Platform Independent-Information System Dependant Database Replication

Tatjana Stanković[1], Milan Stanković[2], Dragan Janković[3]

Abstract – **All well-known database replications are practically impossible in low-speed/low-band networks like modem or even ADSL connections. Existing replication solutions are all platform dependant and applications independent. One different approach is presented in this paper. Described approach, like making replication dependant on information system and independent on database platform, showed that replication could work many times faster, and could be far more simple then existing**

Keywords: **Database replication**

I. INTRODUCTION

Replication is the best way to copy data from one database to another. Its' exact purpose is *coping data and database objects* (views, stored procedures, etc) between servers [1]. Common replication between dislocated databases of one large company requires reliable high-speed Internet connections. Fortunately, there are many parts of the world not jet technically in the position for on-line netting. In our environments, for example, high-speed on-line Internet connections (like optical) can be very expensive for some companies. Modem connection is often the only way for linking such places. This happens to be a significant problem for companies that have their regional offices on such locations. How to transfer data from one company location to another, without manually exporting, copying files and importing data? How to avoid engagement of trained experts and database administrators in thins process? And why avoiding this way of data transferring?

For the needs of small organizations, the way of transferring data mentioned above can be satisfied (a few exports-imports per day). But in large companies that have many tents or hundreds of DBMS servers, this way of transferring can take hundreds of business hours per day. The amount of data for transferring is often too large for mail, web upload, ftp, etc. And the last, but not the least, data transferred this way are usually packed into files that requires parsing and additional processing from the other side, what enlarges the cost of transfer by increasing time of importing.

Example of organizations with replication problems in our environment are wholesale-retail trade companies, or public companies like Electic Power Industry. They usually have a great number of small regional offices all over the country. These companies organization structure can be modeled by n-ary tree. The root of this tree is Central Organization Unit,

[1]Tatjana Stanković and [3]Dragan Janković are from Faculty of Electronic Engineering, Niš, {tanja, gaga}@elfak.ni.ac.yu

[3]Milan Stanković is from POWERSOFT d.o.o. Nis, Serbia, milanst99@yahoo.com

usually supporting the largest amount of processing and storing data. This unit hosts companies central DBMS. The other opened question according to these kinds of companies is: how to avoid an accumulation of unnecessary data on wayside servers? For quick and efficient business operations they only need parts of database relations.

The goal of this paper is to define problem's features of large company's database replication that occurs in low-band/low-speed network connections, and to propose one possible solution of this problem. The assumption of database replication problem referred to one large company with n-ary tree modeled organization structure is described in chapter two. Section three demonstrates the review of the most well-known replication solutions, and the reasons why they can not be used in our environments. Section four proposes one possible database replication solution that operates in low-band and low-speed network connections, and describes this solution's prototype implemented in one larger company in our market.

II. TYPYCAL DBMS SERVER'S STRUCTURE OF LARGE ORGANIZATIONS

The most common model of large companies' organization structure is n-ry tree. The root of this tree represents Central Organization Unit that runs company politics and trading. The second level tree nodes usually correspond to the company's regional offices. Their children can be their stacks or markets, or some smaller regional offices again, etc. All variations according to node types are possible. Tree level number, theoretically unlimited, in practice it is usually 3-10 (Fig. 1).

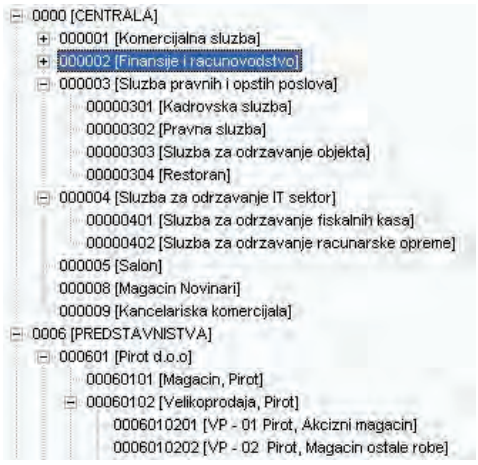

Fig. 1. Example of one company organization structure

Data processing in such companies requires extremely high-speed and reliable network resources, like optical links

between distant locations. In these cases the network infrastructure cost can be extremely large. So quick and reliable database replication, that does not demand such infrastructure, can significantly reduce company's expenses.

DBMS server's structure does not necessary concur with company's organization structure. One DBMS is commonly responsible for data storing and data processing of one subtree. The existence of as less as possible DBMS servers is pretended to. One typical DBMS server's structure is represented in Fig. 2.

