GeoDR - Replication lag check

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:27 AM PST

Contents

GeoDR - Geo-Replication lag Troubleshooting

This is a TSG for troubleshooting the replication lag between geo-replication primary and secondary databases.

Issue

Lag in geo-replication may impact the performance on the primary database. In the worst case, it can cause database availability issues if the issue persists and is not mitigated quickly enough.

Geo-replication is not immediate because the synchronization occurs asynchronously. A lag of a few seconds is normal, especially if the workload is high or if primary and secondary databases are in different regions. If the lag becomes too large, the primary database will be throttled to allow the secondary to catch up with the backlog. Throttling will start if the estimated delay of `redo_queue` / `redo_rate` gets higher than 1 minute. The primary remains throttled until `redo_queue` / `redo_rate` falls below 45 seconds.

The customer usually detects this issue through either of the following symptoms:

- They monitor the geo-replication lag through the sys.dm_geo_replication_link_status DMV and notice an unusual, increasing lag. See below at Option 3: DMVs run by the customer for an example of how to do this.
- They notice that the readable geo-replication secondary has significantly older data than the primary, e.g. when comparing Identity columns on tables or checking a "LastUpdated" datetime column.
- They notice performance issues on their geo-replication primary. When checking the Wait types of their slow-running queries, they see one of the `HADR_THROTTLE_LOG_RATE` wait types:

Wait type	Description
HADR_THROTTLE_LOG_RATE_GOVERNOR	Occurs when the maximum throughput rate is reached (workload too high).
HADR_THROTTLE_LOG_RATE_MISMATCHED_SLO	Occurs when a geo-replication secondary is configured with lower compute size (lower SLO) than the primary. A primary database is throttled due to delayed log consumption by the secondary. This is caused by the secondary database having insufficient compute capacity to keep up with the primary database's rate of change.
HADR_THROTTLE_LOG_RATE_LOG_SIZE	Occurs when the transaction log size is exceeding a threshold, to avoid an out of log space condition
HADR_THROTTLE_LOG_RATE_SEEDING	The secondary is still seeding.
HADR_THROTTLE_LOG_RATE_SEND_RECV_QUEUE_SIZE	Occurs when the send and/or receive replication queues are exceeding a threshold.

Note that geo-replication lag is different from the replication lag that you might see on Premium or Business Critical SLOs. The secondaries associated with Premium and BC are *synchronous* nodes, whereas nodes associated with geo-replication are updated *asynchronously*. For lag in synchronous replication, you'd rather see wait types like `HADR_SYNC_COMMIT` or `HADR_GROUP_COMMIT`.

Investigation / Analysis

Possible causes include:

- Mismatched SLO between primary and secondary, indicated by the wait type `HADR_THROTTLE_LOG_RATE_MISMATCHED_SLO`. The primary has a higher SLO than the secondary and thus can process higher workloads than the secondary will ever be able to catch up with. See the separate article at Mismatched SLO.
- Workload is too high for the chosen service tier. Scale up the database to a higher SLO (scale secondary first before scaling the primary to avoid the "mismatched SLO" scenario).
- The secondary database is readable and its local workload impacts the geo-replication progress. See separate article Blocking on Read Replica for more information.
- The geo-replication process itself is damaged or stuck and cannot proceed.

- Managed Instance: General purpose uses remote storage, with the transaction log file size determining the transaction log I/O performance. Throughput is very limited for smaller file performance levels like e.g. P10.

- Managed Instance: Network configuration issue (mainly for MIs with custom routing through NVAs and complex NSGs/RTs). An overly complex routing scheme or slow processing in custom NVAs can throttle the general network speed between primary and secondary MI.

