idera®

DEMYSTIFY TEMPDB PERFORMANCE AND MANAGEABILITY

BY ROBERT L DAVIS

Applies to: SQL Server 2008, SQL Server 2008 R2

SUMMARY

This whitepaper provides clear guidance on best practices for managing tempdb to provide the optimum balance between performance, reliability, and manageability.

INTRODUCTION

There are many misconceptions and myths about tempdb and purported best practices are inconsistent at best. It's hard to know which advice to follow when one resource says to always do it one way and another tells you to always do it the opposite way. Many times, both resources are correct in certain situations or to a certain degree. Part of the problem is that rarely is there a single correct solution to any scenario in SQL Server. Unfortunately, many administrators will assume that because a practice worked for them in one situation that they should do the same practice in every situation.

The purpose of this whitepaper is to guide you on how to make the right decisions for managing tempdb; to help you determine the proper solution for any given scenario. In order to be able to make these decisions, there are several concepts that must be understood. We will take a look at what tempdb is and how it is used and some common major problems and how to prevent them. We will also provide some best practices on configuring and monitoring tempdb.

idera®

ABOUT THE AUTHOR

Robert L Davis is a Sr. Product Consultant and Chief SQL Server Evangelist for Idera Software. Previously, he was the Program Manager for the SQL Server Certified Master Program in Microsoft Learning. He was also a Sr. Production DBA at Microsoft with more than 12 years' experience with SQL Server -- author of Pro SQL Server 2008 Mirroring -- writer for SQL Server Magazine -- Microsoft Certified Master: SQL Server 2008 -- Speaker/trainer.

Robert blogs at:
www.sqlsoldier.com

Robert uses the Twitter handle:
@sqlsoldier

WHAT IS TEMPDB

The cornerstone of making smart decisions is being educated on the subject. Any deep discussion of tempdb should begin with an explanation of what tempdb is and how it is used. Tempdb is like the catchall for the database engine. One analogy that works well is a well-organized junk drawer.

The database engine uses tempdb for all of the following:

» Temporary user objects
- Temp tables
- Table variables
- Temp procedures
- Global temp tables
- Cursors

» Work tables / work files / intermediate results
- Sorts
  - Index rebuilds
  - Some Group By, Order By, or Union operations
- Spools
- Hash joins and hash aggregates
- Spills to tempdb
- Temporary LOB storage
- Returned tables for table valued functions
- Service broker caching
- Common Table Expressions (CTEs)

» Version store for data modification transactions from
- Read committed snapshot isolation
- Snapshot isolation
- Online indexes
- After triggers
- Multiple Active Results Sets (MARS)

As you can see, tempdb is very much like a junk drawer. SQL has to keep track of everything in tempdb. It organizes the above objects into three groups: internal objects, external objects, and the version store. You can see how much total tempdb disk space is allocated to each category by querying the dynamic management view (DMV) sys.dm_db_file_space_usage.

```
SELECT FreePages =
          SUM(unallocated_extent_page_count),
     FreeSpaceMB =
          SUM(unallocated_extent_page_
     count)/128.0,
     VersionStorePages =
          SUM(version_store_reserved_page_count),
     VersionStoreMB =
          SUM(version_store_reserved_page_
     count)/128.0,
     InternalObjectPages =
          SUM(internal_object_reserved_page_
     count),
     InternalObjectsMB =
          SUM(internal_object_reserved_page_
     count)/128.0,
     UserObjectPages =
          SUM(user_object_reserved_page_count),
     UserObjectsMB =
          SUM(user_object_reserved_page_
     count)/128.0
FROM sys.dm_db_file_space_usage;
```

TEMPDB CONTENTION AND CONFIGURATION

One of the most daunting performance problems with tempdb is contention on the allocation pages, often referred to as just tempdb contention. The default settings for tempdb are not sufficient for active systems. Even a moderately busy system can easily overcome the allocations pages for tempdb.

