IO_Troubleshooting

Last updated by | Joao Antunes | Dec 1, 2022 at 12:41 AM PST

Contents

IO Troubleshooting

Issue

This TSG is designed for partners to follow before they engage SQL DB Perf queue for IO issues. You are welcome to contribute to make it better. However, please do not add corner cases or any info related operations that only SQL DB engineering team can do.

IO can be delayed at different levels.

1. **Physical read/writes from/to the underlying storage layer**: For example, for local SSDs, 2 ms per read/write is generally considered as good even though it can be shorter than that. For remote standard storage, delay can be much more significant.
2. **Azure remote storage layer throttling** (this can happen for standard or premium remote storage). Most common one is that the remote blob tier for premium remote storage is set to too low (such as P10) due to small database size.
3. **IO Governance (RG) induced delay:** Depending on IO limit in each SLO, delay induced here will vary.SQL IO RG uses a flat rate model.Let's assume that you are allowed to do 20,000 IOPS (IO per second). If you post 100 IOs at the same time, RG will let first one post immediately, 2nd one wait 1/20,000 sec, third one 2/20,000 ....
4. **SQL Database Engine IO Completion delay**: IO are posted by the requesting worker.But IO completion is not processed by the requesting worker. Instead, IO completion is processed during context switch by a running worker (on that scheduler that is coming off that schedule). SQL has a quantum of 4 ms. So it is possible that a worker can run up to its quantum before yielding, do context switch and check for IO completion even if the IO is complete at OS /storage level.

IO stats

sys.dm_io_virtual_file_stats tracks total read or write delays plus read or write delays induced by RG. But the numbers are accumulative since start of the SQL Database engine. Therefore you will need to do delta to compute the impact.

IO delays & RG IO delays and IO Delays & IO RG IO Delays(TSQL) are example queries.

io_stall_read_ms / io_stall_write_ms measures total latency. Time spent include:

1. Phsyical read/write from/to strorage layer
2. Any Azure Storage layer throttling
3. Time imposed by RG
4. Time spent waiting for context switch (to process IO completion).

io_stall_queued_read_ms / io_stall_queued_write_ms only meassures latency induced by resource governance (RG)

Investigation/Analysis

First Approach

The current TSG presents serveral T-SQL and Kusto queries to troubleshoot IO issues. However before continue using this TSG, please proceed with the following TSG Workflow-for-IO-troubleshooting

The Workflow TSG uses the ASC and has detailed information about serveral important aspects when troubleshooting a IO issue.

Continuing with this TSG

Identification

Top waits will show:

1. PAGEIOLATCH_* (PAGEIOLATCH_SH, PAGEIOLATCH_EX, PAGEIOLATCH_UP). Note that the wait has IO in it. If there is no IO in the page latch wait, it will be different type of problem (like tempdb contention).
2. WRITE_LOG for log IO
3. If the issue is occurring, you can also query sys.dm_exec_requests to look at wait_type and wait_time.

In Managed Instance, if you notice that IOPS/throughput is close to the limits defined in Managed Instance documentation, then you are hitting the limits of the infrastructure where the Managed Instance is running. In addition, if you see a big difference between *read_latency* and *read_io_latency*, or *write_latency* and *write_io_latency* this is indication that you are hitting the performance issues.

You can fix IO performance issues if you are hitting the limits of individual files on General Purpose instance by pre-allocation data or log files. In General Purpose instance, performance of the files depends on the file size as described here.

If you see that you are hitting the limit of the Azure remote files, you can increase the size of the file to get better performance. Use Azure Remote Storage Throttling to identify if there is any Storage layer throttling was happening. Use Azure Remote Storage Throttling Individual files on GP MI to identify if the Managed Instance is hitting the limits of individual files on General Purpose instance.

Mitigation

User consumed all IO

Use Data or Log IO usage or sys.dm_db_resource_stats to identify data and log IO usage. If the data or log IO was above 80%, it means users have used the IO.

Log consumption can be identified by the query #TopLogIOqueries (as below). Data IO and log Mbps aren't directly related; Data IO is IO to the data file, Log Mbps is transactions writing to the log, so for large inserts, it's normal for log Mbps to hit the cap without data IO being high.