Option 1: ASC

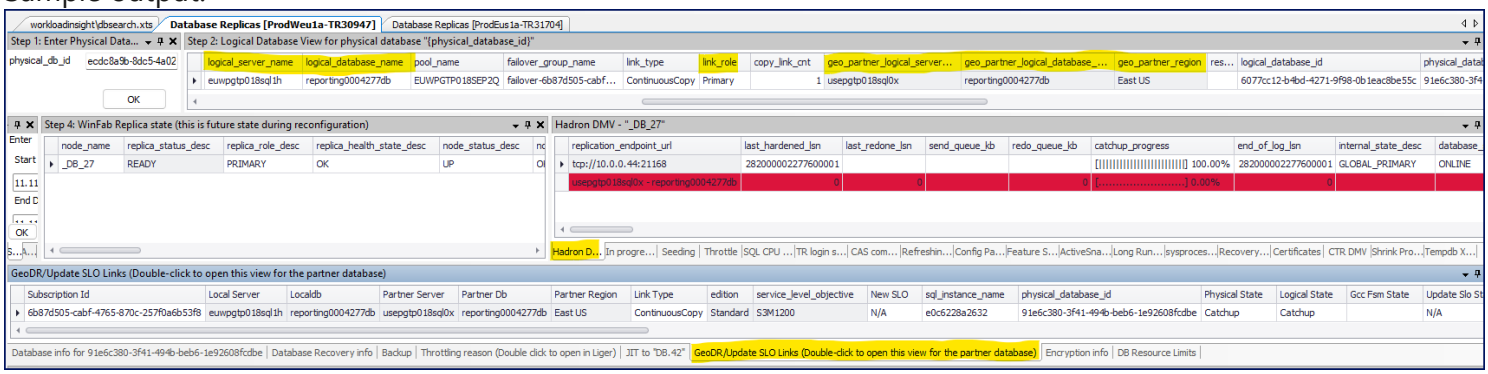
Create ASC troubleshooter reports for both primary and secondary databases and check for any bottlenecks. ASC also has a separate page for geo-replication that might already provide you with sufficient information.

Option 2: Telemetry

XTS

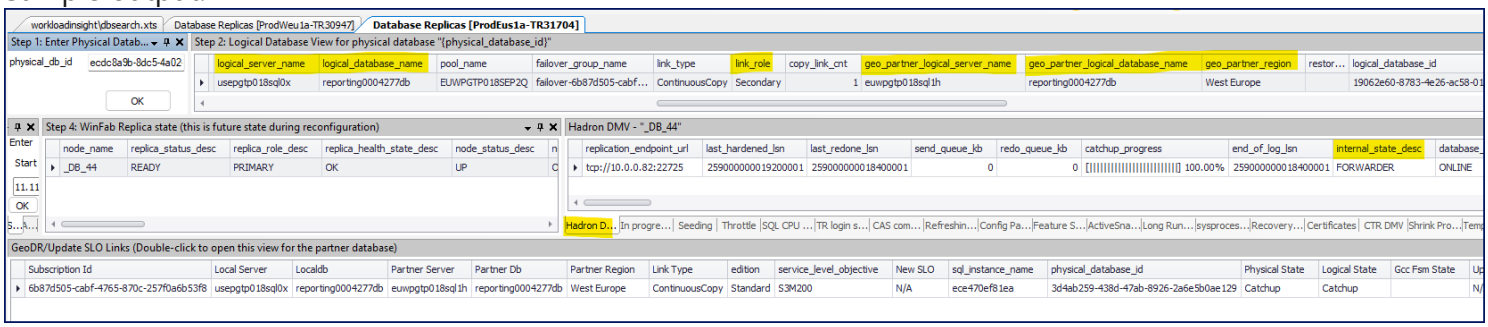
The relevant view is "Database Replicas.xts" with its "Hadron DMV" subquery. You can easily open it through "DBSearch.xts" after selecting the database there. With the help of "Database Replicas.xts", you can figure out what the primary and secondary databases are and what the replication queue status is. You will e.g. see if the secondary database is unhealthy, has availability issues, or if something is wrong with the synchronization itself.

