



*The World's Largest Community  
of SQL Server Professionals*

# Optimizing tempdb Performance

Brad M McGehee

Director of DBA Education

Red Gate Software

[www.bradmcgehee.com](http://www.bradmcgehee.com)

# My Assumptions About You

- You are most likely a DBA or developer with one or more years SQL Server experience.
- You have a basic understanding of how to use Performance Monitor and Profiler.
- You have a basic understanding of DMV/DMFs.
- You have a basic understanding of SQL Server internals.

# What We Are Going to Learn Today

- What is Stored in tempdb
- tempdb Internals
- Ways to Monitor tempdb
- Ways to Optimize tempdb

Note: This session covers SQL Server 2005/2008

# What is Stored in tempdb?

- tempdb is used to store three different categories of temporary data:
  - User Objects
  - Internal Objects
  - Version Stores

# User Objects

- Local and global temporary tables and indexes
- User-defined tables and indexes
- Table variables
- Tables returned in table-valued functions

Note: These lists are not designed to be all inclusive.

# Internal Objects

- Work tables for **DBCC CHECKDB** and **DBCC CHECKTABLE**.
- Work tables for **hash operations**, such as joins and aggregations.
- Work tables for processing static or keyset **cursor**s.
- Work tables for processing **Service Broker** objects.
- Work files needed for many **GROUP BY, ORDER BY, UNION, SORT,** and **SELECT DISTINCT** operations.
- Work files for **sorts** that result from creating or rebuilding indexes (**SORT\_IN\_TEMPDB**).
- Storing **temporary large objects** (LOBs) as variables or parameters (if they won't fit into memory).

# Version Stores

- The version store is a collection of pages used to **store row-level versioning** of data.
- There are two types of version stores:
  1. **Common Version Store:** Used when:
    - Building the inserted and deleted tables for after triggers.
    - When DML is executed against a database using **snapshot transactions** or **read-committed row versioning** isolation levels.
    - When multiple active result sets (**MARS**) are used.
  2. **Online-Index-Build Version Store:** Used for online index builds or rebuilds. EE edition only.

# FYI: Quick Definitions

- **Snapshot Transaction Isolation Level:** Specifies that data read by any statement in a transaction will be transactionally consistent with the data that existed at the start of the transaction. Data modifications made by other transactions after the start of the transaction are not visible to statements executing in the current transaction. The effect is as if the statements in a transaction gets a snapshot of the committed data as it existed at the start of the transaction.
- **Read-Committed Row Versioning Isolation Level:** Row versioning is used to present each statement with a transactionally consistent snapshot of the data as it existed at the start of the statement. Locks are not used to protect the data from updates by other transactions.
- **Multiple Active Results Sets (MARS):** Allows applications to have more than one pending request per connection, and in particular, to have more than one active default result set per connection.

# Keep the Following in Mind

- If your SQL Server instance doesn't employ many of the activities just described, then **tempdb performance may not be an issue** for you.
- On the other hand, if your SQL Server instance uses many of these features, then **tempdb could become a significant bottleneck** for your SQL Server instance.
- Only by **investigation** will you know.
- Keep in mind that there is **only one tempdb**, and it is possible for **one misbehaved application and database to indirectly affect the performance of all the other databases on the same instance**.

# tempdb Internals (1)

- tempdb is dropped and recreated every time the SQL Server service is stopped and restarted.
- When SQL Server is restarted, tempdb inherits **many** of the characteristics of model, and creates an MDF file of 8MB and an LDF file of 1MB.
- Autogrowth is set to grow by 10% with unrestricted growth.
- Each SQL Server instance may have only **one tempdb**, although tempdb **may have multiple physical files**.

# tempdb Internals (2)

- tempdb often doesn't act like other databases:
  - tempdb only uses the simple recovery model.
  - Many database options can't be changed (e.g. Database Read-Only, Auto Close, Auto Shrink).
  - tempdb may not be dropped, detached, or attached.
  - tempdb may not be backed up, restored, be mirrored, have database snapshots made of it, or have many DBCC commands run against it.
  - tempdb logging works differently from regular logging. Operations are only **minimally logged** with enough information to roll back transactions, but not to be rolled forward. **The log is truncated constantly**, although it can grow with long-running transactions.