The strongest demand for database replication is enounced between directly connected nodes of this structure. Company's business dealing itself is represented in a way that regional offices need to have all data derived from their child nodes (because they play part of master-offices to child nodes), and to send them data according to their roles in the system. Central organization Unit need to dispose with data derived from all sub-trees of the organization structure. According to this, dataflow need to run from Central DBMS to the leaves and inversely, but strictly through the tree branches (because not any child organization unit can do business without master organization unit knowing for it). If there is a request for dataflow between nodes that are not directly connected, it will be processed through the shortest possible way between linked nodes (Fig. 2, dot-stroke line).

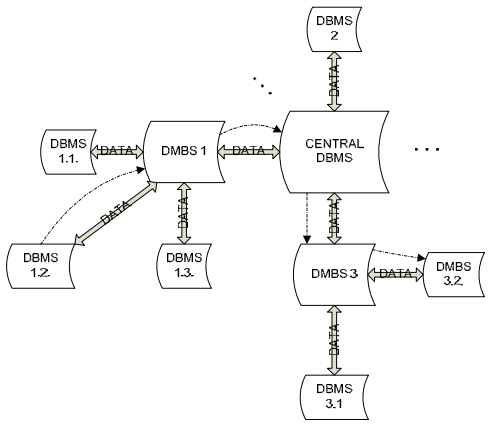

Fig. 2. DBMS server's structure and dataflow

III. Well-Known Replication Solutions In Low-Band/Low-Speed Networks

Let us give a quick review of well-known replication solutions like SQL Server's and Oracle's are. Comparing existing database replication solutions requires definition of three factors:

1. **Autonomy** – the possibility of changing data on deferent servers.
2. **Latency** – elapsed time before one server gets updates from another server.
3. **Consistency** – equality of servers' data.

SQL Server provides three kinds of replication: *Snapshot*, *Transactional*, and *Merge*. The expert level of database administration is required by all three kinds. Three servers need to be set: *Server Publisher*, *Server Subscriber* and *Server Distributor* (One single SQL Server can play part of all three).

Snapshot replication acts in the manner its name implies. The publisher simply takes a snapshot of the entire replicated database and shares it with the subscribers. Of course, this is a very time and resource-intensive process. For this reason, most administrators don't use snapshot replication on a recurring basis for databases that change frequently [2]. It has medium latency, high consistency and high autonomy.

Transactional replication offers a more flexible solution for databases that change on a regular basis. With transactional replication, the replication agent monitors the publisher for changes to the database and transmits those changes to the subscribers. This transmission can take place immediately or on a periodic basis. It requires high-speed and reliable network links and provides high consistency, low autonomy and medium latency.

Merge replication allows the publisher and subscriber to independently make changes to the database. Both entities can work without an active network connection. When they are reconnected, the merge replication agent checks for changes on both sets of data and modifies each database accordingly. If changes conflict with each other, it uses a predefined conflict resolution algorithm to determine the appropriate data. Merge replication is commonly used by laptop users and others who can not be constantly connected to the publisher [2]. It provides the highest autonomy degree, great latency and the lowest consistency.

In the conditions of low-band low-speed network connections, and frequently database changes (few hundreds inserts or updates per table), according to time of synchronization, neither of these SQL Server solutions provides satisfactory results. We must say that SQL Server replication is a fantastic piece of functionality but can lead to a database administration nightmare. Data conflicts are a common occurrence and require constant attention [3]. The third important reason for not using SQL Server replication in our environments is this: SQL Server additionally ballasts database with system relation and attributes added for replication. In some companies (in effort to avoid expenses) wayside database servers are ordinary PCs. This adding can significantly slow down system: for example, if you replicate one *master* server with 20 *child* servers that means that every table taking part in the replication gets additional 20 fields.

Oracle at first separates replication in *synchronous* (all database changes are propagated at once) and *asynchronous* (database changes propagated on demand). Oracle provides three kinds of replication: *Multimaster*, *Snapshot*, and *Multimaster and Snapshot Hybrid Configuration* [5].