Sample output:



Note the hint regarding double-clicking the GeoDR row for opening the view for the partner database. This will give you the situation as seen by the replication partner.

Sample output:



In this specific case, the geo-replication has been established, is in catch-up mode, but got stuck at "0%" in the catch-up progress. The replication is unhealthy. A clue towards the cause is the different values in `service_level_objective` in the "GeoDR/Update SLO" section: the primary is running with "S3M1200", the secondary with "S3M200". This is pointing to a "Mismatched SLO" scenario. In addition, there appears to be contradicting information in the "internal state desc" column: it is empty on the primary but says "FORWARDER"

on the secondary, which looks wrong. There might have been a previous role switch between primary and secondary and it appears that the internal role status has not been updated properly. The serious issue however is the mismatched SLO, which the customer should correct in a first step; though it is unclear if the role mismatch would have any negative impact on it. If the status is unclear like this, or if you see anything that is throwing doubts on the overall health, you should open an IcM to confirm the next steps.

Kusto

Step 1: Identify the status and SLO for primary and secondary

Run this for both the primary and secondary database to see availability status and SLO:

```
let srv = 'servername';
let db = "databasename";
let startTime = datetime(2022-11-06 14:00:00Z);
let endTime = datetime(2022-11-06 19:00:00Z);
let timeRange = ago(7d);
MonAnalyticsDBSnapshot
 | where  TIMESTAMP >= startTime
 | where  TIMESTAMP <= endTime
 | where logical_server_name =~ srv
 | where logical_database_name =~ db
 | extend AppName = sql_instance_name
 //| distinct AppName
| project TIMESTAMP, AppName, region_name, logical_server_name, logical_database_name, physical_database_id, d

// Primary:
TIMESTAMP                    AppName       region_name  logical_server_name  logical_database_name  physical_d
2022-11-06 14:12:27.6392845  e0c6228a2632  West Europe  servername           databasename           91E6C380-3
2022-11-06 14:43:27.2641612  e0c6228a2632  West Europe  servername           databasename           91E6C380-3
2022-11-06 15:14:05.8729466  e0c6228a2632  West Europe  servername           databasename           91E6C380-3

// Secondary:
2022-11-06 14:18:08.1226961  ece470ef81ea  East US      servername           databasename           3D4AB259-4
2022-11-06 14:49:25.1060147  ece470ef81ea  East US      servername           databasename           3D4AB259-4
2022-11-06 15:18:26.1025742  ece470ef81ea  East US      servername           databasename           3D4AB259-4
```

Note the different values in `service_level_objective` : the primary running with "S3M1200", the secondary with "S3M200". This is pointing to a "Mismatched SLO" scenario.

Step 2: Check the status and health of the replication

```
let srv = 'servername';
let db = "databasename";
let startTime = datetime(2022-11-06 14:00:00Z);
let endTime = datetime(2022-11-06 19:00:00Z);
let timeRange = ago(7d);
MonDmDbHadrReplicaStates
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
//| where  TIMESTAMP >= timeRange
| where LogicalServerName =~ srv
| where logical_database_name  =~ db
| where is_forwarder == 1  // on prmary
//| where is_forwarder == 0  // on secondary
//| distinct AppName, NodeName
| project TIMESTAMP, NodeName, AppName, LogicalServerName, logical_database_name, group_id,
is_forwarder, internal_state_desc, is_local, is_primary_replica, is_commit_participant, is_seeding_in_progress
database_state_desc, synchronization_state_desc, synchronization_health_desc, recovery_lsn, truncation_lsn, la
last_sent_time, last_received_lsn, last_received_time, last_hardened_lsn, last_hardened_time, last_redone_lsn,

// Primary:
TIMESTAMP                    NodeName  AppName       LogicalServerName  logical_database_name group_id
2022-11-06 14:00:07.9624721  _DB_8     e0c6228a2632  servername         databasename          91E6C380-3F41-49
2022-11-06 14:01:07.9782414  _DB_8     e0c6228a2632  servername         databasename          91E6C380-3F41-49
2022-11-06 14:02:08.0096422  _DB_8     e0c6228a2632  servername         databasename          91E6C380-3F41-49

// Secondary:
2022-11-06 14:00:56.9838018  _DB_60    ece470ef81ea  servername         databasename          3D4AB259-438D-47
2022-11-06 14:01:56.9841691  _DB_60    ece470ef81ea  servername         databasename          3D4AB259-438D-47
2022-11-06 14:02:57.0158142  _DB_60    ece470ef81ea  servername         databasename          3D4AB259-438D-47
```

