Azure SQL DB or SQL MI used data space is larger than expected

Last updated by | Vitor Tomaz | Jun 8, 2022 at 5:33 AM PDT

Contents

Issue

Azure SQL Database or SQL Managed Instance Database used size is much larger than expected when compared with the actual number of records in the tables. Customer sometimes notices that the database size is larger than expected in terms of used space. (as described in some cases, the database ballooned).

Investigation/Analysis

The data space used in an Azure SQL database or SQL Managed Instance database can be larger than expected - and on occasions significantly larger than expected – when compared with the actual number of records in the individual tables. This can lead to the impression of a problem with the database storage itself. However, this is almost certainly never the case and the issue can be resolved by carrying out a few checkups and maintenance procedures. To check if the issue is happening for a particular table, you can run the below script on the customer side as it will show you the reserved and used page count for each table (this can be used for both Azure SQL DB and Azure SQL MI):

> all the scripts in this TSG will work for both Azure SQL database and Azure SQL managed instance

```
SELECT o.name ,
SUM (p.reserved_page_count) as reserved_page_count,
SUM (p.used_page_count) as used_page_count,
SUM (
CASE
WHEN (p.index_id < 2) THEN (p.in_row_data_page_count + p.lob_used_page_count + p.row_overflow_used_page_count)
ELSE p.lob_used_page_count + p.row_overflow_used_page_count
END
) as DataPages,
SUM (
CASE
WHEN (p.index_id < 2) THEN row_count
ELSE 0
END
) as rowCounts
FROM sys.dm_db_partition_stats p inner join sys.objects o
on p.object_id = o.object_id
group by o.name
```

Samples of the result:

Results | Messages

	name	reserved_page_count	used_page_count	DataPages	rowCounts
1	Address	108	30	10	450
2	BuildVersion	9	2	1	1
3	Customer	107	52	35	847
4	CustomerAddress	50	11	5	417
5	db_ledger_blocks	0	0	0	0
6	db_ledger_transactions	0	0	0	0
7	ErrorLog	0	0	0	0
8	filestream_tombstone_2073058421	0	0	0	0
9	filetable_updates_2105058535	0	0	0	0
10	HeapTest	9	3	2	1
11	plan_persist_context_settings	9	2	1	14

From CSS side, you can use the below kusto query to check the tables size:

```
let myAppName="******";
let PartitionStats=materialize(MonWiDmDbPartitionStats
| where AppName == myAppName and logical_database_name !='master' and index_id in (0,1)
| summarize used_page_count=max(used_page_count), row_count=max(row_count) by database_id, logical_database_na
let FilteredResults=materialize(MonDatabaseMetadata
| where AppName == myAppName and logical_db_name != 'master'
| where (table_name=='sysclsobjs' and class==50) or (table_name=='sysschobjs' and ['type']=='U ') or (table_na
| project TIMESTAMP, table_name, class, ['type'], id, name, nsid, indid);
let schemas=FilteredResults
| where (table_name=='sysclsobjs' and class==50)
| summarize by schema_id=id, schema_name=tolower(name);
let tables=FilteredResults
| where (table_name=='sysschobjs' and ['type']=='U ')
| summarize by schema_id=nsid, object_id=id, table_name=name;
let indexes=FilteredResults
| where (table_name=='sysidxstats' and indid in (0,1))
| extend index_type_desc=iff(['type']==0, 'HEAP', iff(['type']==1, 'CLUSTERED', iff(['type']==5, 'CCI', tostri
| summarize by object_id=id,index_id=indid,index_type=type,index_type_desc;
tables
| join kind=inner (schemas) on schema_id
| join kind=inner (indexes) on object_id
| join kind=inner (PartitionStats) on object_id
| project database_id, logical_database_name, schema_id, schema_name, object_id, table_name, table_type=index_
| sort by schema_name asc, table_name asc, object_id asc, data_date asc
```