*Optional demo using SSMS, Profiler, and T-SQL Scripts*

# Execution Plans and tempdb

- When a query execution plan is cached, the tempdb work tables required by the plan, if any, are often cached.
- When a work table is cached, the table is truncated (from the previous execution of the code) and up to nine pages remain in the cache for reuse.
- This improves the performance of the next execution of the query, but can also take up lots of space in tempdb.
- If the system is low on memory, the Database Engine removes the execution plan and drops the associated work tables.

# Types of tempdb Problems

- Generally, there are **three** major problems you run into with tempdb:
  1. tempdb is experiencing an **I/O bottleneck**, hurting server performance.
  2. tempdb is experiencing **contention on various global allocation structures** (metadata pages) as temporary objects are being created, populated, and dropped. E.G. Any space-changing operation **acquires a latch** on PFS, SGAM or GAM pages to update space allocation metadata. A large number of such operations can cause **excessive waits** while latches are acquired, creating a bottleneck, and hurting performance.
  3. tempdb has **run out of space**.
- Ideally, you should be monitoring all these on a proactive basis to identify potential problems.

# FYI: Quick Definitions

- **PFS (Page Free Space)**: Tracks the allocation status of each page, such as whether an individual page has been allocated, and the amount of free space on each page. PFS uses one byte to track each page.
- **GAM (Global Allocation Map)**: Tracks what extents have been allocated. Each GAM page covers 64,000 extents, or almost 4 GB of data.
- **SGAM (Shared Global Allocation Map)**: Tracks which extents are currently being used as mixed extents, and also have at least one unused page. Each SGAM covers 64,000 extents, or almost 4 GB of data.

# Identifying tempdb I/O Problems (1)

- Use Performance Monitor to determine how busy the disk is where your tempdb MDF and LDF files are located.
- **LogicalDisk Object: Avg. Disk Sec/Read:** The average time, in seconds, of a read of data from disk. Numbers below are a general guide only and may not apply to your hardware configuration.
  - Less than 10 milliseconds (ms) = very good
  - Between 10-20 ms = okay
  - Between 20-50 ms = slow, needs attention
  - Greater than 50 ms = serious IO bottleneck
- **LogicalDisk Object: Avg. Disk Sec/Write:** The average time, in seconds, of a write of data to the disk. See above guidelines.
- **LogicalDisk: %Disk Time:** The percentage of elapsed time that the selected disk drive is busy servicing read or write requests. A general guideline is that if this value > 50%, there is a potential I/O bottleneck.

# Identifying tempdb I/O Problems (2)

- In SQL Server 2005, use the **Performance Dashboard** (you **must download it**) to see how busy your tempdb is compared to other databases.
- In SQL Server 2008, use the **Performance Data Collector** to see how busy your tempdb is compared to other databases.
- **Demo Performance Data Collector**
  - Disk Usage Report
  - Server Activity Report (Disk IO Usage)

# Identifying Contention on Allocation Structures Using Performance Counters

- Use these **performance counters** to monitor allocation/deallocation contention in SQL Server:
  - **Access Methods:Worktables Created/sec**: The number of work tables created per second. Work tables are temporary objects and are used to store results for query spool, LOB variables, and cursors. This number should **generally be less than 200**, but can vary based on your hardware.
  - **Access Methods:Workfiles Created/sec**: Number of work files created per second. Work files are **similar to work tables but are created by hashing operations**. Used to store temporary results for hash and hash aggregates. High values may indicate contention potential.
  - **Temp Tables Creation Rate**: The number of temporary tables or variables created/sec. High values may indicate contention potential.
  - **Temp Tables For Destruction**: The number of temporary tables or variables waiting to be destroyed by the cleanup system thread. **Should be near zero**, although spikes are common.

# Identifying Contention on Allocation Structures Using Wait States

- If latches are waiting to be acquired on tempdb pages, this may indicate allocation page contention.
- Use this code to find out:

```
SELECT session_id, wait_duration_ms,  
       resource_description  
FROM sys.dm_os_waiting_tasks  
WHERE wait_type like 'PAGE%LATCH_%' AND  
       resource_description like '2:%'
```

	session_id	wait_duration_ms	resource_description
1	54	109	2:1:2

The resource description refers to the metadata pages in tempdb.