Tempdb contention is often confused with general blocking. In fact, the way it is generally detected is because it causes blocked processes. In severe cases, it can create a massive blocking chain that grows faster than processes can be cleared out. This results in client connections timing out which leads to a massive chain of blocked processes attempting to roll back.

However, this is not just simple blocking. Tempdb contention is latch contention on the allocation pages. There are three types of pages that experience this issue:

Page Free Space (PFS): Tracks the allocation status of each page and approximately how much free space it has. There is one PFS page for every 1/2 GB of data file. The first PFS page is page number 1 of the data file. The second is page 8088 and then it repeats every 8088 pages thereafter.

Global Allocation Map (GAM): Tracks which extents have been allocated. There is one GAM page for every 4 GB of data file. The first GAM page is page number 2 in the data file, the second is page number 511232, and then it repeats every 511,232 pages.

Shared Global Allocation Map (SGAM): Tracks which extents are being used as mixed (shared) extents. There is one SGAM page for every 4 GB of data file. The first SGAM page is page number 3 in the data file, the second is page number 511233, and then it repeats every 511,232 pages.

You can use the dynamic management view (DMV) sys.dm_os_waiting_tasks to find tasks that are waiting on a resource. Tasks waiting on PageIOLatch or PageLatch wait types are experiencing contention. The resource description points to the page that is experiencing contention and you can easily parse the resource description to get the page number. Then it's just a math problem to determine if it is an allocation page.

The formula for determining the type of page experiencing contention is:

» PFS: Page ID = 1 or Page ID % 8088

» GAM: Page ID = 2 or Page ID % 511232

» SGAM: Page ID = 3 or (Page ID – 1) % 511232

The following query will parse the resource description and determine if the contention being experienced is on allocation pages or not:

```
WITH Tasks
AS (SELECT session_id,
        wait_type,
        wait_duration_ms,
        blocking_session_id,
        resource_description,
        PageID = CAST(Right(resource_description,
               LEN(resource_description) -
             CHARINDEX(':', resource_description,
3))
               AS INT)
    FROM sys.dm_os_waiting_tasks
    WHERE wait_type LIKE 'PAGE%LATCH_%'
    AND resource_description LIKE '2:%')
SELECT session_id,
        wait_type,
        wait_duration_ms,
        blocking_session_id,
        resource_description,
    ResourceType =
        CASE
        WHEN PageID = 1 Or PageID % 8088 = 0
             THEN 'Is PFS Page'
        WHEN PageID = 2 Or PageID % 511232 = 0
             THEN 'Is GAM Page'
        WHEN PageID = 3 Or (PageID - 1) %
 511232 = 0
             THEN 'Is SGAM Page'
        ELSE 'Is Not PFS, GAM, or SGAM page'
    END
FROM Tasks;
```

The key to avoiding this problem is proper configuration of tempdb. The proper number of data files for tempdb is a highly debated topic. Microsoft officially recommends one data file per logical CPU (1:1 ratio). Even with the most modern servers that have very high numbers of CPUs, this recommendation from Microsoft persists. This raises the question, "does tempdb really need one data file per CPU?"

Data File per CPU Ratio

As with most things in SQL Server, the answer is, "it depends." Some extremely busy servers with heavy tempdb usage may need one data file per CPU. This would qualify as one of those edge cases that somehow became the rule. Yes, it is possible that you may need one file per CPU but not likely.

There is a very good reason why Microsoft makes this recommendation and why you may want to follow it. There is a small performance impact to having a large number of data files. The more files you have, the greater the management overhead. But this overhead is minor compared to the impact of tempdb contention.

It is a very logical recommendation to start with a ratio of one data file for every two or four CPUs and adjust as needed. The "as needed" part is what can kill performance of your server. As needed means that tempdb contention is occurring. This can be a minor issue if you are monitoring for tempdb contention because you know what action to take to alleviate it and you are comfortable with experiencing a major performance impact when it occurs. If you cannot answer yes to all of those stipulations, you should follow the best practice of one data file per CPU. The best practice for ratio of data files per CPU can be summarized as below.

