Memory Grant Feedback Persistance and Percentile grant

Last updated by | Soma Jagadeesh | Mar 9, 2023 at 2:37 PM PST

Contents

Overview

Memory Grant is additional memory reserved before execution, for memory-consuming operators (such as sort, join), before their actual usage. This increases reliability, as query with reserved memory is less likely to hit Out-of-memory, hence decreasing IO cycles.

Identifying Issues

When these features do not work well or have adverse effects the manifestation will be like the effect of generic Memory Grant Feedback features. It might affect single query performance or the overall system performance by increasing memory utilization, which may lead to OOM as well. The following are usual symptoms:

On query level, it might spill making the query slow when underestimation happens.
If many plans get overestimated memory, that can lead to OOM errors.
In these cases, if we suspect that new improvements are causing the issues, as a first step, get the query hash/plan hash from ASC report or customer (Performance -> Plans -> Top 5 Plans For Each Cpu Time, Logical Reads and Logical Writes). After getting the query, do the following to verify and mitigate the issue. In case of OOM, first do the analysis for the database as described below, instead of a specific plan.

Investigation/Analysis

Describes the steps/queries to use for confirming the issue

Analyzing a plan for potential issues

Find the query/plan from ASC (Performance -> CPU) which is taking large CPU cycles. This could mean that the query has less memory and hence taking time to spill. Follow Analyze for a database to check for spills. Use below query to check if taking a huge memory grant, to quickly check if the grant is incorrect.

Top 10 queries with largest mem ory grants (cpu times etc) in incident time period:

```
let nname =   <node name>;
let aname =   <app name>;
let startTime =   <start time>;
let endTime =   <end time>;
MonWiQdsExecStats
|  where  TIMESTAMP > startTime and  TIMESTAMP < endTime
|  where  NodeName == “node name” and  AppName == "app name"
|  summarize  sum(execution_count),  sum(cpu_time),  sum(max_query_memory_pages)  by  query_id,  plan_id
|  project  query_id,  plan_id,  sum_execution_count,  sum_max_query_memory_pages
|  order  by  sum_max_query_memory_pages
|  take 10
```

Check for Spill

If the following query returns rows, that means query is spilling.

```
let sname =   <server name>;
let dname =   <db name>;
let aname =   <app name>;
let phash = tolong(<hex plan hash>);
let startTime =   <start time>;
let endTime =   <end time>;
MonQueryProcessing
|   where   originalEventTimestamp  between(startTime..endTime)
|   where   AppName == aname
|   where   logical_database_name == dname
|   where   LogicalServerName == sname
|   where   query_plan_hash_signed == phash
|   where   event ==   "memory_grant_updated_by_feedback"
|   mv-expand   parse_json(memory_summary_by_operator_kb)
|   extend   ni = parse_json(memory_summary_by_operator_kb)
|   where   tolong(ni.Granted) == tolong(ni.Used)   and   tolong(ni.Granted) >   1024
|   project   TIMESTAMP, ideal_additional_memory_before_kb, ideal_additional_memory_after_kb, memory_summary_b
```

Check for Overestimation

If the following query returns rows, check overestimate_kb to find out if this might be causing the issue.

```
let sname =   <server name>;
let dname =   <db name>;
let aname =   <app name>;
let phash = tolong(<hex plan hash>);
let startTime =   <start time>;
let endTime =   <end time>;
MonQueryProcessing
|   where   originalEventTimestamp  between(startTime..endTime)
|   where   AppName == aname
|   where   logical_database_name == dname
|   where   LogicalServerName == sname
|   where   query_plan_hash_signed == phash
|   where   event ==   "memory_grant_updated_by_feedback"
|   extend   overestimate_kb = ideal_additional_memory_before_kb - ideal_additional_memory_after_kb
|   where   overestimate_kb >   0
|   project   TIMESTAMP, overestimate_kb, ideal_additional_memory_before_kb, ideal_additional_memory_after_kb
```

