PostgreSQL Connectivity Latency Troubleshooting

Last updated by | Karthickselvam M | Oct 10, 2022 at 11:44 AM PDT

When a customer case comes in with the customer complaining about high connection latency, one of the first things to understand is whether the latency is occurring in the Azure connectivity layer (Gateway, Host, and socket duplicator), or if it is occurring in the PostgreSQL backend. It also helps to understand if latency is consistently bad or if it is bad only at certain times.

The following Kusto query gives min, max, and average connection latency both in the connectivity layer and in total:

```
let startTime = ago(1d);
let endTime = now();
let serverName = "serverNameHere";
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, total_time_ms, gw_node=NodeName
| 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
| extend connectivity_start_to_end = datetime_diff('millisecond', host_done, gw_start)
| where connectivity_start_to_end > 0
| summarize min(connectivity_start_to_end), max(connectivity_start_to_end), avg(connectivity_start_to_end),
min(total_time_ms), max(total_time_ms), avg(total_time_ms) by bin(gw_start, 1m)
| render timechart
```

It is usually most helpful to look at the maximum times because those are the ones the customer complains about. However, it is sometimes also useful to look at average times.

In almost all cases, the connectivity start-to-end time, which spans between the time when the request first hits the gateway until it is handed off to the socket duplicator, is low. If so, then the latency is mostly occurring in the PostgreSQL server.

In the rare cases where latency is high in the connectivity layer, it can help to understand if the control ring (where the gateway is) and the tenant ring (where the host is) are in separate availability zones. To be honest, I don't know how to find this out, and would ask someone on the Fabric team for help with it. In practice, we really only see this when there is a networking problem in the region, or sometimes in West Europe where one of the AZs is located far from the other two. In the West Europe case, latency would get worse between AZs as the amount of total traffic increases.

If connectivity layer latency is high, then it can also help to check TCP retransmissions for control ring nodes:

```
MonCounterFiveMinute
| where TIMESTAMP > ago(1d)
| where ClusterName startswith "cr1" // <-- The correct control ring prefix here
| where CounterName contains "TCPv4" and CounterName contains "Retransmitted"
| project TIMESTAMP, NodeName, CounterValue, CounterName
| render timechart
```

If there are any nodes where the number of retransmissions is consistently more than 400, then the gateways on those nodes are likely overloaded. In previous incidents, we have had control rings with 3 OrcasSQL gateways out of 18 total gateway nodes. While looking at the above chart, it was clear that retransmissions were far higher on the three nodes running OrcaSQL gateways than on the other nodes. In this case, the solution was to increase the Gateway.PG node count to better distribute the load across nodes.

In most cases, however, latency will be mostly in the PostgreSQL server backend. In those cases, the next step would be to break down where the time is going.

The first step to check relating to the specific server would be the socket duplicator queue depth:

```
MonRdmsPgSqlSandbox
| where originalEventTimestamp > ago(1d)
| where LogicalServerName == "serverNameHere"
| where text contains "queuedepth"
| extend depth = toint(replace("[^0-9]*$", "", replace("^[^0-9]*", "", text)))
| summarize max(depth) by bin(originalEventTimestamp, 10m)
| render timechart
```

If the queue depth gets much above 5, then the latency of the server processing the connections is going to get noticeable. If the queue depth is very high, then customer connections will timeout before they complete.

High queue depth can be due to a number of things.

First would be server failure. If the queue depth suddenly increases until it is up to 400, then likely the server is hung. We have seen this happen due to crashes or failover.

Second would be a connection rush. If the customer is flooding the server with connections, then the queue depth is going to increase just because the number of inbound connections is going to exceed the capacity of the server to process them. The following query will show if the connection rate is high:

```
MonLogin
| where originalEventTimestamp > ago(1d)
| where logical_server_name == "serverNameHere"
| where AppTypeName == "Gateway.PG"
| where event == "process_login_finish"
| where is_success == true
| summarize count() by bin(originalEventTimestamp, 10m)
| render timechart
```

The last reason is that if the server is under heavy load, that can slow down the rate at which it processes new connections. A query of resource metrics can indicate if the backlog is due to server load:

```
MonResourceMetricsProvider
| where originalEventTimestamp >= ago(1d)
| where LogicalServerName == "serverNameHere"
| where isnotnull(cpu_load)
| extend total_iops_scaled = total_iops / 100
| summarize max_cpu = min_of(max(cpu_load_percent), 100), max(working_set_percent), max(total_iops_scaled) by
bin(originalEventTimestamp, 10m)
| render timechart
```