Note that geo-replication primaries are identified through `is_forwarder=1` . There appears to be the same mismatch between "is_forwarder", "internal_state_desc", "is_primary" that already appeared on the XTS view. The remaining columns that are not on this sample resultset correspond to what is seen on the XTS results, including the recovery and truncation LSNs. As explained above in the XTS section, rather go with an IcM if you have any doubts regarding the overall health status of the databases.

Step 3: Check the replication queue status

This Kusto query returns data on the geo-replication primary, but should come back empty on the secondary. Note the two "| project" lines - remove the second one if you want to see a lot more relevant details.

```
let srv = 'servername';
let db = "databasename";
let startTime = datetime(2022-11-06 14:00:00Z);
let endTime = datetime(2022-11-06 19:00:00Z);
let timeRange = ago(7d);
MonAnalyticsDBSnapshot
 | where  TIMESTAMP >= startTime
 | where  TIMESTAMP <= endTime
 | where logical_server_name =~ srv
 | where logical_database_name =~ db
| distinct physical_database_id
| join kind=inner
(
    MonFabricThrottle
    | where  TIMESTAMP >= startTime
    | where  TIMESTAMP <= endTime
    // | where  TIMESTAMP >= timeRange
    | where LogicalServerName =~ srv
    //| where event == "hadr_db_log_throttle"
    | where throttling_reason in~ ("QueueSize", "LogSize", "MismatchedSlo")
) on ($left.physical_database_id) == ($right.database_replica_id)
| limit 10000
| project TIMESTAMP, AppName, NodeName, LogicalServerName, database_name, database_replica_id, event, throttle
| project TIMESTAMP, AppName, NodeName, LogicalServerName, database_name, database_replica_id, event, throttli


TIMESTAMP                   AppName       NodeName  LogicalServerName  database_name
2022-11-06 14:00:40.3994196  e0c6228a2632  _DB_8     servername         91e6c380-3f41-494b-beb6-1e92608fcdbe
2022-11-06 14:00:40.3994196  e0c6228a2632  _DB_8     servername         91e6c380-3f41-494b-beb6-1e92608fcdbe
2022-11-06 14:00:45.3996057  e0c6228a2632  _DB_8     servername         91e6c380-3f41-494b-beb6-1e92608fcdbe
2022-11-06 14:00:45.3996057  e0c6228a2632  _DB_8     servername         91e6c380-3f41-494b-beb6-1e92608fcdbe
```

Possible return values for the `throttling_reason` column are:

> NoThrottling
> QueueSize
> MismatchedSlo
> LogStorageCapabilities
> LogSize
> SloDowngrade

Step 4: Check MonSQLSystemHealth for the SQL error log

Run the following Kusto query to see if there are any messages or errors out of the ordinary. Run it for both the primary and the secondary.

```
let srv = 'servername';
let db = "databasename";
let startTime = datetime(2022-11-06 14:00:00Z);
let endTime = datetime(2022-11-06 19:00:00Z);
let timeRange = ago(7d);
let AppNames = MonAnalyticsDBSnapshot
  | where  TIMESTAMP >= startTime
  | where  TIMESTAMP <= endTime
  //| where  TIMESTAMP >= timeRange
  | where logical_server_name =~ srv
  | where logical_database_name =~ db
  | extend AppName = sql_instance_name
  | distinct AppName;
MonSQLSystemHealth
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
//| where TIMESTAMP > timeRange
| where AppName in~ ( AppNames )
//| where error_id > 0
| summarize count(), take_any(message) by error_id
| order by error_id asc
```