Multimaster replication (also called peer-to-peer or *n*-way replication) allows multiple sites, acting as equal peers, to manage groups of replicated database objects. Each site in a multimaster replication environment is a master site. Applications can update any replicated table at any site in a multimaster configuration. Oracle database servers operating as master sites in a multimaster environment automatically work to converge the data of all table replicas and to ensure global transaction consistency and data integrity. Insufficiency: At times, you must stop all replication activity

for a master group so that you can perform certain administrative tasks on the master group. For example, you must stop all replication activity for a master group to issue data definition language (DDL) statements on any table in the group.
A **snapshot** contains a complete or partial copy of a target master table from a single point in time. A snapshot may be read-only or updateable. All snapshots provide the following benefits: Enable local access, which improves response times and availability; Offload queries from the master site, because users can query the local snapshot instead; Increase data security by allowing you to replicate only a selected subset of the target master table's data set. To ensure that a snapshot is consistent with its master table, you need to *refresh* the snapshot periodically. Every table taking part in the snapshot replication on Oracle gets one *snapshot log table*. A *snapshot log* is a table that records all of the DML changes to a master table. Oracle Snapshot replication provides low (read-only) and high (updateable) autonomy, high latency and lower consistency comparing to Multimaster.

Multimaster replication and snapshots can be combined in **Hybrid** or "mixed" configurations to meet different application requirements. Mixed configurations can have any number of master sites and multiple snapshot sites for each master. Replication conflicts are detected and resolved on master servers. Administration of this kind of replication requires highest Oracle experts. Autonomy, latency and consistency depend on the combination of multimaster and snapshot replication.

In low-band/low-speed networks all these kinds of replication are difficult to set, to maintain, and finally to work properly (according to synchronization time). All jobs like logging data form transfer, resolving replication conflicts, and transferring data between servers, are leaved to the Replication Agents (programs or DBMS processes responsible for replication). Implementing and maintaining those kinds of replication requires high-experts teams, and significantly increases replication costs.

IV. Platform Independent-Information System Dependant Database Replication Prototype

After a few months of researching about world known replication solutions, and according to benefits as well as imperfections of those where implemented in inconvenient networking conditions, this papers authors decided to develop their own replication solution that would be satisfactory to the following requests:

1. Short time of databases synchronization in low networking connections (even modems).
2. Highest speed of databases synchronization in high-speed Internet connection.
3. Reliable data filtering for transmission, in a way that one DMBS in the server tree stores and gets only data needed for that organization structure's branches, without any unnecessary data ballast.
4. The simplicity of setting and maintaining.
5. The simplicity of starting replication process, in a way that system users can do themselves.
6. Minimal ballast of replicated databases with additional relations and attributes.
7. Database platform independency.
8. Central DBMS has to get all of companies' data at the end.

Part of the replication processes were moved to the authority of The Information System, according to the first request. In the way that existing replications work, replication its self is absolutely transparent to the applications. In our solution, every application of the System is aware of the servers tree, and replication between those servers. This step made replication system to become dependant of companies IS, but it contributed to solving request 1 and 7 from the list. This step needed a small intervention of companies IS (as long as IS was multi-layer system, i.e. it had separate *data layer*).

This solution concept is:

- Every database in DBMS servers structure gets only five additional tables for replication:
1. *Server* table – stores information about DBMS server's structure. The number of these tables' rows is equal to the number of servers in the server's tree. This table is the same in every database from DMBS's tree. It is also replicated.
2. *Log table* – table that stores every UPDATE, INSERT or DELETE statement executed on database, together with the current timestamp of their execution.
3. *Two additional tables* that keep information of every synchronization moment with other servers connected to that particular server, one for sending and other for receiving data.
4. *Filter table* – table that stores information about tables taking part in the replication.

- Data is replicated in both directions. Taking over and forwarding data are two separate functional steps, so client does not need to wait double if he expects an important data transmission, he can only take over data.

- Replication conflicts are avoided with primary key offsets. Every server in the servers tree has its key offset of 10 million keys per table. Central server has 100 million keys per table. This should be enough for a year in every company, but it can be modified according to the amount of company's data. Information about key offsets are also stored in databases (and replicated), first table.

- A great autonomy degree is provided, and control about grants over changing data is taken by IS. There are two replication agents. *Rpull agent* is client-agent that demands replication, and *Rpush agent* is server-agent that takes or forwards data. Those agents are placed on both sides of replication, and they communicate over TCP/IP through Win sockets, just like Internet browsers and servers (Fig. 3). They can be installed on any computer (not specially server). Server-agent is running as system service. It is listening for Client-agent's demands on the particular port (determinated in advance) all the time.

- Database for replication is not determined in advance for Server-agent. Rpush agent gets information what database it is going to replicate, and where is that database, through the replication demand coming from Client agent. One Rpush agent can take care of many different replications at the same moment that way.