Cprrelate with MGF Persistence

Check if the plan is using persistent feedback. If the following query returns any rows, it is using persistent feedback.

```
let sname =   <server name>;
let dname =   <db name>;
let aname =   <app name>;
let phash =   <hex plan hash>;
let startTime =   <start time>;
let endTime =   <end time>;
MonWiQueryParamData
|   where   originalEventTimestamp between(startTime..endTime)
|   where   AppName == aname
|   where   logical_database_name == dname
|   where   LogicalServerName == sname
|   where   event ==   "query_parameterization_data"
|   where   query_plan_hash == phash
|   where   mgf_persistence_used_status ==   'FeedbackApplied'
```

Correlate with MGF Percentile Grant

Check if the plan is using percentile grant with the following query and get details about spill count and overestimate in KB.

```
let sname = <server name>;
let dname = <db name>;
let aname = <app name>;
let phash = tolong(<hex plan hash>);
let startTime = <start   time>;
let endTime = <end   time>;
MonQueryProcessing
|   where   originalEventTimestamp  between(startTime..endTime)
|   where   AppName == aname
|   where   logical_database_name == dname
|   where   LogicalServerName == sname
|   where   query_plan_hash_signed == phash
|   where   event ==   "memory_grant_feedback_percentile_grant"
|   summarize   countExec = sum(current_interval), countSpill = sum(spill_count_pg), totalOverestimate_kb = su
```

Analyze for Database

While we look at a specific plan when that plan is running slow, generally overestimating many plans lead to OOM in system level. In those cases, we need to check on database level. As MGF persistence is very unlikely to be cause for OOM, here we mostly focus on MGF percentile grant for possible root cause.

```
let sname = <server name>;
let dname = <db name>;
let aname = <app name>;
let startTime = <start   time>;
let endTime = <end   time>;
MonQueryProcessing
|   where   originalEventTimestamp  between(startTime..endTime)
|   where   AppName == aname
|   where   logical_database_name == dname
|   where   LogicalServerName == sname
|   where   event ==   "memory_grant_feedback_percentile_grant"
|   summarize   countExec = sum(current_interval), countSpill = sum(spill_count_pg), totalOverestimate_kb= sum
```

If the total overestimate value is very high, that indicates that MGF percentile is causing the issue. In that case, summarize above at query_plan_hash_signed level and spot the culprit plans and queries taking large grants.

Mitigation

- The quick mitigation if the issue is too frequent for the customer, is to disable feature at server level using below commands.
- If there is OOM seen at database level in the above analysis, disable the feature at database level using below commands.
- If any specific plan/query is causing slowdown, then disable the feature at query level using below commands.

1. Issues related to MGF Persistence We can apply the mitigation on different levels according to the affected level.

- Disable the feature on server level

```
 Set-ServerConfigurationParameters -ServerName testsvr -Parameters
@('SQL.Config_FeatureSwitches_MemoryGrantFeedbackPercentileGrant ') -Values @('off')
```

- Disable the feature on database level `ALTER DATABASE SCOPED CONFIGURATION SET MEMORY_GRANT_FEEDBACK_PERSISTENCE = OFF`
- Disable the feature for the query using following hint `USE HINT ('DISABLE_MEMORY_GRANT_FEEDBACK_PERSISTENCE')`

2. Issues related to MGF Percentile Grant We can apply the mitigation on different level.

- Disable the feature on server level

```
 Set-ServerConfigurationParameters -ServerName testsvr -Parameters @('SQL.Config_FeatureSwitches_
MemoryGrantFeedbackPercentileGrant') -Values @('off')
```

- Disable the feature on database level `[ALTER DATABASE SCOPED CONFIGURATION SET MEMORY_GRANT_FEEDBACK_PERCENTILE_GRANT = OFF]()`

- Mitigation on query level We don’t have any hint to disable percentile grant to on query level. But if server level/database level mitigation is not possible, we can try to limit the effect of overestimation using the following hint. The percentage indicates the amount of calculated memory grant that will be allocated. So, adjust the percentage accordingly.

```
USE HINT (MAX_GRANT_PERCENT = 20)
```

RCA Template (optional)

- POST Verification

For recurring [by design] issues where there is an RCA template for the engineer to customize with timestamps and server details, provide that here.

More Information (optional)

After disabling the feature at the required level, keep track of the performance of system through the health properties/ customer.

Root Cause Classification

Azure SQL DB/Performance/ Memory Grant