OPTIMIZATION OF DISTRIBUTED QUERY USED IN SYNCHRONIZING DATA BETWEEN TABLES WITH DIFFERENT STRUCTURE

Demian Horia
Faculty of Economics
University of Oradea

Replication can be used to improve local database performance and to improve the availability of applications. An application can access a local database rather than a central database from another site, which minimize network traffic, locking escalation at the central database and achieve maximum performance for current insert, delete or update operations. The application can continue to function if the central database is down, or cannot be contacted due to a communication problem, power or hardware failure. This paper is focused on presenting a synchronization process between a central Microsoft SQL Server database and many remote sites databases. One possible problem in replication can appear when the two databases have different organization of tables and structures.
Keywords: replication, data synchronization, openquery, linked server, parameterized query, uniqueidentifier, SQL Server

In this paper we will focus on the problem of synchronizing data between a central database and many client databases, when the client and the server databases have different organization of tables and structures. All the client databases are identical, this means that have the same number of tables, the tables have the same structure, but the records are different. This model can be used to improve local database performance and availability of applications at the clients, because we access the local database rather than a central database. Other benefits are the minimization of network traffic between the client and the central location, the minimization of locking at the central database, the maximization of up time because we does not need a permanent communication channel between the clients and the central location and also the minimization of costs.

The architecture of the network

The network consists of one Central Server on which we have the central database and many client servers, each of them with its own database used in the local area for storage the local transactions, like in figure 1. Between the client servers and the central server communications channels had been made by using VPN (Virtual Private Network). This communication channels are most of the time down, and from time to time get connected.

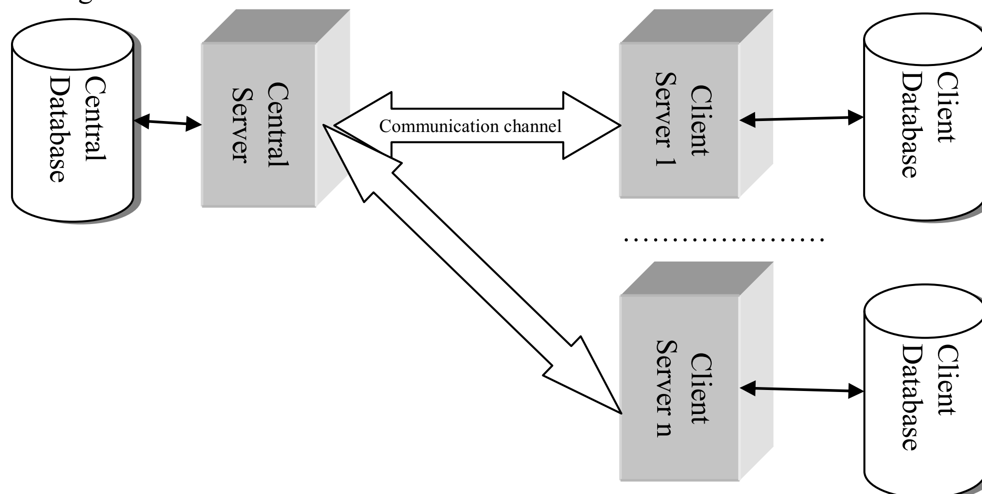

Figure 1. The architecture of the network

If we want to synchronize the information from client database with the server, we have to open first the communication channel and after that to synchronize the data. Microsoft SQL Server offers a mechanism of defining linked servers which can be used to run distributed query against multiple server. Our scenario consists of a one way data transfer from Central Database to the Client Database. We have to obtain new or modified data from the Central Database using inner join operations between TableA and TableB and to copy these data in TableC from Client Database. The fields used for filtering are of uniqueidentifier data type.
We want to obtain the best performance for our process of synchronizing the data.
One syntax for doing this operation is:

```
    Select A.somefields, B.somefields
        From CentralServer.CentralDatabase.dbo.TableA A
        Inner Join CentralServer.CentralDatabase.dbo.TableB B
        On A.idProduct = B.idProduct
        And B.FilteringField = @SomeValue
    Where not exists (Select * from TableC C
                        Where C.idProduct = A.idProduct )
```

In our studying case TableA contains 1915 records, TableB contains 7341 records and TableC contains 1664 records. We obtain 243 new records which have to be copied in TableC. In our test environment this query takes about 47 seconds. The explanation for this bad performance are given to us by the following sentence “Queries involving the following are never delegated to a provider and are always evaluated locally: bit, uniqueidentifier” (Microsoft), which means that the records are taken from the Central Database to the client, and all operations are done at the client. For a poor connection we lose precious time with data transportation.

