Sub Core SLO

Last updated by | Peter Hewitt | Oct 6, 2022 at 10:15 AM PDT

Contents

Issue

If the Service Level Objective (SLO) of the database is equal to S2 or lower, it is considered a Sub-Core SLO. The Basic, S0, S1 and S2 service objectives provide less than one vCore (CPU). For CPU-intensive workloads, a service objective of S3 or greater is recommended. See Sub-Core SLO.

If the customer database is on S2 or lower and the workload is high, they can easily hit the Sub-Core SLO limits. Due to the limits being reached, customers can face the following issues:

1. Database Availability/Connectivity Issues (Error 40613)
   a. Reaching resource limits on Sub-Core SLO's can cause Out Of Memory (OOM) issues which are often accompanied by severe performance issues, especially when resource limits are hit. These other performance issues can trigger failover and cause unavailability. Please note that these failovers will continue to happen untill the workload gets processed and we are out of the OOM loop, thus the period of unavailability can be high.
2. Missing Metrics
   a. The database may remain active, but you may see metrics missing from the Azure Portal. This is due to lack of available memory for telemetry processes.
3. Performance Issues
   a. As the resource limit threshold has already been reached, this may impact the overall performance of the database and cause query slowness/timeouts.

Investigation/Analysis

1. Check for the Sub-Core SLO Insight in the SQL Troubleshooter > Performance > Insights tab. If this insight is triggered it means that the customer is hitting the resource limits on the Sub-Core SLO and can face the isues mentioned above.

The insight should look similar to the below:

```
Resource limits Hit(Sub-core)   Warning The database has hit resource limits on subcore SLOs, during the input
```