To check the database size you can use the below Kusto query:

```
MonDmIoVirtualFileStats
| where AppName == "******"
| project TIMESTAMP, db_name , type_desc , spaceused_mb
```

For such cases, the cause could be one of the below:

1) Index fragmentation

Fragmentation exists when indexes have pages in which the logical ordering within the index, based on the key value of the index, does not match the physical ordering inside the index pages. The following example finds the average fragmentation percentage of all indexes in the Sales.SalesOrderDetail table in the AdventureWorks2012 database:

```
SELECT a.index_id, name, avg_fragmentation_in_percent, fragment_count, avg_fragment_size_in_pages  FROM sys.dm
JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id = b.index_id
```

2) Ghost Records

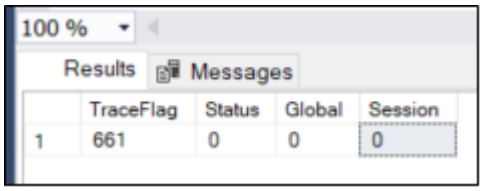
Ghost records are records that are deleted from a leaf level of an index page but are not physically removed from the page. Instead, the record is marked as ghosted meaning to be deleted. This means that the row stays on the page, but the row header is modified to indicate the row is a confirmed ghost record. The reason behind this is to optimize performance during a delete operation. Ghosts are necessary for row-level locking, but also necessary for snapshot isolation where we need to maintain the older versions of rows. The number of ghost records can build up in a database until they are cleaned. The database engine runs a ghost cleanup process in

the background that sometime after the delete transaction is committed, physically removes ghosted records from pages.

It is also possible the ghost cleanup process is disabled (not generally recommended). Disabling the ghost cleanup process can cause your database to grow unnecessarily large and can lead to performance issues. You can check if the ghost cleanup process is disabled by running the following command:

```
DBCC Tracestatus (661)
```

If the "Status" flag is set to 0, then this indicates that the ghost clean-up is enabled. If "Status" flag is set 1, then the process has been disabled.



To confirm if there are ghost records on your database execute this T-SQL:

```
SELECT sum(ghost_record_count) total_ghost_records, db_name(database_id)
FROM sys.dm_db_index_physical_stats (NULL, NULL, NULL, NULL, 'SAMPLED')
GROUP BY database_id
ORDER BY total_ghost_records DESC
```

If there are ghost records, you can delete the ghost records manually from the database by executing an index rebuild. This process reclaims disk space by compacting the pages based on the specified or existing fill factor setting and reorders the index rows in adjoining pages.

For more details about the Ghost clean process refer to the following guide: - Ghost cleanup process guide - SQL Server | Microsoft Docs

3) Persisted Version Store (PVS)

PVS is a database engine mechanism for persisting the row versions generated in the database itself instead of the traditional tempdb version store. PVS enables resource isolation and improves availability of readable secondaries. The accelerated database recovery (ADR) feature uses PVS.

Best practices for Accelerated Database Recovery

Though one objective of ADR is to speed up database recovery due to redo long active transactions, long-running transactions can delay version cleanup and increase the size of the PVS.

- Avoid long-running transactions in the database.
- Avoid long-running transactions in the database. Though one objective of ADR is to speed up database recovery due to redo long active transactions, long-running transactions can delay version cleanup and increase the size of the PVS.
- Avoid large transactions with data definition changes or DDL operations. ADR uses a SLOG (system log stream) mechanism to track DDL operations used in recovery. The SLOG is only used while the transaction

