PostgreSQL Performance TSG

Last updated by | Hamza Aqel | Jan 17, 2022 at 4:19 AM PST

Please don't modify or move as this is part of GT , please contact haaqel@microsoft.com if needed

PostgreSQL Login Latency TSG

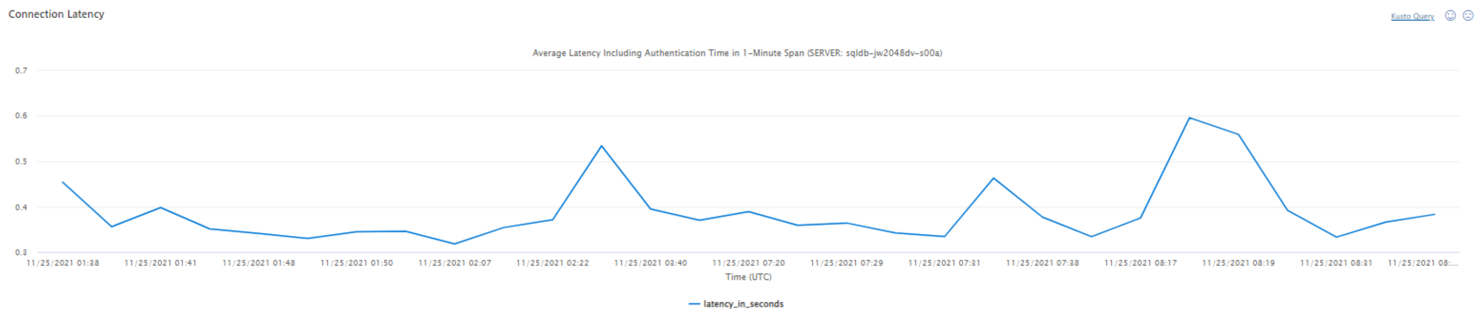
Tuesday, August 6, 2019
11:39 AM

Due to the architectural choice in Single Server PostgreSQL, creating a new connection is a resource (CPU) intensive operation and can take roughly 200-400 milliseconds. This can impact your workload performance. Many customers experience a slowdown in the application, especially when they migrate the PostgreSQL database from on-premise, or IaaS (VM), or from competitive public cloud to Azure PostgreSQL. Common scenarios are described along with recommended best practices.

Connections to Single Server need to be carefully managed. For example, creating a new connection is a resource (CPU) intensive operation and can take roughly 200-400 milliseconds. Similarly, a PostgreSQL connection, even idle, can occupy about 10MB of memory. This can impact your workload performance.

Step1 :

check the connection latency , you can check that from ASC (Connectivity tab):

