Useful Kusto Queries for Performance

Last updated by | Lisa Liu | Nov 6, 2020 at 10:35 AM PST

Useful Kusto Queries for Performance

Thursday, May 11, 2017
2:33 PM

This goal of this document help to list out all the interesting point you can check during the performance investigation. There is no step by step check you can follow. To investigate the perf you need to consolidate the information and customer issue together for finding out root cause.

CPU

The high CPU could result slow performance. If the server is under high CPU, we need to firstly understand what trigger the high CPU. If it is trigger by customer workload, we normal sugge promote to higher SKU. Otherwise, we need to do understand what goes wrong.

To get the CPU percentage of the server, run following query

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonResourceMetricsProvider
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| project originalEventTimestamp, cpu_load_percent
| render timechart
```

IOPS

High IOPS(input/output per second) can cause Performance degradation due to slow reads and writes to disks, If it is trigger by customer workload, we normal suggest customer promote t Otherwise, we need to do understand what goes wrong.

To get the IOPS percentage of the server, run following query

PFS:

```
MonRdmsServerMetrics
| where LogicalServerName == "{ServerName}"
| where TIMESTAMP >= ago(7d)
| where metric_name contains "io_consumption_percent"
| summarize max(metric_value) by bin(originalEventTimestamp, 1m)
```

XIO:

```
let TimeCheckStart = datetime('08/11/2020 09:34:00');
let TimeCheckEnd = datetime('08/14/2020 09:34:00');
let TimeRange = TimeCheckEnd - TimeCheckStart;
let Intervals = iff(TimeRange <= 4h, 1s, iff(TimeRange <= 25h, 5s, 30s));
let Throttling = MonSQLSbs
| where originalEventTimestamp >= TimeCheckStart and originalEventTimestamp <= TimeCheckEnd
| where LogicalServerName =~ 'pg_server_name' and event == 'xio_failed_request' and errorcode == 'ServerBusy';
Throttling
| extend mdf_read_cnt = iff(file_path hassuffix 'sbs.mdf' and request_type == 'XIOTypeRead', 1, 0)
| extend mdf_write_cnt = iff(file_path hassuffix 'sbs.mdf' and request_type == 'XIOTypeWrite', 1, 0)
| extend ldf_read_cnt = iff(file_path hassuffix 'sbs_log.ldf' and request_type == 'XIOTypeRead', 1, 0)
| extend ldf_write_cnt = iff(file_path hassuffix 'sbs_log.ldf' and request_type == 'XIOTypeWrite', 1, 0)
| project Timestamp = originalEventTimestamp, mdf_read_cnt, mdf_write_cnt, ldf_read_cnt, ldf_write_cnt
| summarize mdf_read = sum(mdf_read_cnt),
mdf_write = sum(mdf_write_cnt),
ldf_read = sum(ldf_read_cnt),
ldf_write = sum(ldf_write_cnt) by bin(Timestamp, 1s)
| summarize ['data file read beyond limit'] = max(mdf_read),
['data file write beyond limit (no perf impact)'] = max(mdf_write),
['log file read beyond limit'] = max(ldf_read),
['log file write beyond limit'] = max(ldf_write) by bin(Timestamp, Intervals)
| sort by Timestamp asc
```

Memory

The high memory is not considered as an issue if it did not hit 100% and through the out of memory error which may result slow performance or reconfiguration. We usually need to correl usage with reboot, customer login and sandbox log to understand what contribute the high memory and if it is really impact the performance.

To get the memory and memory cap of the server, run following query

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonResourceMetricsProvider
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| project originalEventTimestamp, memory_used_mb , memory_used_mb_cap
| render timechart
```

Query Load and Pattern

The number of query could be an important factor of performance. The high number of query might (or might not) result slow performance/high CPU/high IO.

