Snapshot Agent failing Connection is broken

Last updated by | Holger Linke | Aug 2, 2022 at 1:51 AM PDT

Contents

Issue

The customer experienced issues when creating new Transactional Replication snapshots for their large Publisher database. Whenever they run the snapshot job, it runs for more than 6 hours before failing with a connection error. The issues is blocking the deployment of new subscriptions and the reinitialisation of failed existing subscriptions.

The Snapshot Agent execution details in Replication Monitor and in `distribution..MSsnapshot_history` are showing the following sequence:

```
(...)
[98%] Bulk copied 100000 rows from [dbo].[articlename] (205000000 total rows copied)
[98%] Bulk copied 100000 rows from [dbo].[articlename] (205100000 total rows copied)
[98%] Bulk copied 100000 rows from [dbo].[articlename] (205200000 total rows copied)
[98%] Bulk copied snapshot data for article 'articlename' (205212345 rows).
[99%] Disconnected from Azure Storage '\\storageaccountname.file.core.windows.net\snapshots' with OS result co
The connection is broken and recovery is not possible.  The connection is marked by the server as unrecoverabl
```

... with adding the command text of `exec sp_replincrementlsn` and the stack trace in the error details section in Replication Monitor:

```
Message: The connection is broken and recovery is not possible.  The connection is marked by the server as unr

Command Text: declare @lsn binary(10) exec sp_replincrementlsn @lsn OUTPUT select @lsn
Parameters:
Stack:    at Microsoft.SqlServer.Replication.AgentCore.ReMapSqlException(SqlException e, SqlCommand command)
   at Microsoft.SqlServer.Replication.AgentCore.AgentExecuteReader(SqlCommand command, Int32 queryTimeout, Com
   at Microsoft.SqlServer.Replication.AgentCore.ExecuteWithResults(CommandSetupDelegate commandSetupDelegate,
   at Microsoft.SqlServer.Replication.Snapshot.TransSnapshotProvider.SetPublisherTranSequenceNumViaReplIncLSN(
   at Microsoft.SqlServer.Replication.Snapshot.TransSnapshotProvider.DoNormalPostArticleFilesGenerationProcess
   at Microsoft.SqlServer.Replication.Snapshot.SqlServerSnapshotProvider.GenerateSnapshot()
   at Microsoft.SqlServer.Replication.SnapshotGenerationAgent.InternalRun()
   at Microsoft.SqlServer.Replication.AgentCore.Run() (Source: MSSQLServer, Error number: 0)
```

The same error as it appears on the SQL errorlog and the job history:

```
2022-01-05 16:27:31.26 spid103     Error: 14151, Severity: 18, State: 1.
2022-01-05 16:27:31.26 spid103     Replication-Replication Snapshot Subsystem: agent xxxxxx-35 failed. The rep
Source: Replication
Exception Type: Microsoft.SqlServer.Replication.ReplicationAgentSqlException
Exception Message: The connection is broken and recovery is not possible.  The connection is marked by the ser
```

Increasing the -QueryTimeout agent parameter to 65000 [seconds] = 18 hours did not help to mitigate the issue.

Investigation

Possible cause

The error message has two significant details:

- the error occurs after a large article has completed processing
- the command text of `exec sp_replincrementlsn`

The snapshot operations need to be inserted into the Distribution database for later consumption by the Distribution Agents. These snapshot commands are identified by their own Log Sequence Number (LSN) which is specifically generated for this purpose. This LSN generation is executed by `sp_replincrementlsn` on a connection into the Publisher database and requires an open transaction. This scenario may point to a connection that was cancelled by the Azure gateway because it had been idle for too long.

Troubleshooting

To find out more about the Snapshot Agent connectivity, you can filter Kusto's `MonLogin` for the agent's application name. The application name will follow the same pattern as the agent job name: "instancename.dns_suffix.database.-publisherdbname-publicationname-agentid" and you can search for it in the `MonLogin.application_name` field.

