A set of tools for building PostgreSQL distributed databases in biomedical environment.

M. Cavalleri, R. Prudentino, U. Pozzoli, G. Reni

IRCCS "E. Medea", Bosisio Parini (LC), Italy. E-mail: greni@bp.lnf.it

***Abstract* – PostgreSQL is an advanced Object-Relational DBMS supporting SQL constructs and accessible with standard protocols. Its object-oriented features qualify it for managing biomedical data and it is freely available to the community via the open-source philosophy. Unfortunately the current available version does not support any database distribution feature.**

The aim of the present work was to develop specific procedures in order to extend original potentials of PostgreSQL ORDBMS and make it able to manage distributed databases by means of asynchronous replication. Particular attention was devoted to conflict resolution rules, including several procedures with different degrees of restrictiveness and even giving the end user the possibility to write user-defined conflict resolution procedures.

The replication system was tested at the IRCCS "E. Medea", an Italian Scientific and clinical research Institute spread over 5 sites in different geographical locations. The testing system was developed with the purpose to make clinical data collected by Epileptology Units in each site available to all branches, without having to care for their physical distribution.

The proposed replica procedures manage only traditional (non-binary) data types because of the very different storage model of large objects. Since large objects are very common for treatment and storage of biomedical data, we are currently working on an improved version of the replica engine that allows large objects to be replicated.

***Key words* – distributed database, ORDBMS, biomedical data**

I. INTRODUCTION

In the last years data communication evolution has rapidly and substantially changed the way information is managed. The growth of low price connectivity and the improvement of the data storage technology led people to ask for even more information to be always accessible and from everywhere.

New technologies were developed to share data scattered on the net (eg. MIDAS, CORBA), but sometimes data aggregation is also needed. For example, if we want to get a unique set of data from databases scattered on different sites, these new technologies require a data distribution mechanism.

PostgreSQL [1,2] is an advanced Object-Relational DBMS supporting SQL constructs and accessible with standard protocols. Its object-oriented features qualify it for managing biomedical data [3] more than other RDBMS and it is freely available to the community via the open-source philosophy. Unfortunately, the current version does not support any database distribution feature.

Data Replication is a process that allows to build a distributed database through the management of multiple copies of data, caching one copy on each site [4] (Fig. 1). In particular, *synchronous replication* (also called real-time data replication) conveys information in real time to all of the involved sites. On the contrary, *asynchronous replication* (store and forward replication) stores operations performed on a database in a local queue for later distribution by a *database synchronization* process.

Synchronous replication technology ensures the highest level of data integrity but requires a permanent availability of servers and transmission bandwidth. On the other hand, asynchronous replication provides more flexibility than synchronous replication as a database synchronization time interval can be defined which can vary from minutes to months and, moreover, a single site could work even if a remote server is unreachable or down. In addition, data operations are performed more quickly and network traffic is more compact. However, a complex replication planning is required in the case of asynchronous replication in order to detect and correct data conflicts due to concurrent modifications occurring at different sites between two database synchronization events.

The aim of the present work was to develop specific procedures in order to extend original potentials of PostgreSQL ORDBMS and make it able to manage distributed databases by means of asynchronous

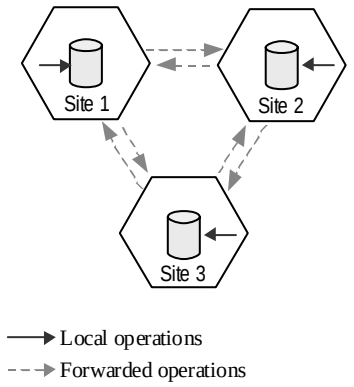

Figure 1: Working scheme of data replication. In this example three sites are considered.

replication. Because of the nature of biomedical data, particular attention was devoted to conflict resolution rules, including several procedures with different degrees of restrictiveness and even giving the end user the possibility to write user-defined conflict resolution procedures.
In the following sections the main issues regarding the development and abilities of the new replication procedures are addressed. First, we describe the mechanism built to uniquely identify a record into the distributed database. Second, we analyze the logger that allows forward and roll back transactions. Third, we examine the module that joins the transactions coming from remote sites preserving data integrity in line with specified conflict resolution rules. Finally, we provide some implementation details for each of the previous steps.

II. IDENTIFICATION OF DISTRIBUTED DATA.

Any table in a distributed database can be represented in the following form:

$$T_{a,i} = P_{a,i} \cup \bigcup_{j=1\ldots N,\, j\neq i} R_{a,j,i} \qquad (1)$$

where a identifies a generic table of the database
i identifies the site (site-id),
$T_{a,i}$ is the entire content of table a on site i,
$P_{a,i}$ is the information entered in table a on site i,
N is the number of sites,
$R_{a,j,i}$ is the replicated partition inside table a coming from site j cached into site i.