To get the total number for query of the server, run following query

```
MonRdmsServerMetrics
| where TIMESTAMP >= datetime({StartTime}) and TIMESTAMP <= datetime({EndTime})
| where LogicalServerName =~ "{ServerName}"
| where metric_name == "Queries"
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
```

You can go further query to get the breakdown of each kind of query

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonRdmsServerMetrics
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| where metric_name contains "Com_"
| where metric_value != 0
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

Slow Query

The slow query log (https://docs.microsoft.com/en-us/azure/mysql/concepts-server-logs) is an important log for performance analysis. If customer provide the slow query log, you can find entries.

```
# Time: 140905 6:33:11
# User@Host: dbuser[dbname] @ hostname [1.2.3.4]
# Query_time: 0.116250 Lock_time: 0.000035 Rows_sent: 0 Rows_examined: 20878 use dbname; SET timestamp=1409898791;
...SLOW QUERY HERE...
```

Even if customer is not enable show query log, we can still see the metric for how many query exceed the long_query_time.

To get the slow query count of the server, run following query

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonRdmsServerMetrics
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| where metric_name contains "Slow_queries"
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

Concurrent Connection

Difference SKU has different max connection setting (https://docs.microsoft.com/en-us/azure/mysql/concepts-limits). If the max connection is hit, the server will encounter the connectivi concurrent connection might indicate the server is under heavy load too.

To get the concurrent connection count of the server, run following query

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonRdmsServerMetrics
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| where metric_name contains "active_connections"
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

Attempt Connection

Attempt connection means how many connection is issued to the server. You can see attempt connection from two place.

From the MySQL metric:

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonRdmsServerMetrics
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| where metric_name contains "Connections"
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

From the Gateway log

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonLogin
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where logical_server_name == ServerName
| where AppTypeName == "Gateway.MySQL" and event == "process_login_finish"
```

```
| summarize count() by bin(originalEventTimestamp, 1m)
| render timechart
```

Backup

SBS Full/Diff backup might impact sever performance. You can check the backup log by following query (If Backup was running during the perfromance degradation)

```
let StartTime = ago(2d);
let EndTime = ago(1d);
let ServerName = "howang57standard-westeurope";
MonBackup
| where originalEventTimestamp > StartTime and originalEventTimestamp < EndTime
| where LogicalServerName == ServerName
| where database_name == "SBS"
| project originalEventTimestamp, backup_type, event_type, ['details']
```

Disk Throttling

```
number of throttling events on a specific db file more than 40 throttles considered high
MonSQLSbs
| where (LogicalServerName =~ "{ServerName}" )
| where TIMESTAMP >= ago(1d)
| where file_path contains "sbs.mdf" or file_path contains "sbs_log.ldf"
| where request_type == "XIOTypeWrite" or request_type == "XIOTypeRead"
| where errorcode == "ServerBusy"
| summarize request_count=count() by bin(TIMESTAMP, 5s), file_path
| sort by TIMESTAMP asc
| render timechart
```

latency establishing a connection with the server MySQL

```
Latencies more than 250 milliseconds are considered high and need troubleshooting
MonLogin
| where TIMESTAMP >= ago(7d)
| where (logical_server_name =~ "{ServerName}" )
| where event == "process_login_finish" or event == "connection_accept"
| where AppTypeName == "Gateway.MySQL" or AppTypeName == "Host.MySQL"
| where is_success == true
| where connection_id != "00000000-0000-0000-0000-000000000000"
| summarize start=min(originalEventTimestamp), end=max(originalEventTimestamp) by connection_id, logical_server_name
| extend duration = bin(end - start, 1tick)
| summarize latency_seconds=tolong(avg(duration))/10000000.0 by TIMESTAMP=bin(start, 1m), logical_server_name
```

Server Instance Health

Purpose: Gives a view of health of the server, mainly storage usage, connection counts and other general health

```
MonRdmsServerMetrics
| where LogicalServerName =~ "{ServerName}"
| project TIMESTAMP, metric_name, metric_value
| evaluate pivot(metric_name, sum(metric_value))
```

More performance metrics

```
MonCounterFiveMinute
| where ClusterName == "cr1.southeastasia1-a.control.orcasql-seas1-a.mscds.com"
| where TIMESTAMP > ago(4d)
| where NodeName == "MN.2"
| where AppName == "Gateway.PG"
| where CounterName contains "Working Set - Private"
| summarize by TIMESTAMP, CounterValue
```

Count of Executions by type of query

