

Troubleshoot Replication Lag

Last updated by | Shawn Xiao | Sep 13, 2022 at 10:53 PM PDT

Before checking lag itself, make sure the primary and replica are in ready state and accepting connections and fix any issues related to that. Use this TSG to understand replica lag only.

Use this TSG to check workload increases on the primary server, that might explain the replication lag. This is more comprehensive than just point 3 below to check the primary's workload.

[https://supportability.visualstudio.com/AzureDBPostgreSQL/_wiki/wikis/AzureDBPostgreSQL/453471/Detect-Customer-workload-\(Statistics-on-INSERT-DELETE-UPDATE-FETCHED-RETURNED-TUPLE\)](https://supportability.visualstudio.com/AzureDBPostgreSQL/_wiki/wikis/AzureDBPostgreSQL/453471/Detect-Customer-workload-(Statistics-on-INSERT-DELETE-UPDATE-FETCHED-RETURNED-TUPLE))

1. Check current lag of replica from primary's perspective.

```
MonDmPgsQLReplicationStatsPrimary
| where LogicalServerName == "{servername}"
| project LogicalServerName, slot_name, active, slot_type, wal_sender_state, PreciseTimeStamp, application_na
| order by PreciseTimeStamp desc
```






total_lag_in_bytes will only show is replica is current in streaming replication. i.e when 'active' column is true or when replication slot is active. If not scroll down and see the last lag you see.

While streaming is active, pay attention to the *_lsn* and the *_lag_in_byte* columns:

<https://www.postgresql.org/docs/11/view-pg-replication-slots.html> 

Please refer to

[https://supportability.visualstudio.com/AzureDBPostgreSQL/_wiki/wikis/AzureDBPostgreSQL/453424/LSN-DISTANCE-MOVED-EVERY-HOUR-\(Measure-of-transaction-logs-size-produced\)](https://supportability.visualstudio.com/AzureDBPostgreSQL/_wiki/wikis/AzureDBPostgreSQL/453424/LSN-DISTANCE-MOVED-EVERY-HOUR-(Measure-of-transaction-logs-size-produced)) to understand the LSN DISTANCE.

Column	Explanation
current_wal_lsn	The pg_current_xlog_location() or pg_current_wal_lsn() of the primary server
restart_lsn	The address (LSN) of oldest WAL which still might be required by the consumer of this slot and thus won't be automatically removed during checkpoints. NULL if the LSN of this slot has never been reserved. From https://www.postgresql.org/docs/11/view-pg-replication-slots.html 
sent_lsn	Last transaction log position sent on this connection. From https://www.postgresql.org/docs/9.2/monitoring-stats.html#PG-STAT-REPLICATION-VIEW 
write_lsn	Last transaction log position written to disk by this standby server. From https://www.postgresql.org/docs/9.2/monitoring-stats.html#PG-STAT-REPLICATION-VIEW 
flush_lsn	Last transaction log position flushed to disk by this standby server. From https://www.postgresql.org/docs/9.2/monitoring-stats.html#PG-STAT-REPLICATION-VIEW 
replay_lsn	Last transaction log position replayed into the database on this standby server. From https://www.postgresql.org/docs/9.2/monitoring-stats.html#PG-STAT-REPLICATION-VIEW 
total_lag_in_bytes	current_wal_lsn - replay_lsn
sending_lag_in_bytes	current_wal_lsn - sent_lsn
receive_lag_in_bytes	sent_lsn - flush_lsn
replay_lag_in_bytes	flush_lsn - replay_lsn

The above LSN and lag breakdowns give a very good idea of why the replica is lagging:





- If total_lag_in_bytes is high, then check if the main lag is in *sending_lag_in_bytes* or in *receive_lag_in_bytes* or on *replay_lag_in_bytes*. In most cases the lag is in *replay_lag_in_bytes*. This means that the replica received all the data from the primary and has written to its wal files, but it still has not replayed it (i.e applied it)
- If *sending_lag_in_bytes* is higher it means the primary for some reason hasn't sent the data to the replica yet.
- If *receive_lag_in_bytes* is high, the replica for some reason hasn't written the received data to its WAL files yet.

Again in most cases "replay_lag_in_bytes" would be the higher lag. This is because replica has to apply all the changes from the primary server in a single threaded *wal_receiver*. While the replication is in streaming state,

these columns give a very good indicate of the lag.

- If replica is not in steaming state, then check step 2. Otherwise proceed to Step 3.

ADDITIONAL INFORMATION: LOOK AT THESE TWO SOURCE FILES FOR THE source of this telemetry.

- <https://www.postgresql.org/docs/9.2/monitoring-stats.html#PG-STAT-REPLICATION-VIEW>  (viewcollectionPrePg10.xml)
- <https://msdata.visualstudio.com/Database Systems/ search? action=contents&text=MonDmPgsqLReplicationStatsPrimary&type=code&lp=code-Project&filters=ProjectFilters{Database Systems}RepositoryFilters{orcasql-shared}&pageSize=25&result=DefaultCollection%2FDatabase Systems%2Forcasql-shared%2FGBmaster%2F%2Fsrc%2FApp%2FWorker.PG%2FXdbLaunchPostgresInstanceAgent%2FActors%2FPostgresCollectorActorBase.cs>  (viewcollectionSincePg10.xml)
- <https://www.postgresql.org/docs/11/view-pg-replication-slots.html>  (pg_replication_slots view)
- <https://www.postgresql.org/docs/9.2/monitoring-stats.html#PG-STAT-REPLICATION-VIEW>  (pg_stat_replication view)

2. If you see streaming is not happening currently, see if replica is doing file-shipping replication by restoring from log files from primary's archive location.
This query will show if replica started doing fileshipping when streaming replication stopped and if it is still doing it now.

```
MonRdmsPgSqlSandbox
| where LogicalServerName == "{servername}"
| where text contains "restored log file"
| project originalEventTimestamp, LogicalServerName, text
```

This query will show the speed at which replica is applying wal files from primary per hour.

```
MonRdmsPgSqlSandbox
| where LogicalServerName == "{servername}"
| where text contains "restored log file"
| project originalEventTimestamp, LogicalServerName, text
| summarize count() by LogicalServerName, bin(originalEventTimestamp, 1h)
| render timechart
```

3. Check if there is an increase in the activity of the primary server.

- If we see lag building up after a high workload activity on the primary, that mostly explains it.
- If the workload comes down, eventually replica should catchup, but if does not the lag can continue to increase.

