Bidirectional Transactional Replication Considerations

Last updated by | Vitor Tomaz | Feb 24, 2023 at 3:30 AM PST

Contents

Considerations on Bidirectional Transactional Replication

This article is also published in our public blog at Bidirectional Transactional Replication Considerations. Its content can be shared freely with customer.

Considerations on Bidirectional Transactional Replication

Bidirectional transactional replication is a specific Transactional Replication topology that allows two SQL Server instances or databases to replicate changes to each other. Each of the two databases publishes data and then subscribes to a publication with the same data from the other database. The "@loopback_detection" feature ensures that changes are only sent to the Subscriber and do not result in the changes being sent back to the Publisher.

The databases that are providing the publication/subscription pairs can be hosted either on the same SQL instance or on two different SQL instances. The SQL instances can either be SQL Server on-premise, SQL Server hosted in a Virtual Machine, SQL Managed Instance on Azure, or a combination of each. You just have to make sure that the instances can connect to each other.

Refer to article Bidirectional Transactional Replication for a detailed, commented sample script about configuring bidirectional Transactional Replication.

But the important question is: If you ***can*** do it, then ***should*** you do it? See the following paragraphs for a discussion about the advantages, risks, and potential pitfalls of Bidirectional Transactional Replication.

The Good

- It is the only available option to implement an Active-Active, two-node Transactional Replication scenario on Managed Instance. Transactional Replication in its standard configuration assumes that the Subscriber is read-only, and the bidirectional extension allows you to replicate changes from the tables at the Subscriber back into the Publisher tables. In SQL Server on-premise or Virtual Machines, you can instead use the multi-node capabilities of Peer-to-Peer Transactional Replication. But Peer-to-Peer is not available on Managed Instance so far.

- It supports high-throughput OLTP application workloads without much of a synchronization issue. Replicated changes are applied through Stored Procedures, using the Primary Key to apply data changes row-by-row. As long as the OLTP application doesn't block large portions of the tables, normal production workload and Transactional Replication won't interfere with each other.

- It can handle potentially-conflicting changes on either node if there is a sufficient time difference between the changes on either side. The Distribution Agent applies changes either continously or on a schedule, usually every 10 minutes or less. Conflicts can usually be avoided if conflicting changes don't occur within 10 minutes of each other, or some more time if you include a buffer for safety. The second node needs to receive the change from the first node before it can safely change the same row again. This makes it suitable for user groups located on different time zones without much overlap, as long as the subscriptions are able to synchronize in a timely manner. But there is an imminent high risk - read the sections below to understand the impact.

The Bad

- It only supports two nodes and doesn't scale further beyond that.

- It supports simultaneous writes on both nodes, but doesn’t have any conflict resolution mechanism. Conflicting writes will cause replication failures and/or inconsistencies, and failures might not be easily repairable. There are two approaches to avoid conflicts: the timely separation of write operations (as mentioned above), or to separate the administrative ownership of a data row between the two nodes (only the owning node is allowed to change the row). This can be achieved either from within the applications or by adding identifying information to the table schema.

- It requires Identity range management if Primary Keys are using `IDENTITY` property columns. You need to avoid that rows inserted on either node receive the same Primary Key value. Possible workarounds are: (1) using GUIDs as Primary Key, or (2) allowing inserts only to occur on one node (preferably the initial Publisher to simplify disaster recovery), or (3) creating a composite Primary Key with the second component indicating the originating node.

The Ugly

- It cannot handle Delete-Update conflict scenarios. If Node1 updates a row at the same time when Node2 deletes the same row, the Distribution Agent applying the update to Node2 will fail with error 20598 "row not found". At the same time, the Distribution Agent applying the delete to Node1 will succeed. You will have data loss on Node1 with a broken Node1-to-Node2 replication. It might be possible to repair the failure by inserting a dummy row on Node2 to let the delete succeed. But this will not be possible if the delete was just one of many deletes, or when the supposed conflict winner should have been the update.

- It requires a clear disaster recovery plan before going into production. You need to consider having to re-establish replication from a situation where both nodes have different data. Applying a snapshot from Node1 to Node2 will overwrite pending data changes at Node2 and usually is not feasible. You need to plan for bringing both nodes back into sync with the same data on both nodes. These are administrative design decisions that you need to take in the planning and implementation phase, well before going live to production.

In summary

If bidirectional transactional replication is suitable for your business needs depends on your applications and the user requirements. The important concern is the consideration on disaster recovery. You need to have a clear plan beforehand and run appropriate tests, including making every possible attempt to break it. That will give you a clear understanding about what can go wrong, and how to fix it if it has gone wrong. Failure of doing so might leave you with a severe downtime and possibly data loss later on.

How good have you found this content?