Platform Independent Database Replication Solution Applied to Medical Information System

Tatjana Stanković[1], Srebrenko Pešić[1], Dragan Janković[1], and Petar Rajković[1]

tatjana.stankovic@elfak.ni.ac.rs
srebrenko.pesic@domzdravljanis.co.rs
{dragan.jankovic, petar.rajkovic}@elfak.ni.ac.rs

Abstract. Medical Information Systems (MIS) in general, like any other complex and large system, can be based on centralized or distributed DBMS. Centralized system is much easier for implementation and administration, but the benefits of distributed over centralized system are various. Load balance of the system is evenly, and when it comes to the internet connections interruption, system continues to work inside its remote nodes. However, database replication between DBMS's, considering the fact that larger Health Care Facilities in Serbia are consisted of at least 70-80 remote clinics, can be a huge problem. All well-known database replications are platform-dependent, and when it comes to the possible replacement of the platform the world known problem of setting replication on different platforms is born (MS SQL Server, Oracle, PostgreSQL, MySQL…). Considering that, we have developed one platform-independent IS-dependent database replication solution that can enable fast replication even in low-band/low-speed internet connections. This paper briefly describes the way information system takes care of the replication together with two developed replication agents, which are very easy to deal with. This solution can be applied not only to MIS, but to any IS with at least three-layer architecture, because the only adjustment need to be done in the information system is the adjustment of the data layer.

Keywords: Database replication, replication conflicts, distributed DBMS

1 Introduction

Modern society implies the existence of highly developed public health system with a strong technical and organizational background. Main building blocks of mentioned technical background are effective and reliable Medical Information Systems in active use within healthcare facilities having high level of interoperability [1].

In general, MIS can be based either on centralized or distributed database management system (DDBMS), like any other complex and large IS. According to the rules that Serbian Ministry of Health enacdet [2], both are allowed, as soon as the request that "the system must be available 24 hours a day, 365 days a year" is satisfied. Centralized system has many advantages compared to DDBMS. All data is stored at one place, and it is much easier for implementation then later one. Database and privileges are to be set only once. Administration is limited to only one DBMS,

which is less expensive for system's maintenance. However, centralized systems are more vulnerable then distributed in a sense of connectivity. Larger healthcare centers in Serbia are usually consisted of one central unit and several tens of remote nodes (small clinics). There are local ambulates in primary and secondary schools, small villages, etc. Not all of them have the possibility to connect to the high-speed internet connections, and even if they do, there is always a possibility of network interruption. Connections can be lost for hours, even days, and in these cases system is unavailable in local clinics. Considering all above, we have oriented to DDBMS in our solution of projecting and developing MIS.

With a choice of distributed instead centralized system, one problem is solved while another appears: database replication. Setting and administering database replication on seventy or more DBMSs demands many expert-hours. Resolving database replication conflicts demands good planning and even then, practice can lead us to unresolved replication conflicts that ends in a dead queue, waiting for a human interaction there. Besides, there is a standard proposed by the Republic of Serbia which says that any MIS solution must ensure the technological platform for the solution to be applicable independent of the database platform [2]. All world known replication solutions are platform-dependant, and when it comes to possible changing of the platform in some nodes, the world known problem of setting replication on different platforms is born. Mentioned reasons forced us to model and develop one platform-independent-information system dependant database replication solution which can be used in any information system that is based on DDBMS.

2 Facility Organization Structure and Server's Structure

Serbian Public Health Care System (SPCS) consists of many units: on the top of the structure there is a Republican Institute for Health Insurance (RZZO). Under RZZO there are region subsidiaries (Branch Belgrade, Branch Novi Sad, Branch Nis, etc.) and they further consists of health centers usually placed in cities. Every health facility is responsible for many local-branches which are actually ambulance stations (surgeries or consulting rooms placed in smaller towns, villages, schools, etc) [3]. Number of these local stations that belong to one Health Center is usually between 30 and 40, even 80 in larger centers. Logical simplified presentation of described structure is one n-ry tree whose root is RZZO. Next level in the tree presents region subsidiaries, etc, as shown in Figure 1. This structure can be modelled in a database with n-ry tree, like an entity called *organization_unit*.

Considering the fact that SPCS functions in a way that child node is always responsible only to its parent node with business operations, direct communication between un-neighbour nodes is not necessary. DBMS server's structure does not necessary concur with facility's organization structure. One DBMS can be responsible for data storing and data processing of one sub-tree. The existence of as less as possible DBMS servers is considered in [4]. DBMS server's structure that corresponds to one healthcare facility, together with necessary dataflow.