The following query would indicate if total connection latency, in general, is related to server load:

```
let serverName = "serverNameHere";
let startTime = ago(1d);
let endTime = now();
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, total_time_ms, gw_node=NodeName
| 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
| extend connectivity_start_to_end = datetime_diff('millisecond', host_done, gw_start)
| where connectivity_start_to_end > 0
| summarize max(connectivity_start_to_end), avg(connectivity_start_to_end), max(total_time_ms),
avg(total_time_ms) by bin(gw_start, 1h)
| extend capped_connectivity = min_of(max_connectivity_start_to_end, 10000) / 100
| extend capped_total = min_of(max_total_time_ms, 10000) / 100
| extend avg_connectivity = min_of(avg_connectivity_start_to_end, 10000) / 100
| extend avg_total = min_of(avg_total_time_ms, 10000) / 100
| join kind = inner (
MonResourceMetricsProvider
| where LogicalServerName == serverName
| where isnotnull(cpu_load)
| extend total_iops_scaled = total_iops / 100
| project originalEventTimestamp, cpu_load_percent, working_set_percent, total_iops_scaled
| summarize max(cpu_load_percent), max(working_set_percent), max(total_iops_scaled) by
bin(originalEventTimestamp, 1h)
) on $left.gw_start == $right.originalEventTimestamp
| project originalEventTimestamp, max_cpu_load_percent, capped_connectivity, capped_total, avg_connectivity,
avg_total, max_total_iops_scaled, max_working_set_percent
| render timechart
```

If the connectivity layer and queue depth are both fine, then the next thing to check is where in the backend server the latency is occurring, using a query such as the following:

```
let startTime = ago(1d);
let endTime = now();
let serverName = "serverNameHere";
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 * "Azure Connection ID: \"" ConnectionId "\", VNET \"" VNet "\", SSL " Protocol "\", 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 w
    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
    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 h
    sema // if it is a long time, the server is taking a long time to create new processes to handle connectio
| render timechart
```

If most connections are good, and there is too much information on the chart, it can be limited by uncommenting the where clause for "connectivity_start_to_end", and limiting the chart to poorly performing connections.

Also, the timers in the "project" clause should be substituted as needed to narrow down where the issue is.

Each of the times in the parsed sandbox times indicates a different phase on backend connection processing.

- The "sema" phase indicates how long the server takes to get a backend-available semaphore, and if it is a long time, the server is taking a long time to create new processes to handle connections.
- "be_start" and "be_ready" indicate how long the backend processes are taking to start and become ready.

- "guc_comp" indicates how long the server took to read the server configuration for the new connection. This is an interesting one because it is the first time that the server accesses storage. If this time is high, then the issue is likely to be slow storage. We often see this taking time on Basic edition servers, but if it is taking a long time on General Purpose or Memory Optimized editions, then that is a sign of a storage issue, either with Azure BLOB or SBS.

- "conn_recv" is how long it takes to get to printing the "connection received" log message. If this time is high, it could be because sandbox logging is slow. In that case, check to see if the rate of logging to the sandbox is high, or if there is a problem with the MDS agent on the node.

- "tls_init_comp" is the time taken to initialize the SSL library. If this is high, then the issue could be time taken to load certificates. This used to be a huge issue because these files are stored on Azure standard file share. However, since introducing the certificate cache, this phase should take almost no time. If it is taking time, then that indicates a problem.

- "ssl_end" is the time to negotiate SSL with the gateway. If this is high, there might be high latency in the Azure network in the region, or the gateway might be overloaded.

- "load_hba" and "load_ident" indicate time taken to load firewall files. This used to also be a huge issue, again due to Azure standard file share. However, since introducing the firewall file cache, this phase should take almost no time. If it is taking time, one thing to check would be the size of the pg_hba.conf file. If it is too large to fit into the cache, then it could cause problems. In that case, there would also be sandbox log messages indicating that the file cannot be cached. If this happens a lot, we need to increase the size of the firewall file cache, which is a code change.

- From "auth_start" to "auth_done" is the password challenge and response. This phase goes end-to-end between the client and the server. If this is taking a long time, then it could be that there is a slow connection between the customer network and Azure, or that the customer's client is not responding quickly to the challenge.

- Other phases between "auth_done" and "con_comp" have to do with opening the database files for the connection. If any of these are slow, then it would indicate a storage issue. Again, if this is Basic edition, then there isn't much we can do (unless it is ridiculously slow, in which case, contact the Xstore team), but for other editions, slowness here is a trouble sign, and it would indicate problems with Azure BLOB storage or with SBS.

- The remaining fields are breakdowns of the SSL negotiation, which indicate amount of time waiting to read handshake message, waiting to write handshake messages, and amount of time spent in the TLS library. Generally, the majority of the time for slow TLS negotiation should be spent in handshake_read, which again indicates slow gateway response. OpenSSL takes very little time in the library, and even Schannel is relatively quick in the library since we optimized trusted root certificates. If time in the library for Schannel servers is high, then that is a trouble sign, and it needs further investigation.

**We recently have a case which customer enabled log_duration server parameter. After a while, they started to notice the connection latency. Thus suggest to check those server parameters.