For reference, the insight uses the following Kusto query:

```
// Note:
//   This check is built on the facts that there are incidents/complaints from customers.
//
let p_check_time_window = 3h;
// If the high utilization lasts more than 1/6 of the overall time period (avg spike 1min every 6min), then we
let p_high_time_ratio = 1.0/6;
// Besides the above cases, if there was any consecutive high usage longer than 10min, we think it is a strong
let p_high_consecutive_time_window = 10m;
//
// Part 1: Check the user-group (DTU) resource limit hit cases, for cpu, IO, log_writes, workers, sessions, or
//
let p_user_sample_window = 30s;
let p_high_single_consumption = 90.0;
// 30% is coming from 99 (over all apps) of 99.9 percentile (over 7 days) for sub-core apps in Prod.
let p_high_workers = 30.0;
// If workers are high, also check cpu and IO usage to be greater than 60% to have stronger confidence.
let p_high_workers_correlated_consumption = 60.0;
let p_high_multi_dimensions = 180.0;
let usergroup_usage = MonDmRealTimeResourceStats
| where TIMESTAMP > ago(p_check_time_window)
| where AppName == '{AppName}'
| where replica_type == 0
// Max over nodes in case for failovers
| summarize cpu = max(avg_cpu_percent),io = max(avg_data_io_percent),log_writes = max(avg_log_write_percent),
            by AppName, database_name, bin(TIMESTAMP, p_user_sample_window)
| extend high_cpu = cpu > p_high_single_consumption,high_io = io > p_high_single_consumption,high_log_writes =
         high_sessions = sessions > p_high_single_consumption, high_workers = workers > p_high_workers and(cpu
         high_multiD = cpu + io + log_writes > p_high_multi_dimensions
| summarize cnt = count(),high_cpu = countif(high_cpu),high_io = countif(high_io),high_logs = countif(high_log
            high_workers = countif(high_workers), high_multiD = countif(high_multiD) by AppName, bin(TIMESTAMP
| extend consecutive_high_cpu = high_cpu >= cnt,consecutive_high_io = high_io >= cnt,consecutive_high_logs = h
            consecutive_high_workers = high_workers >= cnt, consecutive_high_multiD = high_multiD >= cnt
| summarize cnt = 1.0 * sum(cnt),high_cpu = sum(high_cpu),high_io = sum(high_io),high_logs = sum(high_logs),hi
            high_multiD = sum(high_multiD), consecutive_high_cpu = countif(consecutive_high_cpu), consecutive_
            consecutive_high_sessions = countif(consecutive_high_sessions),consecutive_high_workers = countif(
            min(TIMESTAMP), max(TIMESTAMP) by AppName
| extend CPU = high_cpu / cnt > p_high_time_ratio or consecutive_high_cpu > 0,IO = high_io / cnt > p_high_time
         LogWrites = high_logs / cnt > p_high_time_ratio or consecutive_high_logs > 0,Sessions = high_sessions
         Workers = high_workers / cnt > p_high_time_ratio or consecutive_high_workers > 0,MultiDimensions = hi
| where CPU or IO or LogWrites or Sessions or Workers or MultiDimensions
| project AppName, tolong(CPU), tolong(IO), tolong(LogWrites), tolong(Sessions), tolong(Workers), tolong(Multi
// Hide MultiDimensions if any single dimension is very high.
| extend MultiDimensions = iff(CPU == 1 or IO == 1 or LogWrites == 1, 0, MultiDimensions);
//
// Part 2: Check high instance cpu usage and memory pressure (oom or kernel high)
//
let p_sample_window = 1m;
let p_high_instance_consumption = 95.0;
// This 40%(a really high value) comes from 99.9 percentile (over apps) of 99 percentile of Prod sub-core apps
//   For relatively larger cpu caps (e.g. 120 for S2 on Gen5), this kernel high may not have big impacts.
//   Then using this 40% means we somehow skip the check for them.
let p_kernel_threshold = 40.0;
// If we saw two ooms due to user/process, then we think it did have memory pressures.
let p_oom_cnt = 2;
let p_oom_sample_window = 5m;
let instance_cpu_mem_usage = MonRgLoad
| extend TIMESTAMP = originalEventTimestampFrom
| where TIMESTAMP > ago(p_check_time_window)
| where event == 'instance_load' and code_package_name == 'Code'
| where application_name contains '{AppName}' and application_type == 'Worker.ISO'
| extend AppName = extract('fabric:/Worker.ISO/(.+)', 1, application_name)
| extend external_pressure = (memory_current_cap != memory_load_cap) or (memory_load*1.0/ memory_load_cap < 0.
| summarize cpu_usage = max(cpu_load * 100.0 / cpu_load_cap), kernel_usage = max(kernel_load*100.0/cpu_load_ca
| summarize cnt = count(), high_instance_cpu = countif(cpu_usage > p_high_instance_consumption), high_kernel =
  by AppName, bin(TIMESTAMP, p_high_consecutive_time_window)
| extend consecutive_high_instance_cpu = high_instance_cpu >= cnt, consecutive_high_kernel = high_kernel >= cn
| join kind = leftouter (
```

```
      // OOM events, check if the OOM is at process level or user pool level.
      //
      MonSQLSystemHealth
      | where TIMESTAMP > ago(p_check_time_window)
      | where AppName == '{AppName}'
      | where (message contains 'MSR' and message contains 'Memory Manager') or (message contains 'Error: 701' a
      | project TIMESTAMP, AppName, NodeName, message
      | extend UserPoolPressure = iff(message contains 'Error: 701', 1, 0)
      | extend OOMFactor = extract('Last OOM Factor(.+)', 1, message, typeof(long))
      | summarize process_pressure = countif(OOMFactor == 1), pool_pressure = countif(OOMFactor == 5), user_pool
      | where process_pressure > 0 or (pool_pressure > 0 and user_pool > 0)
      | summarize oom_count = count() by AppName, bin(TIMESTAMP, p_high_consecutive_time_window)
  ) on TIMESTAMP, AppName
  | extend exclude_pressure = iff(external_pressure > 0, 0, 1)
  | summarize cnt = sum(cnt) * 1.0, consecutive_high_kernel = countif(consecutive_high_kernel and exclude_pressu
    consecutive_high_instance_cpu = countif(consecutive_high_instance_cpu), high_instance_cpu = sum(high_instanc
    oom_cnt = sum(oom_count * exclude_pressure), min(TIMESTAMP), max(TIMESTAMP) by AppName
  | extend Memory = consecutive_high_kernel > 0 or high_kernel / cnt > p_high_time_ratio or oom_cnt >= p_oom_cnt
    InstanceCpu = consecutive_high_instance_cpu > 0 and high_instance_cpu / cnt > p_high_time_ratio
  | where Memory or InstanceCpu
  | project AppName, tolong(Memory), tolong(InstanceCpu), StartTime2 = min_TIMESTAMP, EndTime2 = max_TIMESTAMP;
usergroup_usage
  | join kind = fullouter (
     instance_cpu_mem_usage
  ) on AppName
  | extend StartTime1= iff(isnotempty(StartTime1) and isnotnull(StartTime1), StartTime1, now()), EndTime1 = iff(
  | extend StartTime2= iff(isnotempty(StartTime2) and isnotnull(StartTime2), StartTime2, now()), EndTime2 = iff(
  | extend StartTime = iff(StartTime1 > StartTime2, StartTime2, StartTime1), EndTime = iff(EndTime1 > EndTime2,
  | project-away AppName, AppName1, StartTime1, StartTime2, EndTime1, EndTime2
```

Mitigation

The recommended solution for the customer is to scale up the database to a higher SLO.

Option 1 (recommended): Upgrade the database to a higher SLO
a. If the customer is unable to scale-up, we recommend that they first reduce the workload, wait for some time for the resource limits to subside and then retry the scale-up operation.
Explanation: There can be scenario's where the database is not available and the scale up operation fails. Under these circumstances **do not create an IcM Incident** as there is not much that the product group can do in this case either. The customer has to stop any new workload and let the workload submitted be processed, so that the database can come out of recurring Out Of Memory (OOM) situations and failovers can be avoided.

Option 2: Review and share the top resource consuming queries with the customer. The customer can potentially tune these queries to reduce resource consumption.

Option 3: Reduce the workload demands on the database.

Public Doc Reference

Sub-Core SLO

Internal Reference

186824565
208647201
208543056

Root Cause Classification

Cases resolved by this TSG should be coded to the following root cause:
Workload Performance/User Issue/Error/Throttling errors/Resource Limit

How good have you found this content?