1. Create 1 data file per logical CPU

Or,

1. Start With 1 data file for every 2 or 4 logical CPUs
2. Monitor for tempdb contention
3. If tempdb contention detected, increase the number of data files
4. Repeat this process to a maximum of 1 data file per logical CPU

There will never be a need to have more than one data file per CPU. The actual number of data files that SQL Server can use is the number of concurrent processes using tempdb at the same time. All processes ultimately run one task at a time on each CPU thread and there is only one CPU thread per logical CPU. If every CPU is running a task and each of those tasks is using tempdb, the number of threads hitting tempdb will be the same as the number of logical CPUs. Your server will never have more processes than number of CPUs hitting tempdb.

Pre-size Files to Avoid Auto-growth

The number of data files for tempdb is only one best practice for tempdb that affects tempdb contention. You will often hear that tempdb writes to the data files in a round-robin method. In an ideal scenario, this description is sufficient. The algorithm that SQL uses to determine which file to use is actually more complex than that.

The process takes into consideration several factors. One of the main considerations is the amount of free space on the data file. SQL also implements a clock-hand type of counter that tracks when it skips over each of the data files and when it uses them. It increments or decrements a counter based on when it uses a particular data file.

Having equal amounts of free space is critical to maintaining the round-robin data flow. It is equally important that the data files are all the same size and maintain the same size. The data files should be pre-sized to avoid any auto-growth of the data files. If one data file is expanded, the newly grown file will have more free space than the other data files and SQL will start writing all data to this single file until it catches up with the other files in free space. At this point, SQL may need to grow the file again. If the files are not the same size, SQL will auto-grow the largest file. You will end up with one file continually growing while the others stay small. This can cause unexpected breakouts of tempdb contention even though you have multiple files.

You want to avoid growing the files if at all possible. You should pre-size the data files so that they are all the same size and large enough that there won't be any need to grow the data files. My recommendation would be to pre-size all of the files, data and log, to use approximately 90% of the available drive space and disable auto-growth on the data files. I recommend allowing the log file to auto-grow by a set size, such as 512 MB, just in case. Since there should only ever be a single log file, there is very little harm if it auto-grows.

I recommend setting the log file to be approximately double the size of a single data file. The formula I recommend for calculating the size of the tempdb files given a drive with a specific size is:

Data file = (Size of drive * 90%) / (Number of data files + 2 [log file will be double the data file])

Log file = Data file * 2

For example, if I have a 100 GB drive to use for tempdb, and I am creating 1 data file per CPU for an 8 CPU SQL Server, I will calculate the file sizes to be:

Data file = (100 GB * 90%) / (8 + 2) = 90 GB / 10 = 9 GB

Log file = 9 GB * 2 = 18 GB

The final calculation is 8 data files of 8 GB each and 1 log file of 18 GB leaving 10 GB free for possible log file growth.

There is another popular recommendation of implementing trace flag 1117 to force SQL Server to auto-grow all data files equally when expansion is needed to work-around this issue. The trace flag will work for tempdb in this situation but there is a major downside to using it. The trace flag affects every database on the server. Unlike tempdb, the files in a user database may not be used in the same round-robin fashion and this behavior may result in some files growing very large even though they have very little data in them.

Disk Performance and RAM

A lot of emphasis is placed on putting tempdb on the fastest disks possible. In my opinion, having plenty of RAM is even more important than disk performance.

Tempdb is being continually optimized in the engine to improve performance. One of the ways that tempdb performance has improved and continues to improve is its use of RAM. Tempdb will try to maintain objects in RAM as much as it can. There is a lot you can do to help tempdb stay in RAM.

First of all, you can help tempdb maintain objects in RAM by giving it plenty of RAM to use. If SQL is experiencing buffer pool memory pressure, SQL may trim some areas of the buffer pool to make more available to other areas such as the data cache or procedure cache.

