Monitoring tempdb space usage -- SQL Server

https://thesqldude.com/2012/05/15/monitoring-tempdb-space-usage-and-scripts-for-finding-queries-which-are-using-excessive-tempdb-space/

Many times during the life of a DBA, you might notice the tempdb database growing excessively, though no changes have recently been done. It's often the case that due to data increase, the application T-SQL queries are not written to scale up, hence end up doing excessive sorting/hashing operations which consume space from your tempdb database. Here are some T-SQL scripts that you can use to monitor who/what is consuming space from tempdb and plan accordingly.

Before we get into identifying queries that use tempdb, it is very important to understand what all activities in SQL Server (both internal & user activities), which use the tempdb database. Broadly you can classify these into 3 categories:-

1. Internal Objects
2. Version Stores
3. User Objects

From a feature perspective, here are the features in SQL Server that use space from tempdb.

1. Query
2. Triggers
3. Snapshot isolation and read committed snapshot (RCSI)
4. MARS
5. Online index creation
6. Temporary tables, table variables, and table-valued functions
7. DBCC CHECK
8. LOB parameters
9. Cursors
10. Service Broker and event notification
11. XML and LOB variables
12. Query notifications
13. Database mail
14. Index creation
15. User-defined functions

From a query performance standpoint, here are some operators that use tempdb space.

1. Sort Operator : The sort operator needs **tempdb** space to sort the full rowset of incoming rows. This is usually send when user ODER BY and also for DISTINCT ORDER BY

2. Hash Match Operator: Depending on the size of row, a hash table could use tempdb

3. Spool Operator: This operator is used to save the intermediate set of rows for re-use and uses the tempdb database to save the query result set.

Tempdb out of space error

Error: 1105, Severity 17, State 2

Could not allocate space for object dbo.TBL1 in database 'tempdb' because the 'PRIMARY' filegroup is full.

Identify which type of tempdb objects are consuming space

The following query helps you understand if user objects or version store or internal objects are the ones using the space in tempdb. According to this output, you can focus on the below sections.

```
SELECT
SUM (user_object_reserved_page_count)*8 as user_obj_kb,
SUM (internal_object_reserved_page_count)*8 as internal_obj_kb,
SUM (version_store_reserved_page_count)*8  as version_store_kb,
SUM (unallocated_extent_page_count)*8 as freespace_kb,
SUM (mixed_extent_page_count)*8 as mixedextent_kb
FROM sys.dm_db_file_space_usage
```

If **user_obj_kb** is the highest consumer, then you that objects are being created by user queries like local or global temp tables or table variables. Also don't forget to check if there are any permanent
tables created in TempDB. Very rare, but I've seen this happening.
If **version_store_kb** is the highest consumer, then it means that the version store is growing faster than the clean up. Most likely there are long running transactions or open transaction (Sleeping state),
which are preventing the cleanup and hence not release tempdb space back.

Query that identifies the currently active T-SQL query, it's text and the Application that is consuming a lot of tempdb space

```
SELECT es.host_name , es.login_name , es.program_name,
st.dbid as QueryExecContextDBID, DB_NAME(st.dbid) as QueryExecContextDBNAME, st.objectid as ModuleObjectId,
SUBSTRING(st.text, er.statement_start_offset/2 + 1,(CASE WHEN er.statement_end_offset = -1 THEN
LEN(CONVERT(nvarchar(max),st.text)) * 2 ELSE er.statement_end_offset
END - er.statement_start_offset)/2) as Query_Text,
tsu.session_id ,tsu.request_id, tsu.exec_context_id,
(tsu.user_objects_alloc_page_count - tsu.user_objects_dealloc_page_count) as OutStanding_user_objects_page_counts,
(tsu.internal_objects_alloc_page_count - tsu.internal_objects_dealloc_page_count) as
OutStanding_internal_objects_page_counts,
er.start_time, er.command, er.open_transaction_count, er.percent_complete, er.estimated_completion_time, er.cpu_time,
er.total_elapsed_time, er.reads,er.writes,
er.logical_reads, er.granted_query_memory
FROM sys.dm_db_task_space_usage tsu inner join sys.dm_exec_requests er
 ON ( tsu.session_id = er.session_id and tsu.request_id = er.request_id)
inner join sys.dm_exec_sessions es ON ( tsu.session_id = es.session_id )
CROSS APPLY sys.dm_exec_sql_text(er.sql_handle) st
WHERE (tsu.internal_objects_alloc_page_count+tsu.user_objects_alloc_page_count) > 0
ORDER BY (tsu.user_objects_alloc_page_count - tsu.user_objects_dealloc_page_count)+(tsu.internal_objects_alloc_page_count
- tsu.internal_objects_dealloc_page_count)
DESC
```

Tempdb and the Version Store

The version stored (SQL 2005 onwards) is a collection of objects that are used when Snapshot Isolation or Read-Committed Snapshot Isolation (RCSI) or online index rebuild etc. are used in a database.