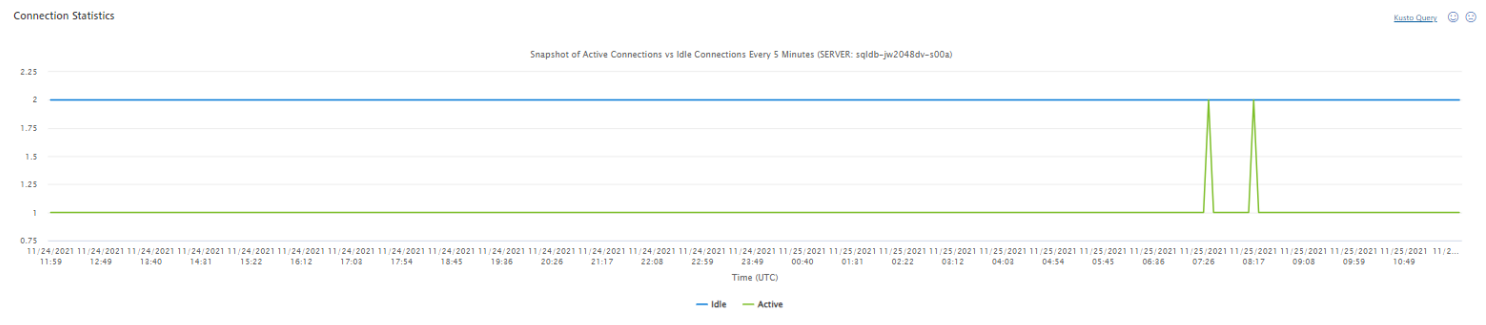


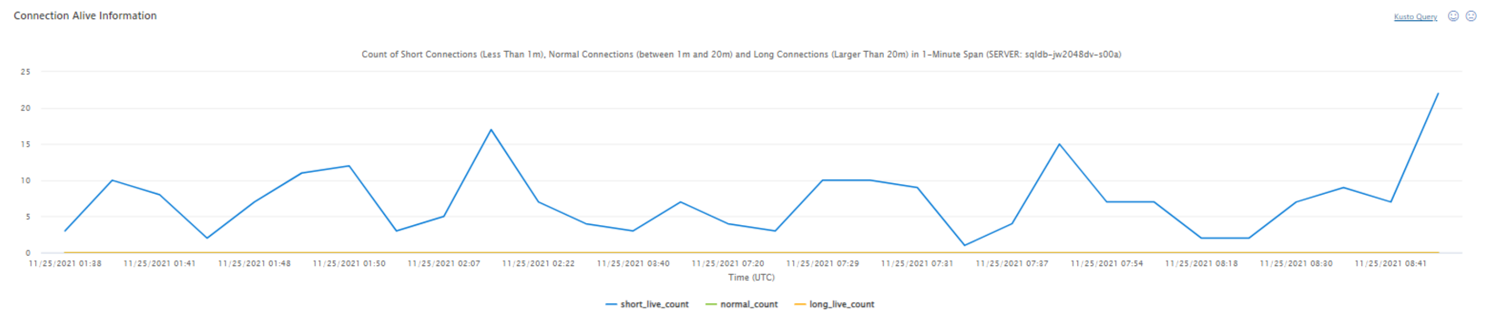
if you did not see any connection latency , the issue is from the client side.

Step 2

check customer connectivity patterns :

you can check the connectivity from our ASC (connectivty tab) :





and make sure that the customer is not creating too many short live connectoions , if that the case , please share with the customer that We highly recommend you use connection pooling, especially if you run into the following scenario:

1- If there are many concurrent, short duration connections. For example, an application with a high level of concurrency creates a connection for each interaction with the database, runs a simple query, and then closes the connection.

2- If there is sudden influx of new connections. For example, when the database server is restarted by users, when the scheduled maintenance is completed, or at the start of peak business hours.

Azure Database for PostgreSQL-Flexible Server offers PgBouncer as a built-in connection pooling solution. See PgBouncer in Azure Database for PostgreSQL - Flexible Server

Step 3

Excessive logging can cause high resource utilization and slow down the system. With certain configurations, your PostgreSQL database server outputs a high amount of log lines. make sure that the below parameters are set based on the below and see if that will help to improve the performance:

log_duration = OFF

log_min_duration_statement = -1

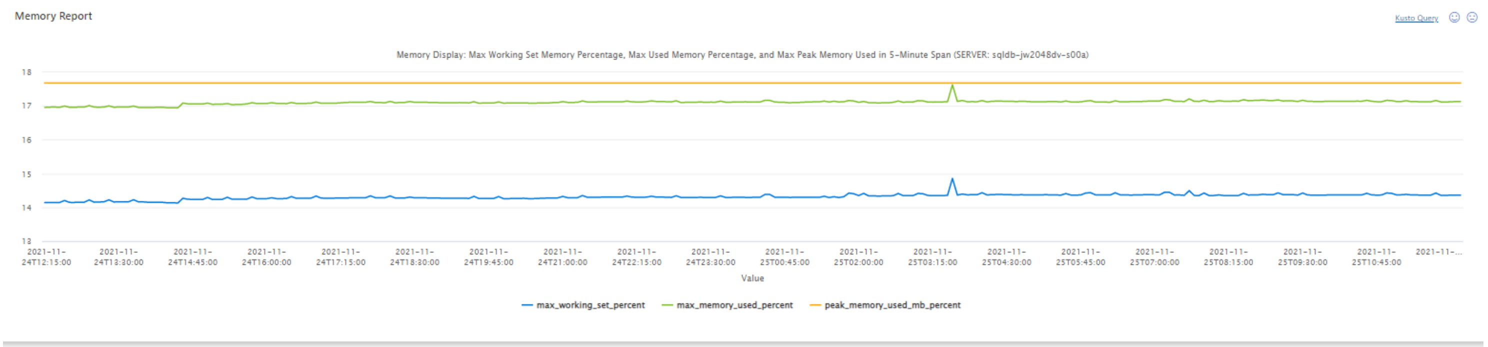
log_statement_stats = OFF

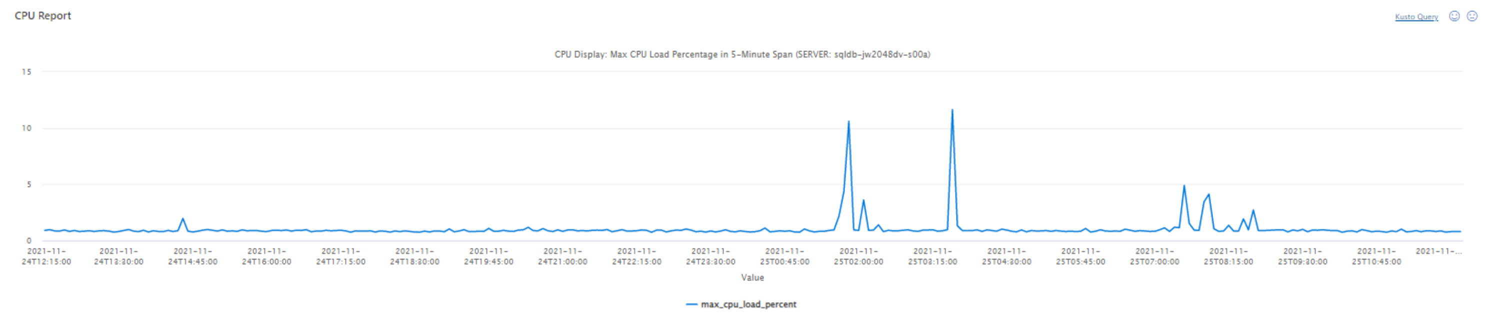
log_statement = NONE

pg_stat_statements.track = NONE

Step 4

check customer resources (CPU/Memory), you can do that from our ASC (Perf tab) to see if there is any resource utilizations :





if there is high utilizations , ask the customer to check his workload or increase the SKU

Step 5

if nothing of the above help, please run the below query if you can identifiy at which stage is the slowness :

```
let startTime = datetime(xxxx-xx-xx xx:xx);
let endTime = datetime(xxxx-xx-xx xx:xx);
let serverName = "pg-server-name";
MonLogin
| where originalEventTimestamp > startTime and originalEventTimestamp < endTime
| where AppTypeName == "Gateway.PG"
| where logical_server_name == serverName
| where event == "process_login_finish"
| where is_success == true
| where connection_id != "00000000-0000-0000-0000-000000000000"
| extend gw_start = datetime_add('millisecond', -total_time_ms, originalEventTimestamp)
| project gw_start, connection_id
| join kind = inner(
MonLogin
| where originalEventTimestamp > startTime and originalEventTimestamp < endTime
| where AppTypeName == "Host.PG"
| where logical_server_name == serverName
| where event == "process_login_finish"
| where is_success == true
| where connection_id != "00000000-0000-0000-0000-000000000000"
| project host_done = originalEventTimestamp, connection_id
) on $left.connection_id == $right.connection_id
| join kind = inner(
MonRdmsPgSqlSandbox
| where originalEventTimestamp > startTime and originalEventTimestamp < endTime
| where LogicalServerName == serverName
| where text contains "azure connection id"
| where text !contains "00000000-0000-0000-0000-000000000000"
| parse text with * "LOG:  Azure Connection ID: \"" ConnectionId "\", VNET \"" VNet "\", SSL " Protocol "\", state " error_state ",
| extend connection_id=toupper(ConnectionId)
| extend backend_done = originalEventTimestamp
) on $left.connection_id == $right.connection_id
| extend gw_to_sockdup = datetime_diff('millisecond', host_done, gw_start)
| extend connectivity_start_to_end = datetime_diff('millisecond', backend_done, gw_start)
//| where connectivity_start_to_end > 500
| project gw_start,
    connectivity_start_to_end,
    gw_to_sockdup,
    guc_comp, //If this time is high, then the issue is likely to be slow storage.
    auth_comp, con_comp, // opening the database files for the connection. If any of these are slow, then it would indicate a storag
    be_start, be_ready, //indicate how long the backend processes are taking to start and become ready
    tls_init_comp, //If this is high, then the issue could be time taken to load certificates.
    ssl_end, //If this is high, there might be high latency in the Azure network in the region, or the gateway might be overloaded.
    load_hba, load_id, //indicate time taken to load firewall files
    auth_start, auth_done, //is the password challenge and response.
    conn_recv, //how long it takes to get to printing the "connection received" log message. If this time is high, it could be becau
    sema // if it is a long time, the server is taking a long time to create new processes to handle connections.
| render timechart
```

Note : for storage issue , make sure to verfiy using this TSG , and if you are noticing high values other than related to storage , you can open an ICM .

Step 6

if nothing of the above help :

1- make sure that the customer client and PostgreSQL server are in the same Azure region.

2- if the customer client is hosted in an Azure VM or VMSS, use accelerated networking for the lowest connection latency.

3- If connecting from an application, check CPU usage on the application. Maxing out CPU can cause slowness and connection drops.

4- Network latency: Check the network latency between the client and the database service instance. You can check the network latency by running simple query as ‘SELECT 1’ , and you might need to engage the netowrk team to capture traces for deeper details.

How good have you found this content?