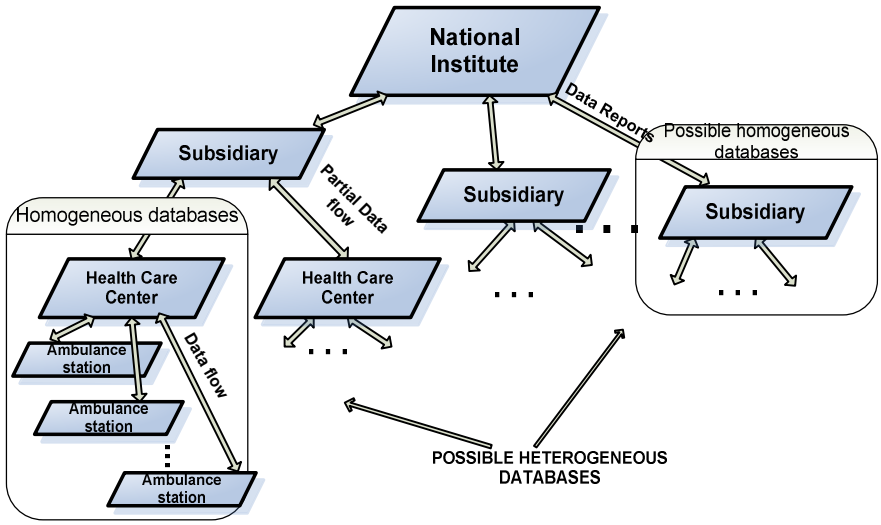

Fig. 1. Public health care structure in Serbia

3 Information System Dependant Database Replication

The main idea in projecting database platform independent replication was to plan replication and solve replication conflicts in advance, in the MIS planning phase. That was entirely possible because of the similarity of structures presented in the previous chapter. To be able to shift replication dependence from database platform to information system, we had to make MIS aware of the replication, which is not the case in platform-dependant replications.

3.1 Modelling of Database Replication Part

Managing replication by MIS requested database modelling of the dependences between two structures. So we introduced the entity called *server* with the same named table in a database. Table *server* stores all necessary information about every single DBMS in the system. Basic table fields are: *server_id* (PK), *parrent_server_id* (the ID of the server parent to this one in n-ry tree), *dns_name* (machine name in DNS), *database* (catalogue name), *ip_address* (server's TCP/IP address), *offset_from* (for primary keys, will be explained later), *offset_to* (the same as previous), *orgj_id* (FK, relation to organization unit for which the server is responsible), *flag_active* (flag-field, true if server is active in the system. If the server is turned off, information is kept), *flag_me* (flag-field, true only in one item in table, when database is on that specific server), *comment* (any useful comment). As it is obvious, all information needed for string connection to the database is contained in the *server* table for every

server in the system. There are two more fields of this table that will be mentioned later in this paper.

Let us look now at the part of data transfer from server to server. Instead of marking records for transfer in the database, we have decided to log performed statements. Application data layer executes statements and logs them at the same time, inside of one database transaction. The dependence of replication on IS is reflected in the fact above – system logs every insert, update and delete statement performed on DBMS for the replication purposes. There is a table called *rpush_log* with original fields: *rpush_log_id*, *upit* (statement, as string with max length), *imenik_id* (relation to the user responsible for statement executing), *trenutak* (timestamp with time zone of statement execution against the data source), *to_server_id* (what is the destination server in n-ry tree to whom information is intended to be sent, default=0 which means that every server in the tree need this information), *from_server_id* (who is the server in the tree that originally sent the information, default=1 – central server), *tabela* (the name of the table that logged insert, update or delete statement was executed on).

Data is replicated in two directions. The other two tables responsible for replication are *rpush* and *rpull* table. The first one stores every moment server sends data to any other server in the distributed DBMS, and the relation to that server. Fields are: *rpush*_id, *trenutak* (timestamp with time zone), and *server_id*. The other one stores record when replication is done in the other direction, relation to the server from the other replication side, and information about user who initiated replication. Fields: *rpull_id*, *trenutak* (timestamp with time zone), *server_id*, and *user_id*.

And finally, the fifth table responsible for this replication is the table that stores information about tables (articles) involved in replication, together with relation to server (*server_id*), because sets of articles replicated to different servers can usually be different. It is called *server_table*. Fields are: *server_table_id*, *server_id, tabela* (the name of database table that is checked for replication to the server *server_id*).

