Memory Grant Feedback Persistence and Percentile grant

Last updated by | Ricardo Marques | Mar 20, 2023 at 10:56 AM PDT

Contents

What are Memory Grant Feedback Persistence and Percentile grant

This features are documented publicly on Query processing feedback features.

In a nutshell, what this features does is to use the data of previous executions and use it for estimating the memory grant of new executions.

Make sure that you look into the public documentation for more details.

How they can help with query performance

When the plan is compiled, an estimation of the memory to be used for que query will be determined. And this information will be present on the execution plan, like mentioned on the wiki article Execution Plans.

Two not wanted situations can result from this:

- the memory grant is overestimated. Memory is wasted and concurrency impacted.
- the memory grant is underestimated. The query will spill. Check What to look for on an execution plan and Troubleshoot Out Of Memory errors

This feature increases reliability, as query with reserved memory is less likely to hit out-of-memory, hence decreasing IO cycles.

Some operators like Hash Joins, Merge Joins and Sorting can take large parts of memory.

How they can contribute for a degradation of performance

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**).

Kusto Queries for Memory Grant Feedback Persistence and Percentile grant troubleshooting

Top 10 queries with largest memory grants 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
```

Correlate with Memory Grant Feedback Persistance

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 Memory Grant Feedback Percentile Grant

Check if the plan is using percentile grant with 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 a 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 Memory Grant Feedback persistence is very unlikely to be cause for OOM, here we mostly focus on Memory Grant Feedback percentile grant for possible root cause.

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

```
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
```

Mitigation

The quick mitigation if the issue is too frequent for the customer, is to disable feature.

If there is OOM seen at database level in the above analysis, disable the feature at database level.

Disable Memory Grant Feedback Persistence at database level:

```
ALTER DATABASE SCOPED CONFIGURATION SET MEMORY_GRANT_FEEDBACK_PERSISTENCE = OFF
```

Disable Memory Grant Feedback Persistence for a query using query hints:

```
USE HINT ('DISABLE_MEMORY_GRANT_FEEDBACK_PERSISTENCE')
```

Disable Memory Grant Feedback Percentile Grant at database level:

```
ALTER DATABASE SCOPED CONFIGURATION SET MEMORY_GRANT_FEEDBACK_PERCENTILE_GRANT = OFF
```

Disable Memory Grant Feedback Percentile Grant for a query using query hints:

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

This ALTER DATABASE statements are available on the public documentation

At server level the feature can only be disabled by the PG using CAS commands.

How good have you found this content?