<table>
<tr><th></th><th>Rows</th><th>Executes</th><th>StmtText</th><th>PhysicalOp</th><th>LogicalOp</th><th>Argument</th></tr>
<tr><td>1</td><td>243</td><td>1</td><td>select P.codprodus from . . .C...</td><td>NULL</td><td>NULL</td><td>NULL</td></tr>
<tr><td>2</td><td>243</td><td>1</td><td>|--Hash Match(Right Anti Semi Joi...</td><td>Hash Match</td><td>Right Anti Semi Join</td><td>HASH:([p</td></tr>
<tr><td>3</td><td>1664</td><td>1</td><td>|--Index Scan(OBJECT:([vanza...</td><td>Index Scan</td><td>Index Scan</td><td>OBJECT:(</td></tr>
<tr><td>4</td><td>1907</td><td>1</td><td>|--Merge Join(Inner Join, ME...</td><td>Merge Join</td><td>Inner Join</td><td>MERGE:([</td></tr>
<tr><td>5</td><td>1915</td><td>1</td><td>|--Remote Query(SOURCE:...</td><td>Remote Query</td><td>Remote Query</td><td>SOURCE:(</td></tr>
<tr><td>6</td><td>1907</td><td>1</td><td>|--Filter(WHERE:([...</td><td>Filter</td><td>Filter</td><td>WHERE:([</td></tr>
<tr><td>7</td><td>7341</td><td>1</td><td>|--Remote Query(SO...</td><td>Remote Query</td><td>Remote Query</td><td>SOURCE:(</td></tr>
</table>

Figure 2. Execution plan

The execution plan demonstrates that two remote query are executed against the linked server for bringing locally data from TableA and also from TableB, and only after that the inner join operation are executed at the client site. From this execution plan we understand that 1915 +7341 records are moving from central database to the client site.
One thing we have to optimize is that of forcing to execute inner join operation at the Central Server site. This can be done if we define a view in the CentralDatabase or if we use OPENQUERY syntax to send queries which will be executed at the central database.

```
Select *
    from OPENQUERY(CentralServer,'Select A.somefields, B.somefields
        from CentralDatabase.dbo.TableA A
        inner join CentralDatabase.dbo.TableB B
        on A.idProduct = B.idProduct') As R
where R.FilteringField = @SomeValue
    and not exists (Select * from TableC C
                        Where C.idProduct = A.idProduct )
```

Because OPENQUERY does not accept variables for its arguments, we have to write filtering condition outside. The execution plan demonstrate us that only 7341 records are coming from the central database which resulted in better performance, which means 43 seconds.

	Rows	Executes	StmtText	PhysicalOp	Argument
1	243	1	Select * from OPENQUERY(;,'...	NULL	NULL
2	243	1	|--Hash Match(Right Anti Semi Joi...	Hash Match	HASH:([p2
3	1664	1	|--Index Scan(OBJECT:([vanza...	Index Scan	OBJECT:([
4	1907	1	|--Filter(WHERE:([A].[idGest...	Filter	WHERE:([A
5	7341	1	|--Remote Scan(SOURCE:(...	Remote Scan	SOURCE:(D

Figure 3. Execution plan

The next big thing to optimize is that of execution of filtering condition at the central database to obtain only 1907 records instead of 7341. If we know exactly the value used for filtering we can write in the OPENQUERY the filtering condition.

```
Select *
        from OPENQUERY(CentralServer,'Select A.somefields, B.somefields
                from CentralDatabase.dbo.TableA A
                inner join CentralDatabase.dbo.TableB B
                on A.idProduct = B.idProduct
                and A.FilteringField =VALUE
                ') As R
where
        not exists (Select * from TableC C
                        Where C.idProduct = A.idProduct )
```

The time needed for executing this query was 7 seconds, and the execution plan can be analyze from the figure 4.

	Rows	Executes	StmtText	PhysicalOp	Argument
1	243	1	Select * from OPENQUERY(;,'...	NULL	NULL
2	243	1	|--Hash Match(Right Anti Semi Joi...	Hash Match	HASH:([
3	1664	1	|--Index Scan(OBJECT:([vanza...	Index Scan	OBJECT:
4	1907	1	|--Remote Scan(SOURCE:(DOUAR...	Remote Scan	SOURCE:

Figure 4. Execution plan

As we can see only 1907 records came from the central database, which is an explanation for our best results until now.