# Identifying Contention on Allocation Structures Using DMVs

- This DMV can tell you how much space is allocated to the **various contents** of the tempdb, so you know what objects are most used in your tempdb.

```
SELECT
SUM (user_object_reserved_page_count)*8 as usr_obj_kb,
SUM (internal_object_reserved_page_count)*8 as
    internal_obj_kb,
SUM (version_store_reserved_page_count)*8 as
    version_store_kb
FROM sys.dm_db_file_space_usage
```

- **Demo**

# tempdb Space Allocation

- A higher % allocation for **user objects** implies that objects that are created by applications are the major consumers of tempdb. This may or may not be a cause of concern, but can **imply potential allocation page contention**.
- A higher % allocation for **internal objects** implies that the query plans make heavy use of tempdb. This may not be a problem, but you may want to **look at the queries and try to optimize them**.
- A higher % allocation for the **version store** implies that the version store cleanup cannot keep pace with version generation. **See if a long-running transaction is preventing version store cleanup**. Or, a **high transaction throughput** might be generating a large number of versions per minute.

# Monitoring tempdb Space

- Performance Counters:
  - SQL Server: Database: Data File(s) Size(KB): tempdb
  - SQL Server: Database: Log File(s) Used Size(KB): tempdb
  - SQL Server: Transactions: Free Space in tempdb (KB)
- DMV
  - `sys.dm_db_file_space_usage`
- *Consider creating an alert so you are notified if tempdb grows larger than its typical size.*

# If tempdb Space Runs Low, Errors Occur

- Check the SQL Server error log for these errors:
  - 1101 or 1105: A session has to allocate more space in tempdb in order to continue
  - 3959: The version store is full.
  - 3967: The version store has been forced to shrink because tempdb is full.
  - 3958 or 3966: A transaction is unable to find a required version record in tempdb.
- Be sure autogrowth is turned on for tempdb, and ensure that you have enough available free disk space.

# Tips for Optimizing tempdb: Overview

- Minimize the use of tempdb
- Add more RAM to your server
- Leave Auto Create Statistics & Auto Update Statistics on
- Pre-allocate tempdb space – *everyone needs to do this*
- Don't shrink tempdb if you don't need to
- Locate tempdb on its own array
- Locate tempdb on a fast I/O subsystem
- Divide tempdb among multiple physical files
- Avoid using Transparent Data Encryption (2008)

# How to Optimize tempdb—General

- Generally, if you are building a new SQL Server instance, it is a good idea to **assume that tempdb performance will become a problem**, and to take proactive steps to deal with this possibility.
- It is easier to deal with tempdb performance issues before they occur, than after they occur.
- The following tempdb performance tips **may or may not apply** to your particular situation.
- It is important to evaluate each one, and **determine which ones best fit your particular SQL Server's instance**. Not a one size fits all approach.

# Minimize the Use of tempdb (1)

- Use **standard performance tuning techniques** to boost your server's performance before you do anything. Examples include:
  - Don't return more rows than you need.
  - Don't sort data that doesn't need sorting, or sort it on the client.
  - Don't use UNION or SELECT DISTINCT if not needed.
  - Keep transactions short.
  - Use proper indexing to (e.g. consider using clustered indexes for columns where a lot of sorting occurs).
  - Avoid using local and global temp tables. Either rewrite your code to avoid using them, or consider creating a permanent work table.

# Minimize the Use of tempdb (2)

- Avoid using static and keyset-driven cursors. In most cases, cursors can be avoided by rewriting the code.
- Avoid using recursive common table expression queries. If the execution plan for such a query shows a spool operator, then tempdb is being used to execute it.
- Avoid using the SORT\_IN\_TEMPDB option when creating or rebuilding an index. If you decide to use this option, schedule to run during a slower time of the day.
- Avoid using online index rebuilding (Enterprise Edition), which uses row versioning and, in turn, uses tempdb. If you decide to use this option, schedule to run during a slower time of the day.
- Avoid using large object data types.
- Avoid using table variables.