Option 1: upgrade SLO to get more IO and memory.

A bigger SLO not only gives user more IO cap but can also have a big memory available which can cache more data and reduces the need for data IO. (For this you could recommend a specialist SLO like M series to go beyond 96Mbps; once on the BC SLOs with 8 or more cores, the LOG rate stays at 96 Mbps)

Option 2: Identify which queries consumed the IO and tune the queries.

- Use Top Data IO queries to identify top data IO consuming queries. to tune
- Use Top Log IO queries and Total log bytes used (user query) to identity top log IO consuming queries to tune.

- Change the application and/or indexing to reduce the data volume that's being written during the load. Turning the large data tables into clustered columnstore indexes (CCI) and using a batch size of greater than 102400 (ideally a million) can significantly reduce the volume; if CCI isn't appropriate, options for a clustered index are using page or row compression, and/or dropping secondary indexes.

User didn't consume all the IO

If user didn't consume all the IO per Data or Log IO usage , explore the following scenarios:

Scenario1: Azure Remote storage throttling:

Use Azure Remote Storage Throttling to identify if there is any Storage layer throttling was happening. If so, engage SQL PG.

Scenario2: User has burst in demand:

Both MonDmRealTimeResourceStats and sys.dm_db_resource_stats log average IO percentage over 15 second period. If customer suddenly had a big request which only lasted 1 second and didn't have much request for rest of the period, that big burst still ended up with waits induced by RG. But average IO percentage may not be that high.

End user can verify using IO Delays & IO Delays induced by RG(TSQL) by modifying the wait for delay to be a very short period of time such as 5 second. Unfortunately our back end telemetry IO delays & IO delays induced by RG is 15 minute interval which is not granular enough to track this.

Solutions are still update SLO or find IO consuming queries to tune.

Scenario3: User query get delayed because of large amount of IO needed:

Assuming neither SQL RG nor Azure remote storage throttled, user can still incur IO delays. Reading from disk can't be 0 ms.If a query requires 200,000 Ios, it will take time to read all the data.

The solutions are still update SLO or tune the query to reduce IO usage.

Root Cause Classification

Performance\DTU Limit\CPU

Query Reference

IOdelays&RGIOdelays

```
// Virtual File Stats IO delays
// every 15 mins
// delta_io_stall_read_ms, delta_io_stall_write_ms are total delays
//delta_io_stall_queued_read_ms, delta_io_stall_queued_write_ms are IO governance induced delay
//
MonSqlRgHistory
| where TIMESTAMP >ago(2h)
| where LogicalServerName =~ "jacklisql" and AppName =~ "d2a790684d31"
| where NodeName == "DB.26"
| where database_id !in (1,2,3,4)
//| where database_id  in (5)
| where event in ("aggregated_virtual_files_io_history")
| project TIMESTAMP,  delta_io_stall_queued_read_ms , delta_io_stall_read_ms, delta_io_stall_queued_write_ms ,
| render timechart
```

IODelays&IORGIODelays(TSQL)

```
/*
This query will compute detal IO stalls
It takes a snapshot of sys.dm_io_virtual_file_stats, waits for 5 minutes (time can be adjusted) and take anoth
*/
SELECT getdate() as runtime, * into #tmp1 FROM sys.dm_io_virtual_file_stats(DB_ID(), null);
waitfor delay '00:05:00'
SELECT getdate() as runtime, * into #tmp2 FROM sys.dm_io_virtual_file_stats(DB_ID(), null);


--Query to Identify IO delays
--delta_io_stall_read_ms, delta_io_stall_write_ms are total delays
--delta_io_stall_queued_read_ms, delta_io_stall_queued_read_ms are delayes induced by IO governance
select
t1.runtime 'StartTime', t2.runtime 'EndTime', T1.database_id, T1.file_id,
(t2.io_stall_read_ms-t1.io_stall_read_ms ) as 'delta_io_stall_read_ms',
(t2.io_stall_write_ms-t1.io_stall_write_ms ) as 'delta_io_stall_write_ms',
(t2.io_stall_queued_read_ms-t1.io_stall_queued_read_ms ) as 'delta_io_stall_queued_read_ms',
(t2.io_stall_queued_write_ms-t1.io_stall_queued_write_ms ) as 'delta_io_stall_queued_write_ms'
from #tmp1 t1 join #tmp2 t2 on t1.database_id=t2.database_id and t1.file_id=t2.file_id
```