Because OPENQUERY does not accept variables for its argument, we can construct the sting dynamically and executes after that like in the following example:

```
DECLARE @SQLSTMT                nCHAR(4000)
DECLARE @SOMEVALUE              uniqueidentifier
--we can put here a code for initializing the value of our variables
SET @SOMEVALUE = '341FE6DA-D431-4D72-93F0-E20B2CDB1767'

SET @SQLSTMT =N' Select * '
+'from OPENQUERY ( CENTRALSERVER, ''Select A.somefields, B.somefields '
+' from CentralDatabase.dbo.TableA A'
+' inner join CentralDatabase.dbo.TableB B '
```

```
+' on A.idProduct = B.idProduct '
+' and B.FilteringField = ''''+cast(@idSomeValue as char(36))+''''') As A '
+' where not exists ( select * from TableC C '
+' where C.idProduct = A.idProduct) '

EXECUTE SP_EXECUTESQL @SQLSTMT
```

The execution plan for this command can be viewed in the following figure, and we can see the same results as in the last example. The execution of the Join operation and the filtering occurs on the central server. We achieve the same results.

	Rows	Executes	StmtText	PhysicalOp	Argument
1	243	1	Select * from OPENQUERY(,'s...	NULL	NULL
2	243	1	|--Hash Match(Right Anti Semi Joi...	Hash Match	HASH:([p
3	1664	1	|--Index Scan(OBJECT:([vanza...	Index Scan	OBJECT:(
4	1907	1	|--Remote Scan(SOURCE:(DOUAR...	Remote Scan	SOURCE:(

Figure 5. Execution plan for dynamic query

The speed is well for the preceding examples, but we can improve performance, if we can rewrite the query without making reference to a table from the client site.

To do this we have to kow for every records from the central site for every records involved in the query, the date when the record was added or modified. This date has to be copied to the client site.

```
DECLARE @SQLSTMT          nCHAR(4000)
DECLARE @SOMEVALUE        uniqueidentifier
DECLARE @DATA             datetime

SET @SOMEVALUE = '391FE6DA-D431-4D72-93F0-E20B2CDB1767'

SELECT @DATA =MAX(DATA)
    FROM TABLEC

    SET @SQLSTMT =N' Select * '
    +'from OPENQUERY ( CENTRALSERVER, ''Select A.somefields, B.somefields '
    +' from CentralDatabase.dbo.TableA A'
    +' inner join CentralDatabase.dbo.TableB B '
    +' on A.idProduct = B.idProduct '
    +' and B.FilteringField = ''''+cast(@SomeValue as char(36))+''''
    +' and P.Dataoperarii >=''''+cast(@data as char(20))+'''''
    +') As A '

EXECUTE SP_EXECUTESQL @SQLSTMT
```

We obtain 1 seconds for 243 records, the best results.

	Rows	Executes	StmtText	PhysicalOp	d
1	243	1	Select * from OPENQUERY(,'...	NULL	
2	243	1	|--Remote Scan(SOURCE:( ),...	Remote Scan	

Figure 6.Execution plan on the linked server

Another benefit of dynamically creation of sql commands consist of construction of a mechanism for calling function from linked server, like we can see in the following example:

```
DECLARE @SQLSTMT            nCHAR(4000)
DECLARE @idParameter        uniqueidentifier
--we can put here a code for initializing the value of our variables
SET @ idParameter = '6EDAE050-F5EA-4B1C-823B-28B8F599FDC9'

SET @SQLSTMT =N' Select * '
    +'from OPENQUERY(DOUAROTI,''select * '
    +' from DataBase.dbo.FunctionName( ''''''+cast(@idParameter as char(36))+'''''')'' )  '
 --we used only apostrophes

EXECUTE SP_EXECUTESQL @SQLSTMT
```

Conclusion:

1. Even if OPENQUERY does not accept variables for its argument we can build dynamically the command and executed against the linked server.
2. For better performance we have to limit the number of records which are transported from one server to other.
3. It's better to force the execution of JOIN operation at the sites where the tables are located, if we need better performance
4. Using OPENQUERY we can call function from the linked server, which can used variables.
5. The best results can be achieved if all operations will be executed on the central server.

Bibliography:

26. Microsoft. (n.d.). Optimizing Distributed Queries. Retrieved may 20, 2010, from msdn: http://msdn.microsoft.com/en-us/library/Aa178113
27. Support, M. (2003, October 3). PRB: User-Defined Function Call in Four-Part Linked Server Query Fails with . Retrieved April 20, 2010, from Support: http://support.microsoft.com/kb/319138
28. Team, Q. O. (2006, April 6). Tips, Tricks, and Advice from the SQL Server Query Optimization Team. Retrieved May 20, 2010, from msdn: http://blogs.msdn.com/queryoptteam/