# Minimize the Use of tempdb (3)

- Avoid aggregating excessive amounts of data.
- Avoid joins that indicate a hash-type join in the query execution plan. Rewrite the query or use better indexing.
- Avoid using triggers.
- Avoid using row-versioning-based transaction isolation levels.
- Schedule jobs, such as DBCC CHECKDB, that use tempdb heavily, at times of the day when the SQL Server instance is less busy. Or perform DBCC CHECKDB on a database backup on another server.

# I Need the Features You Told Me to Avoid

- The previous slides suggest avoiding taking advantage of a lot of tools that are available to you.
- If you need these tools, then use them, but keep in mind how they effect tempdb performance, and plan accordingly.
- On the other hand, don't be lazy.

# Add More RAM to Your Server

- Depending on the operation, SQL Server tries to perform the action in the buffer cache. (e.g. sorts, CTEs)
- If the buffer cache does not have enough available space, then the operation may have to spill to tempdb. This places additional overhead on tempdb.
- If your server is experiencing a memory bottleneck, then adding RAM can help reduce the load on tempdb.
- On the other hand, if your server has plenty of memory, adding more won't help tempdb performance.

# Leave Auto Statistics On

- By default, the database options Auto Create Statistics and Auto Update Statistics are turned on for tempdb.
- In most cases, don't turn these options off, as SQL Server will automatically create and update statistics as needed in temporary tables and indexes, helping to boost performance of many operations performed on them.

# Pre-Allocate tempdb Space

- The **default 8MB** of tempdb space that is automatically allocated is generally way **too small** a value.
- As SQL Server needs more tempdb space, it will request it, and through **autogrowth**, it will get it, **10% at a time**.
- When **autogrowth** kicks in, it uses up resources at often inconvenient times, and **transactions have to wait until autogrowth finishes**, maybe causing timeouts. Also, autogrowth can contribute to **physical file fragmentation**, which hurts I/O.
- Instead, you should **pre-allocate tempdb space** so that when SQL Server is restarted, that it is immediately sized to the optimum size.
- Using **Instant File Initialization** can greatly speed the process of growing the tempdb at system start.

# How Large Should tempdb Be?

- Start with an **educated guess** for new servers. For existing servers, **observe** how much tempdb is used.
- Monitor tempdb usage over **typical usage periods** to help determine typical tempdb MDF and LDF usage.
- Set tempdb size to the **maximum amount (plus ~10% extra)** based on your monitoring.
- **Keep autogrowth on** just in case you are wrong, and some unexpected transactions eats up a lot of additional tempdb space.
- Assuming that you have pre-allocated tempdb, then using a **fixed** autogrowth value from 1-10% of the size of tempdb should cover most unexpected growth spurts with minimal resource effect.

# How to Pre-allocate tempdb Space

- Use ALTER DATABASE, or SSMS, to increase the size of the tempdb database MDF and LDF files.

```
USE master;
GO
ALTER DATABASE tempdb
MODIFY FILE
    (NAME = tempdev,
     SIZE = 20MB);
GO
ALTER DATABASE tempdb
MODIFY FILE
    (NAME = templog,
     SIZE = 10MB)
GO
```

# Don't Shrink Tempdb

- The amount of space actually used out of the amount of **space allocated to tempdb** can vary wildly over time. This is normal behavior.
- **Don't be tempted to shrink tempdb just because it is not using all of its space.** If you do, you could start a cycle of growth and shrinkage that wastes server resources.
- If the **rare cases** where tempdb grows excessively (due to a wild query, for example), and if you don't expect the behavior to reoccur, then you may want to shrink tempdb to its "normal" size.
- **The best way to shrink tempdb is to stop and restart the server** so that the pre-allocated size is recreated. Using DBCC SHRINKFILE might work, but is often ineffective.

# Locate tempdb on its Own Array

- If tempdb becomes very active, disk I/O contention can become an issue.
- One way to help mitigate this problem is to locate tempdb on its own array (or LUN) so that tempdb activity doesn't have to compete with other disk I/O activity.
- In a perfect world, the tempdb MFD and LDF files should also be isolated.

# Locate tempdb on Fast I/O Subsystem

- Always locate tempdb on the **fastest I/O subsystem** you have available.
- **Prefer RAID 1 or RAID 10.** RAID 5 is slow for writes and should generally be avoided for tempdb, as tempdb is often write-intensive.
- If using a SAN, consult with a SAN engineer to **ensure that tempdb won't be affected by other disk I/O.**
- Consider **SSD** drives for tempdb MDF and LDF files.