active. SLOG is checkpointed, so avoiding large transactions that use SLOG can help overall performance. These scenarios can cause the SLOG to take up more space:
    - Many DDLs are executed in one transaction. For example, in one transaction, rapidly creating and dropping temp tables.
    - A table has very large number of partitions/indexes that are modified. For example, a DROP TABLE operation on such table would require a large reservation of SLOG memory, which would delay truncation of the transaction log and delay undo/redo operations. The workaround can be drop the indexes individually and gradually, then drop the table. For more information on the SLOG, see ADR recovery components.
- Prevent or reduce unnecessary aborted situations. A high abort rate will put pressure on the PVS cleaner and lower ADR performance. The aborts may come from a high rate of deadlocks, duplicate keys, or other constraint violations.
    - The `sys.dm_tran_aborted_transactions` DMV shows all aborted transactions on the SQL Server instance. The `nested_abort` column indicates that the transaction committed but there are portions that aborted (savepoints or nested transactions) which can block the PVS cleanup process. For more information, see sys.dm_tran_aborted_transactions (Transact-SQL).
    - To activate the PVS cleanup process manually between workloads or during maintenance windows, use `sys.sp_persistent_version_cleanup` . For more information, see sys.sp_persistent_version_cleanup.
- If you observe issues either with storage usage, high abort transaction and other factors, see Troubleshooting Accelerated Database Recovery (ADR) on SQL Server.

Checking PVS size

a. You can use the below kusto query to check the PVS size:

```
MonSqlTransactions
| where AppName =~ "*****"
//| where database_id ==
| where persisted_version_store_kb > 0
|where database_id  == "5"
| project PreciseTimeStamp , persisted_version_store_kb, database_id
| order by PreciseTimeStamp asc nulls last
```

b. From ASC -> SQL Troubleshooters -> Performance -> Advanced tab, you can see the "Persisted version store si

Then you can check the transaction happens during timeframe to validate if customer have frequent insert/delet

```
MonWiQdsExecStats

| where TIMESTAMP  between(datetime('9/16/2021 05:00 AM')..datetime('9/17/2021 15:20 PM'))

| where AppName == "dc93d02cc10c"

| where LogicalServerName =~"sqlsvrprodpub01" and database_name=~ "sqldbprodpub01"

| where is_primary==1

//| where statement_type == "x_estypInsert"

|summarize sum(execution_count) by bin(TIMESTAMP,30m),statement_type //,query_hash

|render timechart

```

PVS store stores row versions that are generated by any DML operations it's part of the accelarated Datab

c. From customer side, you can check the database PVS size by running the following T-SQL:

```
SELECT DB_Name(database_id), persistent_version_store_size_kb
FROM sys.dm_tran_persistent_version_store_stats
WHERE database_id = add your database ID
```

If PVS size is large you can enforce the PVS cleanup by executing the following T-SQL:

```
EXEC sys.sp_persistent_version_cleanup [database_name]
```

The links below contains more information about PVS and ADR:

- Accelerated database recovery - Azure SQL | Microsoft Docs
- Manage accelerated database recovery - SQL Server | Microsoft Docs

Mitigation

1) Index Fragmentation

The following links detail how to rebuild indexes to reduce the fragmentation (the second link includes an index and statistics maintenance script you can download):

- Resolve index fragmentation by reorganizing or rebuilding indexes
- How to maintain Azure SQL Indexes and Statistics

2) Ghost Records

you can delete the ghost records manually from the database by executing an index rebuild. This process reclaims disk space by compacting the pages based on the specified or existing fill factor setting and reorders the index rows in adjoining pages.

3) PVS

If the PVS size is larger than 50 GB or 10% of the overall database size, active transactions may hold up the PVS cleanup process. Work with customer to identify long running transactions. DBCC OPENTRAN can be used to identify the oldest transaction. Query Performance Insights shows the long running queries. If necessary, run the T-SQL KILL command to end the session.

You can enforce the PVS cleanup by executing the following T-SQL:

```
EXEC sys.sp_persistent_version_cleanup [database_name]
```

Public Doc Reference

blog Article - Azure SQL Database or SQL Managed Instance Database used data space is much larger than expected - Azure SQL Database or SQL Managed Instance Database used data space is much larger than expected - Microsoft Tech Community

Internal Reference

- PVS
- Indexes

Root Cause Classification

/Root Cause: Azure SQL v3/Performance/Space Management/User DB

How good have you found this content?