Distribution cleanup Deadlocks with replication agents

Last updated by | Vitor Tomaz | Jun 8, 2022 at 5:37 AM PDT

Contents

Issue

The customer is running Transactional Replication on their Managed Instance, which is hosting both Publisher and Distributor roles. The `Distribution clean up: Distribution` job is executing on the default schedule of every 10 minutes, but when monitoring the cleanup job history, the cleanup process is regularly running into deadlocks and is chosen as the victim of a deadlock:

```
Executed as user: DB4C1\WF-oM3QjL6VozKSGhF.
Deleted 321 row(s) per millisecond from MSrepl_commands [SQLSTATE 01000] (Message 22121)
Deleted 123 row(s) per millisecond from MSrepl_transactions [SQLSTATE 01000] (Message 22121)
Msg 1205, Level 13, State 52, Procedure dbo.sp_MSdelete_publisherdb_trans, Line 259 [Batch Start Line 0]
Transaction (Process ID 352) was deadlocked on lock resources with another process and has been chosen as the
```

For troubleshooting purposes, the DBA executed the cleanup procedure `EXEC dbo.sp_MSdistribution_cleanup @min_distretention = 0, @max_distretention = 72` from a query window in SSMS, but the issue was still occurring repeatedly.

Investigation / Analysis

Technical Background

The cleanup stored procedure executes a `SET DEADLOCK_PRIORITY LOW` before it starts deleting any rows. This is the intended behaviour, as the cleanup in itself has a lower business priority than any of the processes it might collide with.

Occasional deadlocks do no harm, but if they occur too often, it may be impacting the overall cleanup performance. The risk then is that the incoming rate of replication metadata becomes larger than the cleanup rate, which then might further decrease the cleanup performance due to the increasing size of the system tables. In the worst case, the Distribution database and the Managed Instance might run out of storage space.

Get Deadlock report

To find out what is causing the deadlocks, you should run the ASC troubleshooter and check the deadlock page for details.

If you know the approximate time window when the deadlocks occurred, you can also retrieve the details through the following Kusto query:

```
MonDeadlockReportsFiltered
| where originalEventTimestamp >= datetime(2022-04-18 08:00:00)
| where originalEventTimestamp <= datetime(2022-04-18 14:00:00)
| where LogicalServerName =~ 'instancename' //and database_name =~ 'distribution'
//| where AppName =~  "d3bdedd61a17"
| where event =~ 'xml_deadlock_report_filtered'
| extend Complete_Deadlock_Graph = trim_end(@'[ \t\r\n]+', xml_report_filtered) endswith '</deadlock>'
| order by Complete_Deadlock_Graph nulls last, originalEventTimestamp nulls last
| extend Complete_Deadlock_Graph = iff(Complete_Deadlock_Graph == 1, 'true', 'false')
| project originalEventTimestamp, xml_report_filtered, NodeName, Complete_Deadlock_Graph
| extend query_hashes = extract_all(@'queryhash=\x22([^\s]+)\x22', xml_report_filtered)
| extend query_plan_hashes = extract_all(@'queryplanhash=\x22([^\s]+)\x22', xml_report_filtered)
// Remove 0x0 hashes and empty sets.
// 0x0 hashes represent frames that don't have a query or execution plan, e.g. exec proc frames inside trigger
//
| extend query_hashes = set_difference(query_hashes, dynamic(['0x0000000000000000']))
| extend query_plan_hashes = set_difference(query_plan_hashes, dynamic(['0x0000000000000000']))
| extend query_hashes = iif(array_length(query_hashes) > 0, query_hashes, '')
| extend query_plan_hashes = iif(array_length(query_plan_hashes) > 0, query_plan_hashes, '')
```

Analyze Deadlock details

The deadlock graph that is returned by the ASC Troubleshooter or `MonDeadlockReportsFiltered` is an XML string. It will show you the process ID of the deadlock victim at the top, followed by the individual processes that are involved in the deadlock, and the resources these processes are holding and attempting to get.