TopWaits

```
MonDmCloudDatabaseWaitStats
| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
| top-nested of bin(TIMESTAMP, 5min) by sum(delta_wait_time_ms), top-nested 5 of wait_type by total_wait_time_
| sort by TIMESTAMP asc nulls last
| project TIMESTAMP, wait_type, total_wait_time_ms_per_sec
| render timechart
```

DataorLogIOusage

```
MonDmRealTimeResourceStats
| where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}" and  database_name =~ "{LogicalDatabaseName}"
| where replica_type == 0
| summarize  avg(avg_data_io_percent), avg(avg_log_write_percent) by bin(TIMESTAMP, 1min)
| render timechart
```

TopDataIOqueries

```
MonWiQdsExecStats
    | where LogicalServerName =~ "{LogicalServerName}" and  database_name =~ "{LogicalDatabaseName}"
    | where is_primary  == 1
    | where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
    | summarize total_num_physical_io_reads=sum(num_physical_io_reads) by query_hash
    | top 10 by total_num_physical_io_reads  desc nulls last
```

TopLogIOqueries

```
MonWiQdsExecStats
| where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
| where is_primary == 1
| summarize sum(log_bytes_used) by query_hash
| top 10 by sum_log_bytes_used  desc
```

Totallogbytesused(TSQL)

```
-- top log used
-- please adjust time window by changing rsi.start_time >= DATEADD(hour, -2, GETUTCDATE())
WITH AggregatedLogUsed
AS
(
   SELECT  q.query_hash, SUM(count_executions * avg_cpu_time / 1000.0) AS total_cpu_millisec,
   SUM(count_executions * avg_cpu_time / 1000.0) /SUM(count_executions) as avg_cpu_millisec,
   sum (count_executions*avg_log_bytes_used) as total_log_bytes_used,
   max(rs.max_cpu_time/1000.00) as max_cpu_millisec,
   max(max_logical_io_reads) max_logical_reads,
   COUNT (distinct p.plan_id) AS number_of_distinct_plans,
   count (distinct p.query_id) as number_of_distinct_query_ids,
   sum (case when rs.execution_type_desc='Aborted' then count_executions else 0 end) as Aborted_Execution_Coun
   sum (case when rs.execution_type_desc='Regular' then count_executions else 0 end) as Regular_Execution_Coun
   sum (case when rs.execution_type_desc='Exception' then count_executions else 0 end) as Exception_Execution_
   sum (count_executions) as total_executions,
   min(qt.query_sql_text) as sampled_query_text
   FROM sys.query_store_query_text AS qt JOIN sys.query_store_query AS q
   ON qt.query_text_id = q.query_text_id
   JOIN sys.query_store_plan AS p ON q.query_id = p.query_id
   JOIN sys.query_store_runtime_stats AS rs ON rs.plan_id = p.plan_id
   JOIN sys.query_store_runtime_stats_interval AS rsi
   ON rsi.runtime_stats_interval_id = rs.runtime_stats_interval_id
   WHERE   rs.execution_type_desc in( 'Regular' , 'Aborted', 'Exception') and
    rsi.start_time >= DATEADD(hour, -2, GETUTCDATE())
   GROUP BY  q.query_hash
)
,OrderedLogUsed
AS
(
   SELECT  query_hash, total_log_bytes_used,  number_of_distinct_plans, number_of_distinct_query_ids,  total_e
   Aborted_Execution_Count,Regular_Execution_Count, Exception_Execution_Count, sampled_query_text,
   ROW_NUMBER () OVER (ORDER BY total_log_bytes_used DESC, query_hash asc) AS RN
   FROM AggregatedLogUsed
)
SELECT  * from OrderedLogUsed OD
WHERE OD.RN <=15 ORDER BY total_log_bytes_used DESC

go
```

