file size can grow to 200-300% of the original size, in some cases even more. Size the log file accordingly. Make sure that the growth in log file size does not result in a completely full log disk.

Filegroup Considerations

We found no difference in our tests between having the change table on the same filegroup as the application table and having them on different filegroups. This might be different when the data files are located on an I/O subsystem that is already under heavy load. In this case, putting change tables in a filegroup that is located on a different set of physical disks can improve change data capture performance.

Recommendation: To keep the PRIMARY filegroup small and to have a clear distinction between application data and change data, you should specify @*filegroup_name* in **sys.sp_cdc_enable_table**.

In addition to the change tables, the table **cdc.lsn_time_mapping** can also grow to a significant size and become a target of many I/O operations. This table is created on the default filegroup when **sys.sp_cdc_enble_db** is executed on a database.

Recommendation: Consider changing the default filegroup for the database before you execute **sys.sp_cdc_enble_db**, so that change data capture metadata and especially **cdc.lsn_time_mapping** are located on a different filegroup than PRIMARY. You can change the default filegroup back after the change data capture metadata tables are created.

Conclusion

Change data capture provides an easy and high-performing way to capture changes in a set of tables. There are many ways to tune the performance of change data capture. Understanding workload characteristics, system I/O usage, and allowable latency is key to tuning change data capture performance without negatively impacting the base workload. Tuning the scan job parameters, **sys.sp_cdc_enable_table** parameters, and, if possible, queries in the workload can significantly improve change data capture performance under load.

Here is a summary for the recommendations in this white paper.

Storage:

- Whenplanning change data capture architecture, take a significant increase in log size and log volume I/O operations into account.
- Consider specifying a filegroup in .
- Consider changing the default filegroup for the database before you execute so that change data capture metadata and especially are located on a different filegroup than PRIMARY.

Workload behavior:

- Try to avoid scenarios where a row needs to be updated immediately after insert.
- Try to avoid using change data capture to capture changes to tables that have frequent large update transactions.

Change data capture parameters:

- Always reduce the list of columns captured by change data capture to only the columns you really need to track.
- If you do not require support for net changes, set @ to 0.
- Use to see whether change data capture can keep up with your workload.
- If change data capture cannot keep up with your workload, modify scan job parameters and restart the scan job.

Cleanup:

- If possible, run cleanup when there is no other workload active.
- Test increasing the threshold parameter until you find a sweet spot for your workload.

Appendix A: Test Hardware and Software

Database Server

HP DL 580

- 4 socket dual core
- Intel Xeon 3,4 GHz
- 32 GB RAM
- Windows Server® 2003 Enterprise x64 Edition with Service Pack 2 (SP2)

Application Server (for an ISV Application)

HP BL 460

- 2 socket quad core
- Intel Xeon 2,83 GHz
- 32 GB RAM
- Windows Server 2003 32bit with SP2

Storage

HP EVA SAN with 2 disk groups:

- Disk group 1 with 152 disks 300 GB each @15,000 RPM, RAID1+0. Disk group 1 contained separate logical volumes for the following:
    - Database data files for the source tables
    - Database data files for the change tables
    - Data files for the database
    - Backups

- Disk group 2 with 88 disks 72 GB each @15,000 RPM, RAID1+0. Disk group 2 contained:
    - Database log file

Software

SQL Server 2008

For More Information:

http://www.microsoft.com/sqlserver/ [http://www.microsoft.com/sqlserver/default.aspx] : SQL Server Web site

http://technet.microsoft.com/en-us/sqlserver/ [http://technet.microsoft.com/en-us/sqlserver/default.aspx] : SQL Server TechCenter

http://msdn.microsoft.com/en-us/sqlserver/ [http://msdn.microsoft.com/en-us/sqlserver/default.aspx] : SQL Server DevCenter

http://www.sqlcat.com/ [http://www.sqlcat.com/default.aspx] : SQL Server customer advisory team

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback [mailto://microsoft.com:25
/default.aspx?subject=White%20Paper%20Feedback:%20Tuning%20the%20Performance%20of%20Change%20Data%20Capture%20in%20SQL%20Server%202008
] .