```
MonRdmsServerMetrics
| where LogicalServerName == "{ServerName}"
//| where TIMESTAMP >= ago(48h)
| where metric_name in ("Com_insert", "Com_update", "Com_select")
| summarize sum(metric_value) by bin(originalEventTimestamp, 1m), metric_name
| render timechart
```

Average metric_value

```
MonRdmsServerMetrics
| where LogicalServerName =~ "{ServerName}"
| where metric_name contains "io_consumption_percent"
| summarize avg(metric_value) by bin(originalEventTimestamp, 1h)
```

CPU consumption

```
MonResourceMetricsProvider
| where LogicalServerName =~ "{ServerName}"
| summarize avg(cpu_load_percent) by bin(PreciseTimeStamp, 1h)
```

Memory consumption

```
MonResourceMetricsProvider
| where LogicalServerName =~ "{ServerName}"
| summarize avg(working_set_percent) by bin(PreciseTimeStamp, 1h)
```

IOPS

```
MonResourceMetricsProvider
| where LogicalServerName =~ "{ServerName}"
| summarize avg(total_iops), avg(disk_read_iops), avg(disk_write_iops) by bin(PreciseTimeStamp, 1h)
```

//Correlate Query Count with CPU/Resource usage. Make sure the chart options have Separate Y-axis checked.
//Stats mildly skewed due to not exact timestamps but should be within the same minute.

```
let ServerName = "markhamdistrictenergy";
let StartTime = datetime(2017-08-21 19:00);
let EndTime = datetime(2017-08-21 21:30);
union(
MonResourceMetricsProvider
| where TIMESTAMP between(StartTime..EndTime)
| where LogicalServerName =~ ServerName
| extend cpu_percentage = (cpu_load/cpu_load_cap)*100, memory_percentage = (memory_used_mb/memory_used_mb_cap)*100
| project TIMESTAMP, cpu_percentage, memory_percentage, disk_read_iops, disk_write_iops
),(
MonRdmsServerMetrics
| where TIMESTAMP >= ago(7d)
| where LogicalServerName =~ ServerName
| where metric_name == "Queries"
| project TIMESTAMP, ["Query_Count"]=metric_value
)
| render timechart
```

Logins to the MySQL server

```
MonLogin
| where AppTypeName == "Gateway.MySQL"
| where logical_server_name == "loopnewstestdb"
| where event == "process_login_finish"
| where originalEventTimestamp > ago(24h)
MonLogin
| where logical_server_name == "loopnewstestdb"
| where originalEventTimestamp > ago(1d)
| summarize count() by bin(originalEventTimestamp, 1h), AppTypeName, event
| render timechart
```

Check the Storage usage

```
MonRdmsServerMetrics
| where TIMESTAMP between(datetime(2018-01-18 10:00) .. datetime(2018-01-18 21:00))
| where LogicalServerName =~ "azure-vindicia-dwdev"
| where metric_name == "storage_used"
| project TIMESTAMP, ["storage_used"]=metric_value
| render timechart
```

Appname and Logical Server Name (super set of all 3 columns would give you info about the servers which had recent logins)

```
let logins_after =  ago(7d);
MonLogin
                | where AppTypeName in ( "Gateway.PG" , "Host.PG" , "Worker.PAL.PG","Gateway.MySQL" , "Host.MySQL" , "Worker.PAL.MySQL")
                | where SourceMoniker contains "Prod"
                | where event == "process_login_finish" and error != 17830
                | where originalEventTimestamp > logins_after
                |  extend outcome = iff(is_success == 1, "Success", iff(is_user_error == 1, "User Error", "System Error"))
                |  project PreciseTimeStamp, ClusterName, logical_server_name, error, ['state'] , outcome, total_login_time_ms , total_time_ms, SourceM
LogicalServerName, server_name
            | summarize count() by logical_server_name,  LogicalServerName, server_name
```

Created with Microsoft OneNote 2016.

How good have you found this content?