3.2 Logging

Logging of executed statements is the main part of information system dependence. Data layer of the IS must be aware of the DBMS server structure, which is provided with the *server* table in every DBMS. The dependency between facility structure and DBMS structure is provided with relation of the table *server* with the table *organization_unit*. There are cases when IS data layer logs executed statement to the *rpush_log* table with *to_server_id*=0, which means that this statement must be executed against every database in servers tree (it is replicated everywhere, in another words). This is usually the case when it comes to information of general importance for the whole system (medical catalogues, insurer's data, etc). But there are cases when it is not necessary to burden the entire system, as information on drugs issued to pharmacies, or entrance or exit of medical materials from the warehouses. Then IS logs statement with *to_server_id* field set to the ID value of the exact server to whom information is intended for. That statement will flow through server's tree until it comes to the destination server, and then it will be executed. This is possible because of the existence of special function we implemented in the data layer, which is called

every time when system executes insert, update or delete statement against the data source. This function determines if the statement need to be logged for execution against every database in the tree, or a specific database (server), and then it inserts the record in *rpush_log* table with all needed information about table against statement was executed, user who executed it, and the exact moment (it relays only on server's time setting) when it was executed. This is all done in one transaction, so it is not possible statement to be executed against data source, and not to be logged in *rpush_log* table. Details about this replication-algorithm are beyond the scope of this paper, because it is simply not possible to describe it in a few pages. However, we will try to give a basic overview of the functioning in the next subchapter.

3.3 Replication Agents and Replication Functioning

There are two program agents that are responsible for our replication, client and server agent. One is called *rpull* and the other *rpush* agent. They function independently from MIS. *Rpush* agent is replication server agent and is installed in every sub-tree where there is a DBMS. It works as a service on any machine that has network approach to the DBMS, and it never stops. It is always ready to service the replication on its side, to take or forward data. Users do not have any access to this agent. It is important to note that this agent supports multithreading, so it can serve many replication requests at the same time, as is shown in Figure 2. The other important notice is that database for replication is not determined in advance for server agent.

Rpull agent is replication client, installed to be available to users, or to the MIS. This is the agent that initiates replication when it is necessary to transfer some data. Users with a specific privilege can start this program-agent and initiate replication between their server, and the server they can choose from the list. They can choose whether they want to send or to receive data (or both). Of course, this can be automated, and scheduler can start replication instead of users (every night or every hour or similar). For example, this is convenient for pharmacies that expect a drug delivery, and before the shipment arrives, the pharmacist will take over data about it, and update the status of drugs in the warehouse automatically, loading the system only when there is a need for it. If the replication is set to work during the night, and it comes to the case when a patient come the same day for examination in two different clinics of the facility-tree, the replication agent will be started by the user, and in a few minutes, data will be there.

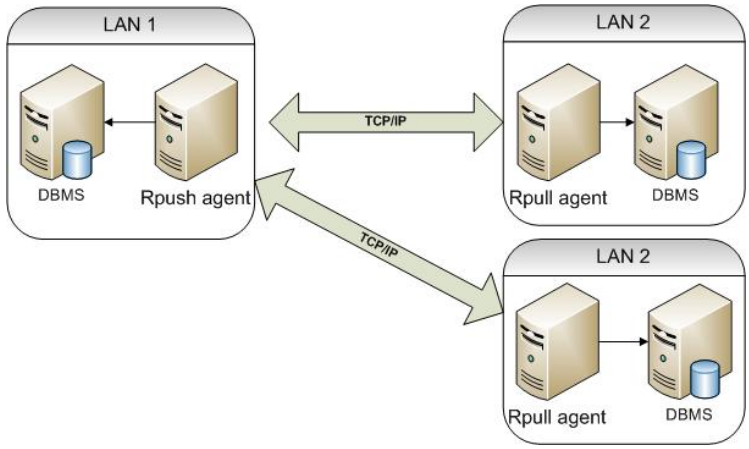

Fig. 2. Communication between replication client and server agents. One server agent can serve many replications simultaneously

Rpull agent connects to the database and reads the information from the *server* table about server to whom it will replicate database with, (beside fields in the table mentioned above, there is a field containing port number for communication with specific server, and the IP address of the machine on which *Rpush* agent resides on from that side of the replication). Then it sends a demand to the specific *Rpush* agent through windows socket, on the port that *Rpush* agent listens, representing its self (from what server in a server tree it comes and what database on that DBMS (this replication is wide solution and it can be applied to many servers and more than one database on every server). *Rpush* agent (if everything is fine with network connections) accepts this call, and returns information about it. The hand-shake between replication agents is performed. *Rpull* agent then sends pocket with database name and server IP to whom it wants to be replicated with. *Rpush* agent creates connection string based on the received information, and connects to the database for the replication.