Get the Snapshot Agent job's start time and failure time from the customer and then search for it in MonLogin, like this:

```
// example: snapshot job name = MIHLI.8F08E6D34D3B.DATABASE.-Repl_PUB-Publication_Tran-1
let startTime = datetime(2022-04-21 12:00:00);
let endTime = datetime(2022-04-21 14:00:00);
let srv = "instancename";
let db = "publisherdatabasename";
MonLogin
| where TIMESTAMP >= startTime
| where TIMESTAMP <= endTime
| where logical_server_name =~ srv
| where database_name =~ db
| where application_name startswith "MIHLI.8F08E6D34D3B.DATABASE"
| project TIMESTAMP, logical_server_name, database_name, application_name, event, connection_peer_id, peer_act
| order by TIMESTAMP asc
```

It will provide you with distinct values for `connection_peer_id` and `peer_activity_id`. Use those to widen the search in MonLogin, like this:

```
// change start/end times, connection_peer_id, peer_activity_id
let startTime = datetime(2022-04-21 12:00:00);
let endTime = datetime(2022-04-21 14:00:00);
MonLogin
| where TIMESTAMP >= startTime
| where TIMESTAMP <= endTime
| where connection_peer_id =~ "482E2337-4FFA-431E-81AE-F48902A18289" or peer_activity_id =~ "BA3C6416-8F56-437
| project TIMESTAMP, logical_server_name, database_name, application_name, event, connection_peer_id, peer_act
| order by TIMESTAMP asc
```

In this resultset, take a look at the `process_close_connection` and `proxy_close_connection` events and the details in the `extra_info` column:

- A value like `"PrxyRdDnCli":"10054"` would indicate a normal client-side disconnect.
- But if you see something like `"PrxyWtDnSvr":"10054"` or any error, it would point to a server-side disconnect which can explain the error seen by the Snapshot Agent.

In this specific case, a `proxy_close_connection` event with `"PrxyWtDnSvr":"10054"` was found, confirming the server-side disconnect.

Analysis

In this specific case, the reason behind the failed snapshot generation attempts was in the engine-level transaction manager that interrupted long-running transactions after exceeding 6 hours of idleness.

The long-running transaction was held because the publication has been configured with the `@sync_method='native'` option. With `native`, the snapshot process will keep a SQL transaction and a lock on the source tables while the snapshot is created. Because the tables are very large, the snapshot took longer than 6 hours to complete. The idle transaction was then closed after 6 hours which ultimately caused the snapshot to fail.

Mitigation

Mitigation 1

If this is critical for the customer, open an IcM with the `SQL Managed Instance (CloudLifter) - Replication Expert` PG team. In IcM 281683524, it was possible to toggle a feature switch for disabling the `AbortIdleTransaction` feature, which allowed the Snapshot Agent to complete.

But this is only an emergency step, as the feature switch cannot be changed for all servers and regions. It will be only an individual, on-demand mitigation.

Mitigation 2 - preferred solution

To avoid this type of problems, you can change your publication to use the concurrent snapshot method `@sync_method = 'concurrent'` instead of `native`. With `concurrent`, the snapshot process will only hold a short lock at the start of the snapshot scripting process, but will then continue without holding any locks or transactions.

Please note though that `@sync_method = 'concurrent'` has other implications and is not a feasible solution for all scenarios. For example, it is prohibiting the Snapshot Agent from using parallel BCP threads, limiting it to a single BCP thread, which is a major drawback for large published data sizes. Also, if the publication is configured for Snapshot replication instead of Transactional replication (`sp_addpublication ... @repl_freq = 'snapshot'` instead of `@repl_freq = 'continuous'`), the `concurrent` option is not supported.

The recommended mitigation therefore is to try increasing the overall execution speed of the Snapshot Agent. See the related article Performance: Applying Snapshots to the Subscriber for further information about this topic.

Mitigation 3 - alternate solution

If the customer is able to redesign the publication, it might be possible to split it into separate publications. For example, if the largest tables would be moved into individual publications or sub-groups of tables, their snapshots could be generated in parallel or before or after all the other snapshots. That way, it might be possible to keep the individual snapshot execution time below the 6-hour mark, thus avoiding the long transaction disconnect issue.

The data at the Subscriber database would then be refreshed by applying the individual snapshots. Note there would be periods when some tables still have the previous data versions whereas other tables already have newer, updated data. The customer needs to make sure that the applications are aware of that possibility and take appropriate precautions in the application.

Public Doc Reference

sp_addpublication (Transact-SQL)
[@sync_method =] _'sync_method' Is the synchronization mode. sync_method is nvarchar(13), and can be one of the following values.

Value	Description
native	Produces native-mode bulk copy program output of all tables.
character	Produces character-mode bulk copy program output of all tables.
concurrent	Produces native-mode bulk copy program output of all tables but does not lock tables during the snapshot. Only supported for transactional publications.
concurrent_c	Produces character-mode bulk copy program output of all tables but does not lock tables during the snapshot. Only supported for transactional publications.

How good have you found this content?