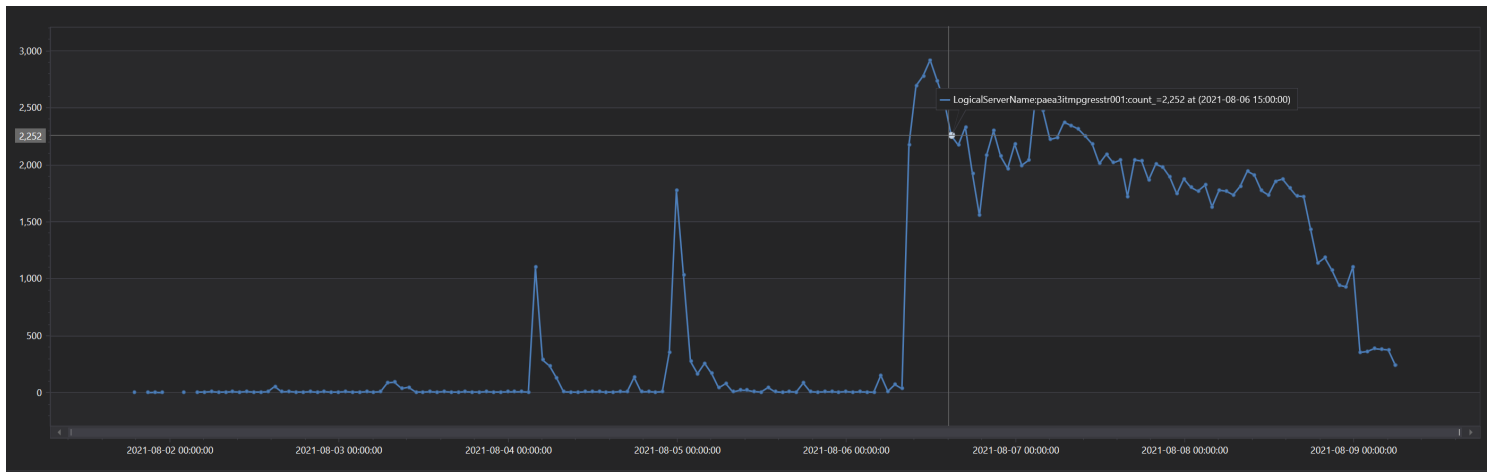
Also the below query will give the amount of wal files the primary is archiving per hour. (So will give a decent indication if primary is producing more wal, i.e more workload) If there is a significant different between the amount of wal primary is producing compared to what replica can apply in step 2, that explains the increasing lag.

```

MonRdmsPgSqlSandbox
| where LogicalServerName == "{servername}"
| where text contains "xlogcopy.archive: archived"
| project originalEventTimestamp, LogicalServerName, text
| summarize count() by LogicalServerName, bin(originalEventTimestamp, 1h)
| render timechart

```

Something like below clearly shows that the primary stated doing too much write/update activity. The amount of wal archived gives a very good idea of amount of wal produced.



4. Check the lag from the replica perspective (lag is reported in seconds)

```

MonDmPgSqlReplicationStatsReplica
| where LogicalServerName == "{servername}"
| project LogicalServerName, PreciseTimeStamp, todouble(log_delay_seconds), last_wal_receive_lsn, last_wal_rep
| project PreciseTimeStamp, todouble(log_delay_seconds)
| render timechart

```

In picture below, you can see that the *last_wal_replay_lsn* is lagging the *last_wal_receive_lsn* by a lot.

- *replaying_lag_in_bytes* gives an measure of the amount of wal to be still applied on the replica.
- *last_wal_receive_lsn* is only available when replica is in streaming state and so *replaying_lag_in_bytes* is also only available at those times.
- *log_delay_seconds* is available in both replication modes. Streaming or file-shipping.

```

5
6 MonDmPgSqlReplicationStatsReplica
7 | where LogicalServerName == "eun-prod-coupondbserver-green"
8 | project LogicalServerName, PreciseTimeStamp, todouble(log_delay_seconds), last_wal_receive_lsn, last_wal_replay_lsn
9 | order by PreciseTimeStamp desc
10

```

LogicalServerName	PreciseTimeStamp	log_delay_seconds	last_wal_receive_lsn	last_wal_replay_lsn
eun-prod-coupondbserver-green	2021-02-03 20:12:22.1564686	145674.930725	E9D/8A9D0000	DFF/DFC2A770
eun-prod-coupondbserver-green	2021-02-03 20:07:17.4971066	145376.708893	E9D/5E890000	DFF/CA3FEE18
eun-prod-coupondbserver-green	2021-02-03 20:02:12.7922499	145081.926069	E9D/32A80000	DFF/B435DE08
eun-prod-coupondbserver-green	2021-02-03 19:57:08.1343966	144782.971438	E9D/6370000	DFF/9E56D238

If you have any doubts when following this page, please reach out to *xixia* for clarification and wiki/TSG improvement.