Step 5 - Check for I/O stalls on remote storage

This Kusto query applies to Azure SQL Database and Managed Instance. It shows if there were any I/O stalls on remote storage at database level. Filter for LDF files to isolate transaction log issues.

```
let srv = 'servername';
let startTime = datetime(2022-11-06 14:00:00Z);
let endTime = datetime(2022-11-06 19:00:00Z);
let timeRange = ago(7d);
MonSQLXStore
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
//| where TIMESTAMP > timeRange
//| where LogicalServerName =~ srv
| where  http_errorcode == 503
| where SourceNamespace == "AzDbProdSql" //or SourceNamespace == "AzMiProdSql" // SQL DB or MI
| where file_path endswith ".mdf" or file_path endswith ".ndf" or file_path endswith '.ldf'
//| where file_path endswith ".ldf"
//| where file_path !contains 'replicatedmaster' and file_path !contains 'msdb' and file_path !contains 'manag
| limit 5000
| project TIMESTAMP, NodeName, AppName, LogicalServerName, event, file_path, page_blob_tier, mapped_errorcode,
```

Step 6 (only Managed Instance - General Purpose): Check XIO remote storage details

This Kusto query will give you an idea about the peformance between the MI compute node and the remote storage that is used for General Purpose instances. Note what the `page_blob_tier` column is showing you - if it is a "P10" as in the example below, than this database is a good candidate for replication lag. You rather want to see a "P30" there, unless the database is mostly idle.

```
// for MI and Synapse only - does not work for SQL DB
let srv = 'servername';
let startTime = datetime(2022-11-06 14:00:00Z);
let endTime = datetime(2022-11-06 19:00:00Z);
let timeRange = ago(7d);
MonSQLXStoreIOStats
| where  TIMESTAMP >= startTime
| where  TIMESTAMP <= endTime
//| where TIMESTAMP > timeRange
| where LogicalServerName =~ srv
| where file_path endswith ".ldf" and  isnotempty(page_blob_tier)
| where file_path !contains 'replicatedmaster' and file_path !contains 'msdb' and file_path !contains 'managed
//|limit 50
| project TIMESTAMP, NodeName, AppName, LogicalServerName, event, file_path, total_requests, total_xio_request
// there are a lot more columns with detailed performance information - remove the "| project" lines to see al
//| summarize min(TIMESTAMP), max(TIMESTAMP) by file_path, page_blob_tier, AppName

TIMESTAMP                    NodeName  AppName       LogicalServerName  event           file_path
2022-11-06 15:36:51.9155947  _DB_30    c548ca7c1063  servername         xstore_io_stats https://wasdpxxxx.blob
2022-11-06 15:41:51.9460156  _DB_30    c548ca7c1063  servername         xstore_io_stats https://wasdpxxxx.blob
2022-11-06 15:46:51.9763724  _DB_30    c548ca7c1063  servername         xstore_io_stats https://wasdpxxxx.blob
2022-11-06 15:51:51.9911071  _DB_30    c548ca7c1063  servername         xstore_io_stats https://wasdpxxxx.blob
```