Each site is identified with a *site-id i*. $P_{a,i}$ and $R_{a,j,i}$ are partitions of table $T_{a,i}$. When an insertion occurs in table a on local site i the information is stored in the local partition $P_{a,i}$ and later forwarded to every replicated partition $R_{a,i,j}$ of each remote site j. A mechanism for the unique identification of a record either in local ($P_{a,i}$) or remote ($R_{a,j,i}$) partition is required.
Since the *record-id* number is site-dependent information that is unique for each record stored in a local database but is not unique in a distributed database, for each record we keep track of the insertion site-id (master site) together with the record id number related to the master site. Both local and replicated copies of the record contain this information that is combined with a map of the distributed tables to ensure unique identification of each record. By this way we can always know the origin (master site) of a record and we can represent each table as logically splitted into partitions (1).
Taking advantage of the method we can freely adopt one among the most common replication data ownership models, and eventually apply the selected model to a reduced set of tables. In particular, *Workload Partitioning* data ownership model [5] is implemented by read-only access to $R_{a,j,i}$ and read-write access to $P_{a,i}$.(Fig. 2). The *Master/Slave* [6] data ownership model is obtained by read-write access to $R_{a,j,i}$ and $P_{a,i}$ when the site i is Master for table a, and by read-only access to $R_{a,j,i}$ and $P_{a,i}$ when the site i is Slave. *Update Anywhere* [7] data ownership model is obtained by read-write access to $P_{a,i}$ and $R_{a,j,i}$.

III. LOGGER FOR FORWARD AND ROLL BACK TRANSACTIONS

Let us consider $T_{a,i}(t_0)$ a generic table of the database at a given initial time t_0. At any following time t, the same table $T_{a,i}(t)$ will be a function of the initial status $T_{a,i}(t_0)$ and of the sequence of operations applied to it in the interval $[t_0 - t]$. We store each operation applied to table $T_{a,i}$ as a record in table $C_{a,i}$ thus obtaining

$$T_{a,i}(t) = f[T_{a,i}(t_0), C_{a,i}(t)] \qquad (2)$$

where f() is the function able to apply the sequence of operations $C_{a,i}(t)$ to table $T_{a,i}(t_0)$.
It is possible to define a roll back function $f^{-1}()$ too, which is able to apply the sequence of operations $C_{a,i}(t)$ to table $T_{a,i}(t)$ in a reverse way, obtaining

$$T_{a,i}(t_0) = f^{-1}[T_{a,i}(t), C_{a,i}(t)] \qquad (3)$$

As table T_a is replicated at several sites, we have a different $C_{a,i}$ at each site as well as a different $T_{a,i}(t)$. The actual and aligned replicated table $T_a(t)$ containing every operation performed both in local and remote

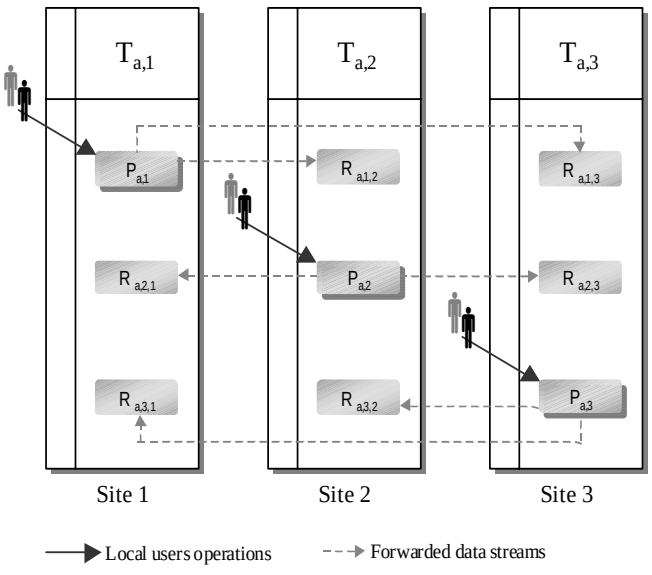

Figure 2: Workload partition data ownership model obtained allowing users to modify local partitions only.

sites will be obtained considering changes $C_{a,j}(t)$ occurred in each site *j* and applying them to the table $T_{a,i}(t_0)$ in a generic site *i*:

$$T_a(t) = f[T_{a,i}(t_0), \bigcup_{j=1...N} C_{a,i}(t)] \quad (4)$$

IV. Replication Conflicts Management

An asynchronous replication system allowing data updates in each site (update anywhere data ownership model) may cause some replication conflicts between various $C_{a,j}(t)$.

