Query Tuning Troubleshooting

Last updated by | Peter Hewitt | Oct 18, 2022 at 11:48 AM PDT

Contents

Query Tuning Troubleshooting

Issue

This guide has steps to follow for troubleshooting Query Tuning issues before engaging the SQL DB Performance queue. 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.

Investigation/Analysis

1. Verify if the query has required indexes: Find the most optimum query plan from the first Kusto results and check if it has any missing index hint in the query plan xml.

- Execute this SQL query to obtain the query plan xml:

```
select query_plan from sys.query_store_plan where plan_id = <>
```

- Save the result as .sqlplan and open in SSMS (Sql Server Management Studio); you can check the missing index hint by searching for the keyword "missing" in any xml editor.
- Test the index is being picked up by the query plan for that query on Stage; after verifying add the index to CMS schema.

2. Look at the operators where most of the cost spend in the query plan; optimize the query accordingly to minimize the cost. If most of the cost is in Key Lookup operator, than those columns can be added as included columns to the index being seek; this will remove the key lookup operator cost.

3. Remove the IS NULL OR anti- patterns: "IS NULL OR " should not be used in the Where clause, it will prevent using the index on the column. The "IS NULL" check should be done in C# not in TSQL, look at examples in core FSM's.

4. If most of the cost of the Query plan is spend on the Join then:

- Divide the query into two small queries with strict filters set in the where clause (only use AND in the where clause).
- Union all the result set of the small sub queries instead of large joins on unnecessary data.

5. If query plan are the same, but performance is different, remember to check if there is code package change using the following Kusto query:

```
MonSQLSystemHealth
| where LogicalServerName =~ "<>"
| where AppName == "<>"
| distinct code_package_version
```

6. You can check the Query Store (QDS) using the Kusto query below to get information about the most consuming CPU queries, and ask the customer for the execution plans of the most consuming queries (using query hash or query plan hash).

```
let  varLogicalServerName = "<servername>";
//let varNodeName = "DB.116";
let  startTimestamp = ago(48h); //datetime(2022-07-06 11:00);
let  endTimestamp = now(); //datetime(2022-07-10 11:30);
let  totalCpu_ms = toscalar (
MonWiQdsExecStats
| where  TIMESTAMP > startTimestamp  and  TIMESTAMP < endTimestamp
| where  LogicalServerName =~ varLogicalServerName
| summarize  TotalQueryCPU_ms = sum(cpu_time)
|project  TotalQueryCPU_ms/1000.0
);
MonWiQdsExecStats
| where  TIMESTAMP > startTimestamp  and  TIMESTAMP < endTimestamp
| where  LogicalServerName =~ varLogicalServerName
| summarize  TotalQueryCPU_ms = sum(cpu_time) / 1000.0, TotalQueryCPU_Pct = (100.0*(sum(cpu_time) / 1000.0)) /
total_executions=sum(execution_count) by  query_hash, query_plan_hash,database_name
| top 10 by  TotalQueryCPU_ms  desc
```

7. You can check the QDS to get information about memory consumption and memory clerks, to check in a graphical way the memory usage and pressure of the server.

```
let  varLogicalServerName = "<servername>";
let  startTimestamp = datetime(2022-09-10 11:40);
let  endTimestamp =datetime(2022-09-14 08:20);
//Mem clerks
let  AggInterval_do_NOT_change=time(5m);
MonSqlMemoryClerkStats
| where  TIMESTAMP > startTimestamp  and  TIMESTAMP < endTimestamp
and  LogicalServerName =~ varLogicalServerName
| summarize  MemoryInMB=sum(pages_kb+vm_committed_kb)/1024 by  bin(TIMESTAMP,AggInterval_do_NOT_change), memor
| project  TIMESTAMP, MemoryInMB,memory_clerk_type
| where  memory_clerk_type  in ("MEMORYCLERK_SQLQERESERVATIONS", "CACHESTORE_SQLCP", "MEMORYCLERK_SQLBUFFERPOO
| order  by  TIMESTAMP  asc
| render  timechart
```

8. Always check ASC performance tab to get information about Spin locks, waits, query blocks, timeouts, errors, memory grants, etc. This information is very useful to set up a performance troubleshoot.

Root Cause Classification

Performance\Specific Query Slow

How good have you found this content?