# How to Move tempdb

- Determine the current location of the MDF and LDF files
- Run the ALTER DATABASE command to move the files

```
USE master;  
GO  
ALTER DATABASE tempdb  
MODIFY FILE (NAME = tempdev, FILENAME = 'E:\SQLData\tempdb.mdf');  
GO  
ALTER DATABASE tempdb  
MODIFY FILE (NAME = templog, FILENAME = 'F:\SQLLog\templog.ldf');  
GO
```

- Stop and restart the SQL Server instance
- Delete the old MDF and LDF files

# Divide tempdb Into Multiple Files

- By default, when tempdb is created, the MDF and LDF are created as single physical files.
- While the LDF should always remain as a single physical file, often dividing the tempdb MDF into multiple physical files can help performance.
- Multiple files can reduce contention on various global allocation structures by spreading activity over multiple physical files. This is particularly useful for those tempdb database which spend a large percentage of time allocating and de-allocating tables. If you don't have contention issues, then you probably don't need multiple files.
- Keep in mind that using more physical disk files for tempdb can increase switching costs and file management overhead, which could potentially hurt performance.
- Thus, a balance must be maintained. How is this done?

# How Many Physical Files Should tempdb Have?

- Microsoft recommends (very generally) that tempdb should have **one physical file per CPU core**.
- This often **makes sense with 8 or fewer cores**.
- But with more than 8 cores, you may be reaching the point where the **overhead of maintaining multiple tempdb physical files overcomes the benefits**.
- Only through **formal testing** will you be able to determine the optimum number of physical files for your particular instance's tempdb.

# Recommendations for Multiple tempdb Files

- If you choose to use multiple tempdb files, keep the following in mind:
  - Each physical file must be identical in size.
  - The autogrowth setting for each physical file must be identical.
- The above is important because SQL Server uses a proportional fill strategy to fill the physical files, and if the files become different sizes, then tempdb becomes less efficient.

# Example for Creating Multiple Files

- Determine where current tempdb file is at, and its size.
- Determine what file(s) have to be added.
- Run ALTER DATABASE command, or use SSMS.

```
USE master;  
GO  
ALTER DATABASE [tempdb]  
ADD FILE ( NAME = 'tempdev1',  
FILENAME = 'f:\tempdb1.ndf' ,  
SIZE = 8192KB , FILEGROWTH = 10%)
```

- Note: Restart SQL Server to get physical files in synch

# Avoid Using TDE

- SQL Server 2008 offers a database-level encryption feature called Transparent Database Encryption (TDE).
- If this is turned on for **one or more** databases on a SQL Server instance, then **all the activity in tempdb** (whether it comes from a encrypted or non-encrypted database) will be encrypted.
- Encryption increases CPU usage and slows down tempdb performance.
- If you decide to use TDE, you will want to incorporate as many of the tempdb performance tuning tips that I have suggested in order to help overcome the additional burden added by TDE.

# Take Aways From This Session

- You don't know if you have a tempdb performance issue unless you check. Check often on critical servers.
- It is easier to prevent tempdb performance problems before they occur, than after they occur. This includes both application design and server configuration.
- Implement as many as these recommendations as you can on your SQL Server instances to be proactive and to help prevent potential tempdb performance issues.

# Q&A

- Please ask your questions clearly and loudly.
- If you don't get your questions answered now, see me after the session, or e-mail me.

# Find Out More

## Free E-Books:

- [www.sqlservercentral.com/Books](http://www.sqlservercentral.com/Books)

## Check these out:

- [www.SQLServerCentral.com](http://www.SQLServerCentral.com)
- [www.Simple-Talk.com](http://www.Simple-Talk.com)

## Contact me at:

[bradmcgehee@hotmail.com](mailto:bradmcgehee@hotmail.com)

## Blogs:

[www.bradmcgehee.com](http://www.bradmcgehee.com)

[www.twitter.com/bradmcgehee](http://www.twitter.com/bradmcgehee)

[Click Here for a free 14-day trial of the Red Gate SQL Server Toolbelt](#)