A uniqueness conflict occurs when different sites try to insert different records with the same unique key.

An update conflict occurs when different transactions on different sites refer to the same record and the requested operations are inconsistent.

The number of conflicts increases with the number of sites N and the time interval between two database synchronization events.

We implemented a conflict resolution module that acts as a filter on $\cup_{j=1..N} C_{a,j}(t)$ forwarding only operations that can be completed according to the resolution algorithms defined for each table. This particular conflict management scheme allows serial application of resolution algorithms, making it possible, for example, to check some sort of data consistency based on medical group privileges priority, after preliminary conflict resolution.

A set of record-based conflict resolution algorithms was also developed in order to reduce the number of rejected operations due to update conflicts. These algorithms were based on the following two observations: it is always possible to identify an intermediate period without any update conflict $[t_0,t_c]$, being $t_0 \le t_c \le t$; it is usually possible to modify the sequence of operations on different records preserving data integrity and shifting t_c as close as possible to t.

V. Implementation details

In our system replica actions are controlled by a specific database administrator username (*replicator*). An automatic mechanism checks this username in order to prevent unwanted operations from being recorded on $C_{a,i}(t)$ during replica process.

A complete map of all the tables making up the database and the rules for their replication is stored in a set of tables, called Replica Scheme Tables (RST), specifically designed to provide flexible definition of distribution of the tables over the sites. For example, tables can be independently replicated in different subsets of the distributed database.

Modifications to data stored in RST fire triggers that qualify or retreat a generic table from being replicated, dynamically creating or destroying triggers and auxiliary tables for logging applied operations, greatly reducing the DBA tasks. In particular, each replicated table $T_{a,i}$ has its own auxiliary table $C_{a,i}$ in which sequences of applied operations are stored.

The object-oriented features of *PostgreSQL* make it possible to inherit each auxiliary table from its original one, thus keeping the same record structure. Additional information such as timestamp, type of operation (insert, update, delete) and user name are recorded too, to allow construction of advanced security algorithms for data replication, visibility and treatment.

To accomplish replicated database synchronization a massive planning and coordination of various steps is required because of several distributed and parallel processes running at different sites.

The DBA plans the scheduling of the replication process that can be started from any site at any time. The site where the replication process starts is defined as replica-master site, which will coordinate each step during that particular replication. A replica daemon running on each site answers requests coming from the replica-master site.

During database synchronization (Fig. 3), the generic site *i* creates a set of data $C_i(t)$ that contains operations applied inside the site to any replicated table since the last synchronization event. This set of data is transmitted to every destination site and filtered by the conflict resolution module.

The conflict resolution module contains several predefined algorithms that manage common conflict situations, whose probability is related to the number of sites, the synchronization interval, the type of application running on the database, the replication data ownership model chosen. Custom procedures that improve flexibility of the replica process can be written too and added to the module.

The DBA chooses the conflict resolution algorithm to be used with each table and stores this information in the RST. The DBA can also specify if operations rolled back by conflict resolution algorithms have to be signaled to the end user with e-mail messages.

Special attention is devoted to data propagation planning and data are compressed during transmission in order to minimize bandwidth requirements and speed up the process. Data encryption is also available to ensure data privacy.

Trigger procedures were written in the procedural language PL/Tcl. The daemon was written in Tcl [8] linked with PostgreSQL connectivity API and using Tcl/DP libraries for the communication layer over TCP/IP sockets.

The replication system was developed on a PC with CPU Intel Pentium III running Linux Red Hat 6.0 with kernel 2.2.5. The tests were performed on SUN SparcStation 20 and SUN Ultra 1 running Unix Solaris 2.x, and on PCs with CPU Intel Pentium II and III running Linux.

VI. Case study

The replication system was developed and tested at IRCCS “E. Medea”. The IRCCS “E. Medea” is an