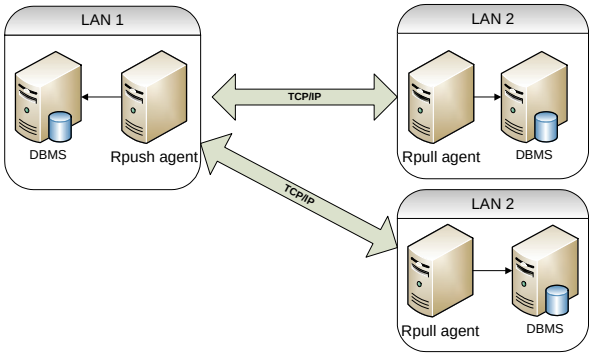

Fig. 3. Communication between replication agents

- Unlike other replication solutions, this one has data transit determined in advance. The pockets of data that circulates through the Network carries with them information about determination server. This information is initially stored by the Information System. For example, if one regional office sends merchandise to the other that is placed in another branch of the organization structure, the application (aware of the server's tree existing) writes to log table address of determination server. Log table stores information about source server too, so that data could not be given back to the server that had it at first place (which could result in a replication conflict).

- Agents work as following: when Rpull agent gets verification from Rpush agent that he is connected, it sends information about database he want to synchronize to. Rpush makes connection string to the wanted database, and sends confirmation. For example, Rpull wants to take data over from Rpush. Rpush agent selects all statements from log table in database according to the defined algorithm, and packs them to text file along with determination server information attached to every statement. File is then compressed and sent to the Rpull agent. When Rpull gets file, it extracts it, parses and simply executes statements. Rpull decides whether to log statements for further synchronization or not, according to the predefined algorithm.

Described solution prototype was made by this paper's authors. It has been tested for seven months in real conditions of one large company on our market. 51 companies DMBS server has been successfully connected in the server's tree with this prototype (like presented in Picture 2). For the needs of Central server's replication two Rpush agents were installed on two independent computers in LAN. Theoretically one Rpush agent could serve the unrestricted number of Rpull's request at the moment, but we restricted it to five. After 7 month of testing and improving it, our solution's prototype showed following performances: 1) about 200 000 data changes or inserts are delivered to the Central server per day. One Rpull client asks for Rpush agents favors more then one time a day. One synchronization with one server's subtree lasts for 2 (ADSL) to 10 (MODEM) minutes (depending on subtree's business amount of data for that day). 2) Replication conflicts are partially avoided by key offsets, and the other part is resolved in advance by Information System (taking care who can change what data). Considering that IS is fully aware of replication server's tree existence, it has information about source server for every data row in database (according to primary key offset). So it is easy to IS to determine if someone can change some data. The solution showed excellent performance in replication conflicts, considering that there were no conflicts at all. 3) The solution performed great autonomy degree, high consistency, and medium latency.

V. CONCLUSION

Completely new approach to database replication is exposed in this paper, and it is based on dependency of Information System. The basic idea was: Why wait to come to replication and then make moves, why not taking care of replication in advance. In other words, why not make Applications while working to prepare data that can be used for database replication. Storing that data in the same database and few more steps made this replication solution completely independent on database platform. It can work on *PostgreSQL*, *Oracle*, *SQL Server*, *MySQL*, etc, but only with specialized IS.

This paper exposes basic concept of replication solution projected and developed (prototype) after ten months of researching for satisfying existing solution. Researching and practical experience in applying described concept showed that proposed replication can seriously reduce one company's expenses according to network infrastructure, and database administration, extremely in companies that have many small distinct regional offices. Anybody (does not need to be an expert) can perform such replication, because it was reduced to two button-clicks for common Information System users. We can conclude that making database replication dependant on applications made it extremely simple and quick for maintaining and performing, and made the minimum of additional ballast on databases themselves. Also, it made it Database Platform independent. That way, this kind of replication can be performed successfully on weak wayside database servers, and in low-speed/low-band network connections.

REFERENCES

[1] M. Gunderloy, J.L. Jorden, SQL Server 2000, Mikro knjiga, 2000.
[2] M. Chapple, SQL Server Replication, http://databases.about.com/cs/sqlserver/a/aa041303a.htm
[3] SSW SQL Total Compare - Utility to Manage SQL Server Replication, http://www.ssw.com.au/ssw/SQLTotalCompare/
[4] http://www.dbasupport.com/oracle/ora9i/ors.shtml/
[5] Oracle8i Replication Release 2 (8.1.6), http://www.cs.bris.ac.uk/maintain/OracleDocs/server.816/a76959/repover.htm