IODelays&IODelaysinducedbyRG(TSQL)

```
/*
This query will compute detal IO stalls
It takes a snapshot of sys.dm_io_virtual_file_stats, waits for 5 minutes (time can be adjusted) and take anoth
*/
SELECT getdate() as runtime, * into #tmp1 FROM sys.dm_io_virtual_file_stats(DB_ID(N'demo2'), null);
waitfor delay '0:05:0'
SELECT getdate() as runtime, * into #tmp2 FROM sys.dm_io_virtual_file_stats(DB_ID(N'demo2'), null);

--Query to Identify IO delays
--delta_io_stall_read_ms, delta_io_stall_write_ms are total delays
--delta_io_stall_queued_read_ms, delta_io_stall_queued_read_ms are delayes induced by IO governance
select
t1.runtime 'StartTime', t2.runtime 'EndTime', T1.database_id, T1.file_id,
(t2.io_stall_read_ms-t1.io_stall_read_ms ) as 'delta_io_stall_read_ms',
(t2.io_stall_write_ms-t1.io_stall_write_ms ) as 'delta_io_stall_write_ms',
(t2.io_stall_queued_read_ms-t1.io_stall_queued_read_ms ) as 'delta_io_stall_queued_read_ms',
(t2.io_stall_queued_write_ms-t1.io_stall_queued_write_ms ) as 'delta_io_stall_queued_read_ms'
from #tmp1 t1 join #tmp2 t2 on t1.database_id=t2.database_id and t1.file_id=t2.file_id
```

IODelays&IODelaysinducedbyRG

```
// Virtual File Stats IO delays
// every 15 mins
// delta_io_stall_read_ms, delta_io_stall_write_ms are total delays
//delta_io_stall_queued_read_ms, delta_io_stall_queued_write_ms are IO governance induced delay
//
MonSqlRgHistory
| where TIMESTAMP >ago(2h)
| where LogicalServerName =~ "jacklisql" and AppName =~ "d2a790684d31"
| where NodeName == "DB.26"
| where database_id !in (1,2,3,4)
//| where database_id  in (5)
| where event in ("aggregated_virtual_files_io_history")
| project TIMESTAMP,  delta_io_stall_queued_read_ms , delta_io_stall_read_ms, delta_io_stall_queued_write_ms ,
| render timechart
```

AzureRemoteStorageThrottling

```
let srv = 'servername';
let startTime = datetime(2022-11-06 14:00:00);
let endTime = datetime(2022-11-06 19:00:00);
MonSQLXStore
| where TIMESTAMP >= startTime
| where TIMESTAMP <= endTime
| where LogicalServerName =~ srv
| where file_path endswith ".mdf" or file_path endswith ".ndf" or file_path endswith '.ldf'
| project TIMESTAMP, NodeName, AppName, LogicalServerName, event, file_path, page_blob_tier, mapped_errorcode,
```

AzureRemoteStorageThrottlingIndividualFilesonGPMI

```
//Check the individual file allocation
MonDmIoVirtualFileStats
//| where AppName contains '{AppName}'
| where LogicalServerName == '{ServerName}'
| where database_id <> 2
| where TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
| summarize max(TIMESTAMP), max(size_on_disk_bytes) by database_id, file_id, db_name
| extend sizegb = max_size_on_disk_bytes / 1024.0 / 1024.0 / 1024.0
| extend allocgb = iff(sizegb < 128.0, 128.0, sizegb)




//Check if the total allocated storage is hitting the service tier limits
MonDmIoVirtualFileStats
        | where TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
        | where LogicalServerName == '{ServerName}'
        | where database_id <> 2
        | extend logical_database_id = db_name
        | summarize gb = max(size_on_disk_bytes)/1024./1024/1024., used_gb = max(spaceused_mb)/1024. by  bin(T
        | extend allocated_size = iif(gb <= 128., 128., iif(gb <= 256., 256., iif(gb <= 512., 512., iif(gb <=
        | summarize allocated_tb = sum(allocated_size) / 1024. by TIMESTAMP
```

How good have you found this content?