The second thing you can do is to have a highly optimized system with current and meaningful statistics. The second very large performance hit on tempdb is when memory space for queries spill to tempdb. When compiling a query, the query optimizer will calculate the amount of memory it thinks the query will need for sort, spool, or hash operations. Queries will only use a contiguous block of memory for these operations. If the amount of memory granted to the query is insufficient, the process will spill to tempdb. It will never request additional memory as there is no way for it to ensure that the memory will be contiguous.

When a spill occurs, everything that has been written to memory will be flushed to disk. This operation is very disk intensive on tempdb and can cause major blocking or delays in processing of the query.

You can identify a currently executed spill to tempdb using the DMV sys.dm_os_waiting_tasks. The wait types most associated with a spill to tempdb are IO_COMPLETION and SLEEP_TASK. IO_COMPLETION literally

means what the name implies; it is waiting for a disk IO task to complete. SLEEP_TASK is a generic wait type that can be seen in many different scenarios, but in most scenarios other than a spill to tempdb, the wait is held for such a short time that you will rarely catch it in progress. If you see sustained SLEEP_TASK waits, it is very likely a spill to tempdb.

Spills to tempdb may be caused by statistics not being maintained, by parameter sniffing (which results in discrepancies in the number of estimated rows versus the actual number of rows), and by widely skewed data (making it difficult for the optimizer to estimate the number of rows). In my experience, index and column statistics are the leading contributors to this problem.

Some common tactics for preventing these problems include using a higher sample rate for statistics updates and updating statistics more frequently. For the last case where data is widely skewed, the problem may persist even with the best possible statistics. Statistics are limited to 200 entries to try to describe the entire dataset in the index or columns. The greater the number of distinct values that must be represented, the harder it is to represent them accurately. Sometimes you can help the optimizer by adding statistics for a column that is not indexed.

A known instance of data skew can be worked around by applying a filter to statistics. If you created a filtered index, the optimizer may or may not be able to use the filtered index. However, it would be able to take advantage of the statistics created to support the filtered index. Statistics are much lighter weight than indexes and if you only need the statistics for the optimizer to get accurate counts, then it makes sense to only create the statistics.

If spills are currently occurring, you can use the DMV sys.dm_exec_query_memory_grants to see how much memory the optimizer thought the query needed. While the query is still running, the ideal_memory_kb column will show how much physical memory the optimizer thought the query would need.

You can also see how much buffer pool space tempdb is using by querying the system catalogs sys.allocation_units and sys.partitions. The below query provides some raw details of the tempdb objects in the buffer pool.

```
USE tempdb;

SELECT ObjectName = OBJECT_NAME(object_id),
    object_id,
    index_id,
    allocation_unit_id,
    used_pages,
    AU.type_desc
FROM sys.allocation_units AS AU
INNER JOIN sys.partitions AS P
     -- Container is hobt for in row data
     -- and row overflow data
    ON AU.container_id = P.hobt_id
        -- IN_ROW_DATA and ROW_OVERFLOW_DATA
        AND AU.type In (1, 3)
UNION ALL
SELECT ObjectName = OBJECT_NAME(object_id),
    object_id,
    index_id,
    allocation_unit_id,
    used_pages,
    AU.type_desc
FROM sys.allocation_units AS AU
INNER JOIN sys.partitions AS P
     -- Container is partition for LOB data
```

```
    ON AU.container_id = P.partition_id
        -- LOB_DATA
        AND AU.type = 2
```

You can combine these results with the DMV sys.dm_os_buffer_descriptors to dig deeper into the tempdb objects in the buffer pool. The full query and description of the output can be found in the blog post IO — Where Are My TempDB Objects? (http://www.sqlsoldier.com/wp/sqlserver/iowherearemytempdbobjects).

Version Store Monitoring

The version store is a fairly new concept to tempdb administration. As noted at the opening of this document, the version store is used by a lot of processes, not just snapshot isolation. SQL attempts to keep the version store as clean as possible. One of the misunderstood concepts of version store cleanup is that it does not de-allocate versions as soon as they are no longer being used. SQL will only clean up versions that are older than the oldest active transaction in the version store. A long running transaction can easily cause the version store to grow very large because unused versions must be maintained as long as an older transaction than itself exists.