Here is an abbreviated example for the cleanup colliding with the Log Reader Agent:

```
<victimProcess id="process2001ff508c8"/>

<process-list>
  <process id="process2001ff508c8" taskpriority="5" logused="0" waitresource="PAGE: 16:1:92491251 " waittime="
  transactionname="DELETE" lasttranstarted="2022-04-18T14:12:37.677" XDES="0x22e83604420" lockMode="U"
  schedulerid="55" kpid="15412" status="suspended" spid="352" sbid="0" ecid="0" priority="-5" trancount="2"
  lastbatchstarted="2022-04-18T14:01:21.937" lastbatchcompleted="2022-04-18T14:01:21.930" lastattention="2022-
  clientapp="Microsoft SQL Server Management Studio - Query" hostname="filtered" hostpid="128260" loginname="f
  isolationlevel="read committed (2)" xactid="1967391462" currentdb="16"
  currentdbname="distribution" lockTimeout="4294967295" clientoption1="673187936" clientoption2="390168">

  <process id="process1e35ff17848" taskpriority="0" logused="292" waitresource="PAGE: 16:1:178247949 " waittim
  transactionname="user_transaction" lasttranstarted="2022-04-18T14:12:40.197" XDES="0x22eba2e0040" lockMode="
  schedulerid="10" kpid="22924" status="suspended" spid="341" sbid="0" ecid="0" priority="0" trancount="2"
  lastbatchstarted="2022-04-18T14:12:40.207" lastbatchcompleted="2022-04-18T14:12:40.197" lastattention="1900-
  clientapp="Repl-LogReader-0-publisher_pubdb-5" hostname="filtered" hostpid="28256" loginname="filtered"
  isolationlevel="read committed (2)" xactid="1967395907" currentdb="16"
  currentdbname="distribution" lockTimeout="4294967295" clientoption1="538968096" clientoption2="128024">
</process-list>

 <resource-list>
  <pagelock fileid="1" pageid="92491251" dbid="16" subresource="FULL" objectname="filtered" id="lock2061db6fd8
  <pagelock fileid="1" pageid="178247949" dbid="16" subresource="FULL" objectname="filtered" id="lock1f7f81302
 </resource-list>
```

You can read several details from this output for verification:

- The `victimProcess ID` is pointing to the first process in the list
- The **first process** is waiting on an Update lock for a data page in database_id=16, which is currentdbname=distribution
- The `transactionname` is "Delete", which confirms that it is the process for the cleanup
- The `priority` is "-5", which corresponds with `SET DEADLOCK_PRIORITY LOW`
- The `clientapp` is a query window in SSMS, because the customer ran the cleanup proc manually; you might also see the name of the cleanup job here instead
- The session ID `spid="352"` matches the spid returned in one of the error messages
- The **second process** runs in the same database and is attempting to get an Intent-Exclusive page lock
- Its `priority="0"` is higher than the priority of the victim
- The `clientapp` indicates that the process is the writer thread of the Log Reader Agent

Mitigation

Mitigation 1 - quick short-term attempt

If the deadlocks are seriously impacting the cleanup performance, you can try the following:

1. Stop the replication agents that are involved in the deadlocks (usually it is the Log Reader), or if possible, all replication agents (Log Reader and Distribution Agents).
2. *If you can stop all replication agents:* Increase the Delete batch sizes of the cleanup significantly. See the mitigation steps on Distribution cleanup Performance issues or Run the cleanup with increased Delete batch sizes for details. On the next execution of the cleanup procedure, either through the cleanup job or a manual execution, the larger batch sizes will be picked up. Don't forget to reset to the previous values once the cleanup has succeeded.

Mitigation 2 - worth a try

Change the schedule of the cleanup job to run every minute (default: every 10 minutes). This allows for the job to restart faster after a deadlock. You could also change the number of retries from the default of "0" to "10" (the option is available on the "Advanced" tab of the Job Step Properties).

This mitigation step might not help though, depending on how often the deadlocks occur; but it might run the cleanup often enough to improve the Delete rate.

Mitigation 3 - at your own risk

This is an unsupported option, which the experienced customer might want to try if they are really desperate. I'm mentioning this here because it is discussed in various forums on the internet.

The source code of the cleanup stored procedures is available in the Distribution database. You can see it for yourself by running:

```
sp_helptext sp_MSdistribution_cleanup
sp_helptext sp_MSdistribution_delete
sp_helptext sp_MSdelete_publisherdb_trans
sp_helptext sp_MSdelete_dodelete
```

The procedures `sp_MSdelete_publisherdb_trans` and `sp_MSdelete_dodelete` contain the actual Delete commands. They have a query hint of "WITH (PAGELOCK)". If this is changed to "WITH (ROWLOCK)", it would reduce the deadlock probability between cleanup procs and replication agents. But it would also increase the consumption of lock resources and thus might impact the Managed Instance in itself, especially if the instance is sized too small.

=> ***Note that if you mess up these stored procedures, you are on your own and nobody will help you.***

Public Doc Reference

Listing some articles for Azure SQL Database - there are similar articles for on-premise SQL server but these here are a closer match to Managed Instance:

Analyze a deadlock for Azure SQL Database

Analyze and prevent deadlocks in Azure SQL Database

Understand and resolve Azure SQL Database blocking problems

How good have you found this content?