Managing space for databases and pools

Last updated by | Vitor Tomaz | Jun 8, 2022 at 5:37 AM PDT

Contents

See Managing space for databases and pools under SQL DB as well for more TSGs

Self-help content presented in Azure Portal

(This content was shown to the customer during case submission. It's also visible on 'Diagnose and solve problems' blade.)

Resolve low or no free space available

- Increase storage, if service tier allows it.
- Upgrade to a service tier that can provide more storage.
- If you are facing issues due to Tempdb being full, you can do a failover to clear tempdb.

Review the different options to failover your Azure SQL Managed Instance to a new node, which clears tempdb. Existing connections will be dropped during the failover, so applications should handle the disconnect with appropriate retry logic. Consider shrinking database files one at time using DBCC SHRINKFILE.

Queries to calculate database and objects sizes

Return the size of your database (in megabytes)

```
SELECT SUM(CAST(FILEPROPERTY(name, 'SpaceUsed') AS bigint) * 8192.) / 1024 / 1024 AS DatabaseSizeInMB
FROM sys.database_files
WHERE type_desc = 'ROWS';
GO
```

Returns the size of individual objects (in megabytes) in your database

```
SELECT sys.objects.name, SUM(reserved_page_count) * 8.0 / 1024
FROM sys.dm_db_partition_stats, sys.objects
WHERE sys.dm_db_partition_stats.object_id = sys.objects.object_id
GROUP BY sys.objects.name;
GO
```

Monitor tempdb space

Tempdb contention

The top wait types associated with tempdb issues are PAGELATCH_* (not PAGEIOLATCH_*). However, PAGELATCH_* waits do not always mean you have tempdb contention. This wait may also mean that you have user-object data page contention due to concurrent requests targeting the same data page.

To confirm tempdb contention, use sys.dm_exec_requests to verify that the wait_resource value begins with `2:x:y`, where 2 is tempdb database ID, x is the file ID, and y is the page ID.

For example: the Wait Resource `2:1:3` is:

- DatabaseID: 2 (TempDB)
- File Number: 1 (The first data file)
- Page Number: 3 (SGAM Page)

For tempdb contention, a common method is to reduce or re-write application code that relies on tempdb. Common tempdb usage areas include:

- Temp tables
- Table variables
- Table-valued parameters
- Version store usage (specifically associated with long running transactions)
- Queries that have query plans that use sorts, hash joins, and spools

Useful queries

Identify top queries that use table variables and temporary tables

```
SELECT plan_handle, execution_count, query_plan
INTO #tmpPlan
FROM sys.dm_exec_query_stats
     CROSS APPLY sys.dm_exec_query_plan(plan_handle);
GO

WITH XMLNAMESPACES('http://schemas.microsoft.com/sqlserver/2004/07/showplan' AS sp)
SELECT plan_handle, stmt.stmt_details.value('@Database', 'varchar(max)') 'Database', stmt.stmt_details.value('
INTO #tmp2
FROM(SELECT CAST(query_plan AS XML) sqlplan, plan_handle FROM #tmpPlan) AS p
    CROSS APPLY sqlplan.nodes('//sp:Object') AS stmt(stmt_details);
GO

SELECT t.plan_handle, [Database], [Schema], [table], execution_count
FROM(SELECT DISTINCT plan_handle, [Database], [Schema], [table]
     FROM #tmp2
     WHERE [table] LIKE '%@%' OR [table] LIKE '%#%') AS t
    JOIN #tmpPlan AS t2 ON t.plan_handle=t2.plan_handle;
```

Free space in TempDB

```
SELECT SUM(unallocated_extent_page_count) AS [free pages],
(SUM(unallocated_extent_page_count)*1.0/128) AS [free space in MB]
FROM tempdb.sys.dm_db_file_space_usage
```

Long running transactions in TempDB

```
SELECT transaction_id
FROM tempdb.sys.dm_tran_active_snapshot_database_transactions
ORDER BY elapsed_time_seconds DESC
```

Sessions that are consuming logs in TempDB

```
SELECT SUM(user_object_reserved_page_count) AS [user object pages used],
(SUM(user_object_reserved_page_count)*1.0/128) AS [user object space in MB]
FROM tempdb.sys.dm_db_file_space_usage
select dst.session_id, dbt.transaction_id, dat.transaction_begin_time, dbt.database_transaction_begin_time, db
FROM tempdb.sys.dm_tran_database_transactions dbt, tempdb.sys.dm_tran_active_transactions dat, tempdb.sys.dm_t
WHERE dst.transaction_id = dbt.transaction_id and dbt.transaction_id = dat.transaction_id and dbt.database_id
```

Additonal information on monitoring tempdb space.

T-Log full due to 'Active_Transaction'

'ACTIVE_TRANSACTION' explains that there is an active transaction in the database and because of this, the transaction log file for the database can't be truncated. To resolve this issue, find the active transaction and terminate it. To understand which transaction is holding up the transaction log for tempdb, use the following query:

```
SELECT SUBSTRING(st.text, er.statement_start_offset/2 + 1,(CASE WHEN er.statement_end_offset = -1 THEN LEN(CON
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_cou
(tsu.internal_objects_alloc_page_count - tsu.internal_objects_dealloc_page_count) as OutStanding_internal_obje
er.start_time, er.command, er.open_transaction_count, er.percent_complete, er.estimated_completion_time, er.cp
er.reads,er.writes, er.logical_reads, er.granted_query_memory,es.host_name , es.login_name , es.program_name
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
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
```

Once you have found the user activity, use the 'session_id' and kill or cancel the query.