Version store contains the committed rows which is how a SELECT operation does not get blocked when another UPDATE/DELETE is operating on the same row, because the SELECT reads the row from the version store, instead of the actual base table. When you enable this, the row has to be stored somewhere and tempdb happens to be the place. A row is maintained in the version store when there are transactions operating on that row in questions. When the transaction is committed, the row is cleaned up from the version store tables.

You can check the version store using the DMV sys.dm_tran_version_store

At times, when there are long running transactions or orphaned transactions, you might notice tempdb growth due to the version store.

You can use the following query to find the oldest transactions that are active and using row versioning.

```
SELECT top 5 a.session_id, a.transaction_id, a.transaction_sequence_num, a.elapsed_time_seconds,
b.program_name, b.open_tran, b.status
FROM sys.dm_tran_active_snapshot_database_transactions a
join sys.sysprocesses b
on a.session_id = b.spid
ORDER BY elapsed_time_seconds DESC
```

Trace Flag 1118

This trace flag is available starting with SQL 2000 SP3 to reduce tempdb contention by forcing uniform extent allocations as opposed to mixed extent allocations. This trace flag is only to be used if you seeing contention (wait_Stats) on the PFS/GAM pages like 2:1:1 etc.. More internal details on this trace flag is available in Paul Randal's blog post **here**.

Not only does enabling the trace flag help but you need to create multiple tempdb files equal to the number of logical processors. So if you have 4 CPU's you will create 4 tempdb data files. Now, what if you have 16 or 32 processors, do you still need to create that many tempdb files?

The answer is NO, you don't have to. The above recommendation has been stated in many KB articles like **http://support.microsoft.com/default.aspx?scid=kb;EN-US;328551**

If the number of logical processors on your server is greater than or equal to 8, then use 8 data files for tempdb. If the number of logical processors is less than 8, then use as many data files as your processor count.

You can use the following against any of the SQL Servers you manage to find out if any change is required in the tempdb data files to reduce contention and improve general performance.

```
Declare @tempdbfilecount as int;
select @tempdbfilecount = (select count(*) from sys.master_files where database_id=2 and type=0);
WITH Processor_CTE ([cpu_count], [hyperthread_ratio])
AS
(
    SELECT  cpu_count, hyperthread_ratio
    FROM sys.dm_os_sys_info sysinfo
)
select Processor_CTE.cpu_count as [# of Logical Processors], @tempdbfilecount as [Current_Tempdb_DataFileCount],
(case
    when (cpu_count<8 and @tempdbfilecount=cpu_count)  then 'No'
    when (cpu_count<8 and @tempdbfilecount<>cpu_count and @tempdbfilecount<cpu_count) then 'Yes'
    when (cpu_count<8 and @tempdbfilecount<>cpu_count and @tempdbfilecount>cpu_count) then 'No'
    when (cpu_count>=8 and @tempdbfilecount=cpu_count)  then 'No (Depends on continued Contention)'
    when (cpu_count>=8 and @tempdbfilecount<>cpu_count and @tempdbfilecount<cpu_count) then 'Yes'
    when (cpu_count>=8 and @tempdbfilecount<>cpu_count and @tempdbfilecount>cpu_count) then 'No (Depends on
continued Contention)'
end) AS [TempDB_DataFileCount_ChangeRequired]
from Processor_CTE;
```

Here is a sample output. As you can see I have 8 processors and only 1 tempdb file. So I need to add 7 more files.

	# of Logical Processors	Current_Tempdb_DataFileCount	TempDB_DataFileCount_ChangeRequired
1	8	1	Yes

Last point before I wrap up this post. Once your tempdb database or log file is full, you have these options:-

1. Either you have to rollback any transactions consuming tempdb space or kill the transactions (not a good idea).

2. Create additional tempdb files in other drives which have free space, while you dig around to find the culprit who is growing tempdb.

3. Restart your SQL Server service.

Have fun working with tempdb. Here are some good references

Working with tempdb – **http://technet.microsoft.com/hi-in/library/cc966545(en-us).aspx**

Storage Engine Blog – **http://blogs.msdn.com/b/sqlserverstorageengine/archive/2009/01/12/tempdb-monitoring-and-troubleshooting-out-of-space.aspx**