The next step in the replication is checking *rpull* table from the *Rpull agent* side, and *rpush* table from the *Rpush agent* side. Agents exchange information about last moment of replication between these two databases, and if it is not the same, the latter timestamp is inserted on the appropriate side, and the replication can begin. Basic steps of this communication are presented in Figure 3.

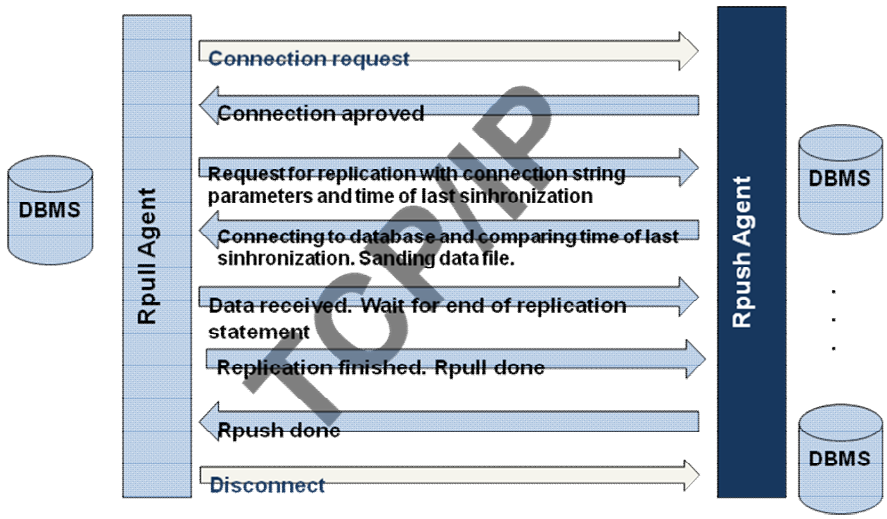

Fig. 3. Replication process workflow

Sending data from Rpull agent side: SELECT statement over the *rpush_log* table is done. Briefly, all queries with a timestamp later then the moment of the last executed replication from this side to that server are packed in a text file, and sent to the *Rpush* agent. It initiates the execution of statements against the data source on its side, with checking what statement needs to be logged for further execution on another server, according to the predefined algorithm of decision. Agent starts one large transaction that will last until all statements that need to be executed are executed, some of them are logged in the *rpush_log* table of that server with all necessary information, and the information about that synchronization is recorded in *rpush* table of the replicated database. Also, there can be a case that the statement is logged for further replication without being executed against this data source (if the algorithm decides so). The transaction can be committed after all mentioned steps are performed well, or rolled back if opposite. If committed, agent sends the acknowledgement to *Rpull* agent (with a message that *Rpush* has done its job). *Rpull* accepts that confirmation and records information in its *rpull* table for that specific server.

Receiving data from Rpull agent side is not much different from sending. There is a hand-shake between two agents, only in this case *Rpush* agent sends file with queries. The procedure is the same. In both cases text file with queries is packed, in an effort to reduce network traffic as much as possible.

4 Advantages and Disadvantages

As in any other new approach, there are a lot of disadvantages when we compare it to the well known world replication solutions. Main disadvantages will be described in following section.

4.1 Main Disadvantages

The main disadvantage of the presented solution is that it does not support replicating of Binary Large Objects (BLOB), at least not yet. Resolving of this problem requires more researching and designing, and it belongs to our future research.

The other disadvantage is that if you want to replicate schema changes between databases, you need to log DDL statements by yourself in *rpush_log* table, or perform schema changes through specially designed interface that will log everything in the database. But that is the price we have to pay, if we want to make our replication independent from database platform. Anyhow, schema changes can be replicated, and the fact that only those that we log are going to be replicated is convenient in some cases in practice.

And the last, but not the least, is the lack of automated way of including table in replication later. The snapshot of any table that was not checked for the replication in the very beginning of it must be done manually by database administrators. Considering the fact that in Medical Information Systems there can be a lot of new demands by the state, this disadvantage can represent a lot of work for administrators.

4.2 Advantages

Compared to world known replications (SQL Server replication, Oracle replication for example), our replication is much easier to be implemented and controlled by administrators. Databases in advance have replication tables included, and MIS takes care of logging statements its self. Setting replication agents to both sides of replication involves only installing programs and training users how to initiate data synchronization from their side, which can be done with two mouse clicks and one server-choosing.