Additional references:

- 'Tempdb' is full due to 'ACTIVE_TRANSACTION'
- Monitoring TempDB usage

T-Log full due to 'REPLICATION'

This issue can occur if the transaction log size has been set to an upper limit or autogrow is not allowed. In this case, enabling autogrow or increasing the log size manually may resolve the issue.

If autogrowth on the databases is disabled, the database is online, and sufficient space is available on the disk, do either of the following:

- Manually increase the file size to produce a single growth increment. These are general recommendations on log size growth and size.
- Turn on autogrow by using the ALTER DATABASE statement to set a non-zero growth increment for the FILEGROWTH option. See Considerations for the autogrow and autoshrink settings in SQL Server.

If the log file size has met maximum limit, increase the maximum size to mitigate the issue:

```
ALTER DATABASE DB
MODIFY FILE
(
    NAME = db_log,
    MAXSIZE = <something_larger_than_setsize_MB>
);
```

Additional references:

- Change log size limit or enable Autogrow
- Troubleshoot transaction log errors

Shrink is slow

- Verify that the file has adequate free space:

```
SELECT name, size/128.0 FileSizeInMB, size/128.0 - CAST(FILEPROPERTY(name, 'SpaceUsed') AS int)/128.0 AS
FROM sys.database_files;
```

- Run the DBCC SQLPERF command to return the space used in the transaction log:

```
DBCC SQLPERF(LOGSPACE);
```

- Try incremental shrink using [DBCC Shrinkfile]().
- If the shrink operation is complete and the size was not reduced, check for possible blockers:
  - High fragmentation: To check if there is any high fragmented index in the database use the following query:

    Note: This could take some time on a very large database

    ```
    SELECT DB_NAME(database_id) AS [Database Name], OBJECT_NAME(ps.OBJECT_ID) AS [Object Name],
    i.name AS [Index Name], ps.index_id, ps.index_type_desc, ps.avg_fragmentation_in_percent,
    ps.fragment_count, ps.page_count, i.fill_factor, i.has_filter, i.filter_definition
    FROM sys.dm_db_index_physical_stats(DB_ID(),NULL, NULL, NULL ,'LIMITED') AS ps
    INNER JOIN sys.indexes AS i WITH (NOLOCK)
    ON ps.[object_id] = i.[object_id]
    AND ps.index_id = i.index_id
    WHERE database_id = DB_ID()
    AND page_count > 250
    ORDER BY avg_fragmentation_in_percent DESC OPTION (RECOMPILE);
    ```

  - Paused resumable index operation: To check if there is any paused operation check this on customer database:

    ```
    SELECT count(*) FROM sys.index_resumable_operations
    ```

- To perform an incremental shrink, use the most recent version of [this script]().
- After a shrink operation is completed against data files, indexes may become fragmented and lose their performance optimization effectiveness for certain workloads, such as queries using large scans. If performance degradation occurs after the shrink operation is complete, consider performing index maintenance to rebuild indexes.
- If page density in the database is low, a shrink will take longer because it will have to move more pages in each data file. Microsoft recommends determining average page density before executing shrink commands. If page density is low, rebuild or reorganize indexes to increase page density before running shrink. For more information, including a sample script to determine page density, see [Optimize index maintenance to improve query performance and reduce resource consumption]().

Additional references:

- [Why LOB data makes shrink run slooooowly]()

Could not allocate a new page for database 'x' because of insufficient disk space in filegroup 'PRIMARY'

This error usually occurs due to insufficient disk space or the database autogrowth settings have limited the growth (You may have disabled autogrowth and the database size has reached the maximum limit). To resolve, increase the max size of the database files. You can find these settings in Management Studio: right-click the database -> **Properties** -> **Files** -> **Autogrowth/Max Size** column.

You can also delete data from your database to free up space for the new object. The space is released after data deletion, but the database file size might stay as it is if the SQL Server didn't shrink files automatically. In that case, shrink the files manually using DBCC SHRINKFILE.

If deleting the data/table fails to free up space, you can add some space and rebuild the clustered indexes. If the table does not have a clustered index, create a new one. Rebuilding the clustered index can reduce fragmentation.

Long term mitigations

- Analyze if column data types are the right ones for the data they will store.
- Test compressing tables and/or indexes.
- Depending on the workload and service tier, test Columnstore indexes.
- If possible, export and remove data that is not needed or move to another database, with lower service tier, if the access pattern is low.

Data Compression to reduce size of database

The data compression feature helps to reduce the size of the database. In addition to saving space, data compression can help improve performance of I/O intensive workloads because the data is stored in fewer pages and queries need to read fewer pages from disk. However, extra CPU resources are required on the database server to compress and decompress the data, while data is exchanged with the application.

Set up alerts to monitor storage

Several maintenance and administrative actions performed in a database require some storage for temporary data, so allowing the storage usage of the database to become very close to its limits is not recommended. To prevent your database from getting very close to the storage limit, implement alerts, or find other ways to monitor the storage usage.

Resources

- Troubleshoot TempDB issues
- Enable Compression on a Table or Index

How good have you found this content?