Option 3: DMVs run by the customer

The DMV sys.dm_geo_replication_link_status exposes geo-replication performance details to the customer. On the primary database, it shows the replication lag in seconds between the transactions being committed on the primary and being hardened to the transaction log on the secondary. For example, if the lag is 30 seconds, it means that if the primary is impacted by an outage at this moment and a geo-failover is initiated, transactions committed in the last 30 seconds will be lost. To measure lag with respect to changes on the primary database that have been hardened on the geo-secondary, compare `last_commit` time on the geo-secondary with the same value on the primary.

```
SELECT
    @@servername as local_server,
    db_name(DB_ID()) as local_database,
    role_desc,
    partner_server,
    partner_database,
    last_replication,
    last_commit,
    replication_lag_sec,
    replication_state_desc,
    secondary_allow_connections_desc
FROM sys.dm_geo_replication_link_status;

-- on primary:
local_server  local_database  role_desc  partner_server  partner_database  last_replication
------------- --------------- ---------- --------------- ----------------- ----------------------------------
primarysrv    AdventureWorks  PRIMARY    secondarysrv    AdventureWorks    2022-11-10 08:43:57.6866667 +00:00
-- on secondary:
secondarysrv  AdventureWorks  SECONDARY  primarysrv      AdventureWorks    NULL
```

The important columns for determining the replication lag are:

> `last_replication` : The time when the primary received the acknowledgment that the last log block has been hardened by the secondary, based on the primary database clock. Log blocks are sent to the geo-secondary continuously, without waiting for transactions to commit on the primary. This value is available on the primary database only.
> `last_commit` : The time of the last transaction committed to the database. If retrieved on the primary database, it indicates the last commit time on the primary database. If retrieved on the secondary database, it indicates the last commit time on the secondary database. If retrieved on the secondary database when the primary of the replication link is down, it indicates until what point the secondary has caught up.
> `replication_lag_sec` : Time difference in seconds between the `last_replication` value and the timestamp of that transaction's commit on the primary based on the primary database clock. This value is available on the primary database only.
> `replication_state_desc` : The state of geo-replication for this database, one of "Seeding", "Pending", "Catch-Up", or "Suspended". "Catch-Up" is the normal status of a healthy replication.

If `replication_lag_sec` on the primary is NULL, it means that the primary does not currently know how far behind a geo-secondary is. This typically happens after process restarts and should be a transient condition. Consider sending an alert if `replication_lag_sec` returns NULL for an extended period of time. It may indicate that the geo-secondary cannot communicate with the primary due to a connectivity failure.

There are also conditions that could cause the difference between `last_commit time` on the geo-secondary and on the primary to become large. For example, if a commit is made on the primary after a long period of no changes, the difference will jump up to a large value before quickly returning to zero. Consider sending an alert if the difference between these two values remains large for a long time.

Mitigation

The following options might help with reducing or avoiding the impact:

- Make sure that both primary and secondary geo-replication partners operate with the same SLO. If the secondary has a lower SLO than the primary, then scale up the secondary to the same SLO than the primary.
- Reduce the workload, e.g. move some tasks off the peak times and distribute the load more evenly throughout the day or week.
- Scale up to a higher service level within the same service tier to get to the maximum log rate of the service tier, or switch to a different service tier. Scale the secondary first before scaling the primary for avoiding the "mismatched SLO" scenario.
- Consider moving to Hyperscale. The Hyperscale service tier provides 100 MB/s log rate regardless of the chosen service level.
- Consider moving the main Insert/Update workload into tempdb. This may help if the loaded data is transient, e.g. when staging data in an ETL process. Tempdb is minimally-logged and its content is not geo-replicated.
- For analytic scenarios, load into a clustered columnstore table or a table with indexes that use data compression. This reduces the required log rate. This technique however increases CPU utilization and is only applicable to specific data sets that benefit from clustered columnstore indexes or data compression.
- Managed Instance: Ensure that the blob performance tier for the transaction log files is at least P30 to mitigate XIO throttling. Smaller files may default to P10, which is rather inadequate for higher workloads. See File IO characteristics in General Purpose tier and Storage performance best practices and considerations for Azure SQL DB Managed Instance (General Purpose) for more information.

- Managed Instance: Consider switching to VNET global peering if NVAs are used to minimize network latency.

Internal Doc Reference

- IcM 312753011
- IcM 278421371

Public Doc Reference

- sys.dm_geo_replication_link_status

How good have you found this content?