The main advantage of this approach is the fact that there cannot be replication conflicts, because they are solved in advance. Earlier mentioned table *server* fields *offset_from* and *offset_to* takes care of Primary Key Violation replication conflicts. Every server in the server tree has predefined offset of integer IDs (primary keys) for tables: *offset_from* is their minimum value and *offset_to* maximum. This offsets are carefully planned according to statistics on number of records per year for servers multiplied with number of years for keeping data before archiving [5]. For central server usually the biggest offset is planned (for example, in MIS in Health Center Nis, Serbia, it is 100 million). Update or delete conflicts are impossible because MIS determines what action can be done on what server. Considering the fact that replication conflicts can be a nightmare to database administrators, the advantage of the lack of them is clear to all who had ever have to deal with them in their working experience.

And finally, this approach can be performed in any kind of networks, including a low-band/low-speed like modem connections are. The communication of two replication agents is reduced to minimum, and if it comes to the network interruption after the file with statements is transferred, replication will not be interrupted. It will be performed and the last synchronization will be recorded, but only from one side of the replication. However, next time when two agents hand-shake, this information

will be checked and recorded in order to make a pair of records corresponding to each other on both sides of the replication. This is especially convenient for mobile users, like nurses and doctors on home visits. They can have DBMS on laptop and replicate data when they are back on their clinics. They can even replicate when they are at patients home, connecting through modem connection, when they have emergency situations.

5 Conclusions and Remarks

Database platform we have started to work with on our project was MS SQL Server 2005. The first phase of our project was developing modern MIS and implementing it in Health Center Nis, Serbia, which represents a facility with 80 remote smaller ambulates. Since we have decided to relay on DDBMS, considering all the facts given above, database replication was one of the key facts of our success.

MS SQL Server Standard Edition supports all types of replication (snapshot, transactional, merge, merge bidirectional…), but MS SQL Server Express, which is non-commercial and convenient for smaller remote clinics, has very limited scope of database replication [6]. Setting replication between all these servers has turned to be very hard and expensive, in a meaning of specialist hours. There is a possibility to use the combination of platform-dependant replication both for objects that are replicated entirely over the whole n-ry tree (like many medical catalogues, given by the state, not allowed to insert in remote clinics, so no replication conflicts could happen in these cases), and our platform-independent simple replication for those whose setting is complicated (like shipments of medical materials and drugs, patient examination data, etc). The other item is related to data whose existence is not necessary in all tree nodes, but only to specific ones, determined in advance. This means much less data waste for wayside servers, quick and simple data synchronization for users, and reliable in low-band/low-speed network connections like some connections in our villages still are.

This kind of replication can easily be applied to any IS beside MIS we have developed, by taking several actions. First: 5 tables need to be added to the database, like it was mentioned in chapter 3 of this paper. Organization structure of the IS company must be well designed, as an n-ry tree (which is the case in most companies). Data layer need to be modified by implementing a bottom-level function that will determine logging, and log executed queries to the database. And finally, replication agents need to be set to the nodes for replication.

Thanking Note

This paper is supported by the Ministry of Science and Technological Development of the Republic of Serbia (Project Nr TR13015).

References

1. Wager, K.A., Wickham Lee, F., Glaser, J.P., "Managing Healthcare Information Systems - A Practical Approach for Health Care Executives", Jossey-Bass; 1 edition (May 5, 2005), ISBN-10:078797468
2. Article 3, paragraph 3, Uredba o Programu rada, razvoja I organizacija integrisanog zdravstvenog informacionog sistema "e-Health", Sluzbeni glasnik RS, br. 55/09
3. Stankovic, N.T., Jankovic, S.D., Pesic, N.S.: Public Health Care Distributed DBMS with Resolving Database Replication Conflicts in the Health Care Information System Project Early Phase, 9th International Conference on Telecomunications in Modern Satellite, Cable and Broadcasting Services - TELSIKS, IEEE Catalogue Number: CFP09488-PRT, Vol. 2, pp. 487-490, Nis, Serbia (2009)
4. Stankovic, N.T., Stankovic, R.M., Jankovic, S.D.: Platform Independent-Information System Dependent Database Replication, 43th International Scientific Conference On Information, Communication and Energy Systems and Technologies - ICEST, pp. 415-418, Nis (2008)
5. Pesic, N.S., Stankovic, N.T., Jankovic, S.D.: Benefits of Using OLAP Versus RDBMS for Data Analyses in Health Care Information Systems, ELECTRONICS, Vol. 13, Number 2, pp. 56-60, Banja Luka, Bosnia and Herzegovina, (2009)
6. http://msdn.microsoft.com/en-us/library/ms165686.aspx