The DMV sys.dm_tran_active_snapshot_database_transactions returns information about all active transactions that generate or may potentially access a version in the version store, not including system transactions. You can query this DMV for the maximum elapsed_time_seconds column to see the oldest transaction in the version store.

```
SELECT TOP (1) *
FROM sys.dm_tran_active_snapshot_database_
transactions
ORDER BY elapsed_time_seconds DESC;
```

Key performance counters to watch for the version store are the Version Generation rate (KB/s), Version Cleanup rate (KB/s), and Version Store Size (KB) counters in the SQLServer:Transactions object. You can compare generation rate to cleanup rate to determine if the version store is growing faster than it is being cleaned up. These values can fluctuate considerably based on activity and length of transactions. The version store size should also be considered when evaluating the generation and cleanup rate. If the store size is not growing, then a high generation rate may not be much of a concern. Likewise, a really low cleanup rate may not be worth worrying about if the size is very low.

The recommendation is to baseline these counters and determine what is the normal functional range for your system when it is healthy. Then you can determine at what point the numbers may be indicative of a problem. The query below will allow you to capture all three counters at the same time.

```
SELECT object_name AS 'Counter Object',
     [Version Generation rate (KB/s)],
     [Version Cleanup rate (KB/s)],
     [Version Store Size (KB)]
FROM (SELECT object_name,
          counter_name,
          cntr_value
     FROM sys.dm_os_performance_counters
     WHERE object_name = 'SQLServer:Transactions'
     AND counter_name IN (
          'Version Generation rate (KB/s)',
          'Version Cleanup rate (KB/s)',
          'Version Store Size (KB)')) AS P
PIVOT (MIN(cntr_value)
     FOR counter_name IN (
          [Version Generation rate (KB/s)],
          [Version Cleanup rate (KB/s)],
          [Version Store Size (KB)])) AS Pvt;
```

CONCLUSION

Tempdb is a very critical component of your SQL Server and can be a major performance pain point if not managed properly. Having a performant tempdb starts with proper configuration and includes consistent base lining and monitoring of tempdb.

» Tempdb configuration:
  - Multiple data files pre-sized equally to avoid auto-growth
    - Use 1 data file per CPU if you are not comfortable with any part of the alternative
    - Or start with 1 data file per 2 or 4 CPUs, monitor for tempdb contention, and adjust the number of data files as needed
  - Pre-size the data and log files to use 90% of the available disk space
  - The log file should be twice the size of a single data file
  - Disable auto-growth of the data files
  - Set auto-growth of the log file to a hard value such as 512 MB

» Provide plenty of RAM for tempdb to use

» Make sure queries are optimized to avoid tempdb spills

» Monitor for tempdb contention

» Monitor the version store for long running transactions

» Baseline the version store to get a functional range for your server and monitor for changes outside of this range

These best practices will help you prevent, detect, and mitigate common performance problems. Couple these best practices with an understanding of how tempdb works and the role it plays will help guide you to troubleshooting tempdb issues.

idera®

ABOUT IDERA

Idera provides systems and application management software for Windows and Linux Servers, including solutions for performance monitoring, backup and recovery, security and compliance and server administration. Our market leading product suites provide 360 degree management solutions for the Microsoft SQL Server and SharePoint platforms as well as high performance backup and recovery for server operating systems. Whether you have ten servers or ten thousand, we give you the best performance and value with products that are easy to install and deploy and that deliver real and measurable ROI.

Idera is headquartered in Houston, TX with offices in London and Melbourne.

US	+1 713 523 4433 877 GO IDERA (464 3372)
EMEA	+44 (0) 1753 218410
APAC	+61 1300 307 211
MEXICO	+52 (55) 8421 6770
BRAZIL	+55 (11) 3230 7938

WEB	www.idera.com
TWITTER	www.twitter.com/Idera_Software
FACEBOOK	www.facebook.com/IderaSoftware
LINKEDIN	www.linkedin.com/groups?gid=2662613