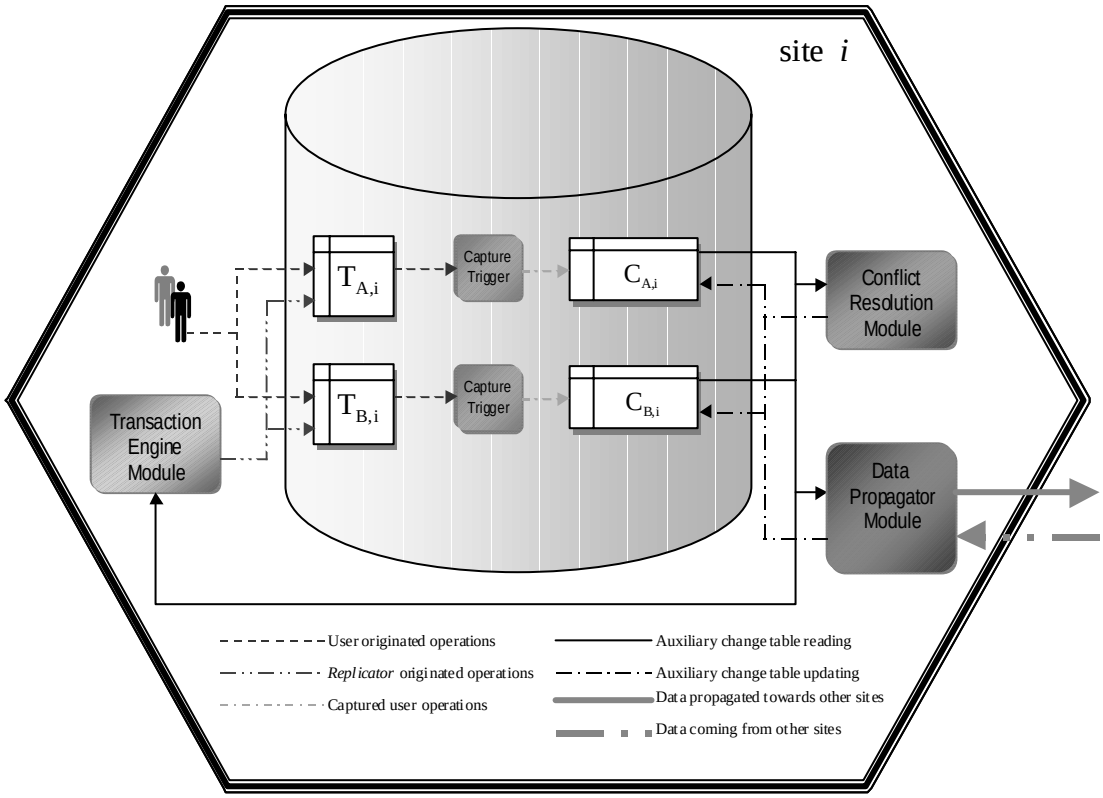

Figure 3: Replica process schema. User operations on a shared table are captured and stored into auxiliary tables. During database synchronization phase local streams are propagated over involved sites and locally filtered by Conflict Resolution Module; Transaction Engine Module applies accepted operations and roll back rejected ones.

Italian Scientific and clinical research Institute spread over 5 sites in different geographical locations, connected through a private network.

The testing system was developed with the purpose to make clinical data collected by Epileptology Units in each site available to all branches, without having to care for their physical distribution.

A Workload Partitioning replication data ownership model was chosen because the Health and Management Direction asked to put read-only limitations to remote departments.

The application was tested successfully.

VII. Conclusion

In the present study we developed a set of tools to extend capabilities of PostgreSQL in order to make it able to manage database replication in different sites.

The new set of tools implements asynchronous data replication over several sites connected by a wide area network. Having more than one copy of the same database increases availability of the system, ensuring data access or data backup even if some servers are down or not reachable. Better performances are also obtained working on local data. One of the drawbacks is that local operations performed in different sites between two database synchronization events lead to temporary data inconsistency over the whole system.

The system was tested in a clinical research institute and gave encouraging results.

The proposed replica procedures manage only traditional (non-binary) data types because of the very different storage model of large objects. Since large objects are very common for treatment and storage of biomedical data, we are currently working on an improved version of the replica engine that allows large objects to be replicated.

VIII. Acknowledgments

Replication procedures were built using many free software tools. The authors wish to thank PostgreSQL development team for making available the source code which was the starting point of this work. We would also like to thank developers of the Tcl language and developers of the extensions reported in the previous sections.

IX. References

[1] Stonebraker, M., **The design of the POSTGRES storage system,** *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, Sept. 1987; 289-300

[2] Stonebraker M., Kemnitz G., **The POSTGRES next-generation database management system,** *Communications of the ACM*, Oct. 1991, vol.34, (no.10):78-92.

[3] Diallo B., Travere J.-M., Mazoyer B., **A Review of Database Management Systems Suitable for Neuroimaging,** *Methods of information in medicine*, F. K. Shattauer, Feb. 1999

[4] Chen S. W., Pu C., **A structural classification of integrated replica control mechanism,** *Technical Report CUCS-006-92*, Columbia Univ., New York, NY, 1992.

[5] **Enterprise Replication: A high-performance solution for distributing and sharing informations**, INFORMIX Software Inc.
[6] **Comparing Replication Technologies**, PEERDIRECT Inc.
[7] **Oracle8™ Server Concepts: Database Replication**, Oracle Corporation.
[8] Ousterhout J., **Tcl and the Tk Toolkit**, *Addison-Wesley*, 1994.