PGSYNC: A Multiple-Reader/Single-Writer Table Replication Tool For High Loaded Distributed Relational SQL Databases

Eric Sadit Téllez 1
sadit@lsc.fie.umich.mx

José Ortiz 1
job@lsc.fie.umich.mx

Mario Graff 1
mgraffg@lsc.fie.umich.mx

Abstract

This paper presents a new tool to preserve coherency of data in a set of Relational Databases: PGSYNC. It performs replication over a set of single tables with a simply Multiple-Reader/Single-Writer scheme.

PGSYNC claims to be closely to real-time, simple, low latent, secure, multi-platform, and open source solution to the synchronization database problem. It’s specially designed to process that generates high volumes of SQL queries.

Additionally we present a real life heavy loaded system using PGSYNC as incremental backup manager and work load distributor.

1. Introduction

The high amount of information in government’s databases and the number of simultaneous queries that the database receives are critical problems in the design of new services and reports, and this problem increases when the server reaches its limit capacity making impossible to incorporate a new service into the server.

The naive solution to this problem is to migrate the system to a newer server with better characteristics, but this approach would be a temporal solution. Another solution is to distribute the load over a cluster of computers and balance the load over the cluster. The later solution has its own problems like: data coherency in clustered computers.

This paper investigates a solution to this problem that is a fast, low latent, secure, multi-platform, open source SQL Table synchronizer. Our prototype is named PostgreSQL Synchronizer (PGSYNC). It is capable to work in heterogeneous hardware and software under the Postgresql RDBMS.

PGSYNC is based on a Single Writer and Multiple Reader (SWMR) scheme. This behaviour means: At a given time only one master peer in the cluster is allowed to modify an specific table and remaining peers (slaves) are synchronized from it or from other synchronized slaves. It is important to mention that this restriction is a per Table constraint.

Roadmap. In Section 1 a brief introduction of the problem is given. Section 2 reports the State of the Art in the area. The Section 3 shows the architecture of PGSYNC. In Section 4 a real system using PGSYNC is presented and a set of benefits exposed. Finally, Section 5 shows conclusions and future work of PGSYNC.

2. Related Work

In the case of open source there are few replication software like pgpool [7], PGCluster [2], and Slony-I [5]. For example pgpool can be used as a replication software but its main application is being a pool server. In other words, pgpool is a load balance system. It doesn’t perform well as a replication tool.

PGCluster is the synchronous replication system of the multi-master composition for PostgreSQL. It is really synchronous, so there is no delay in synchronization between computers, but this implies a high latency on the communication process.

Slony-I is a single master to multiple slaves replication system supporting cascade (one client can feed another client). Slony-I is a very good replication system, but it’s complex and difficult to port to different systems.

3. Architecture

PGSYNC is composed of two main layers: The Database Layer, and the User Layer (see Figure 1). PGSYNC is written in Python [6], PLPython [3], and SQL.

The Database Layer (DBL) manages the replication, based on table’s triggers. DBL stores every modification on a table named replica. Table 1 shows the structure of replica.
The User Layer (UL) is controlled by the PGSYNC daemon, and it is present in both Master and Slave peers. The UL manages the replication across the network. Connection between peers are performed using SPyRO [9] as middleware.

### 3.1 Configuration of PGSYNC

PGSYNC uses a simple peer configuration of masters and slaves, as shown the Figure 2. In a replicated table only one master is allowed, but many slaves (replicas) can be used. This constraint yields to a SWMR scheme.

There are two basic operation modes of slaves:

- **Incremental Backups.** In order to maintain the security (integrity and multiple localizations of the data), PGSYNC supports incremental backups. This mode saves every performed changes over *watched* tables in master peers. Then slave peers can regenerate tables or just save the changes (this one is specially useful to stop errors propagation).

- **Load Balancing.** After regeneration of tables, the new database could be used to distribute the load of the RDBMS, with just a little effort and with a configurable delay to become synchronous.

PGSYNC supports cascade: one slave can feed other slaves.

### Table 1. Description of the Table replica

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Integer Primary Key</td>
<td>DBM Dependent</td>
</tr>
<tr>
<td>EventTime</td>
<td>Timestamp</td>
<td>DBM Dependent</td>
</tr>
<tr>
<td>EventName</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>Username</td>
<td>Character</td>
<td>12</td>
</tr>
<tr>
<td>OriginHost</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>TableName</td>
<td>Character</td>
<td>64</td>
</tr>
<tr>
<td>OldData</td>
<td>Text</td>
<td>No Limits</td>
</tr>
<tr>
<td>NewData</td>
<td>Text</td>
<td>No Limits</td>
</tr>
</tbody>
</table>
3.2 Regeneration of Tables

The straightforward method to regenerate tables is generate SQL sentences based on the previous and actual data. PGSYNC provides a handler mechanism tunable at table level, to ensure the right management of the data. Each handler receives registers of the replica table and returns SQL sentences. General default handlers are provided for easy installation and execution of PGSYNC. In order to enhance performance (enable usage of specific indexes to eliminate unnecessary automatic generated constraints), every table must have its own set of handlers (i.e. Insertion Handler, Deletion Handler, and Update Handler).

3.3 Synchronization

The synchronization process is the core of PGSYNC and its performed using SPyRO [9] as middleware. PGSYNC uses the MARSHAL [6] serializer to enhance performance, but it can use any of the SPyRO serializers in order to increment the number of target platforms.

The synchronization is performed by chunks, and the chunk size is configurable. Optimal chunk size is application dependent, and must be chosen based on the desired network latency and synchronicity. The default chunk size is 40000 replica registers at one time. Figure 3 shows an experiment for the synchronization of 100000 modifications of a table. In the X-axis we show the chunk size, and in the Y-axis the time in seconds. The time doesn’t change to large values of chunk sizes (in the left side of the Figure, we can see a decreasing line, this is because for small values of chunk size, the synchronization process is very slow). We must note that the Figure is in log-log scale. Figure 3 shows the total time of synchronization, the cost of the communication (data’s transfer), and storage time of the replicated registers. The Figure shows that for a big chunk size, the latency is low and the storage of replicated elements takes most of the time at the replication process.

The synchronization is performed by request of slave peers, then a timed synchronization is used. This asynchrony delay can be tuned to fill the requirements of the system. Ten minutes is the default asynchrony delay.

3.4 Concurrent Access

The concurrent access is handled using the built-in tools of PostgreSQL, and the timestamp marks of the table replica. In addition, an exclusive lock is used at the moment of the synchronization to ensure data consistence. It’s necessary to mention that write access to replica doesn’t lock the table, due to the PostgreSQL’s Multiversion Concurrent Control (MVCC [4]).

4. A Case of Study

PGSYNC is a tool primarily designed to be used as a backup tool and load balance of the database servers of Vehicles Taxes Collection (codename SARVE1) in the Mexican State of Michoacán.

SARVE uses a set of tables clustered in schemas. Two main schemas are producción and extranjeros. Producción is focused on the traditional vehicle taxes (proprietary changes, registration cards, buy and sales taxes, refrendo, tenencia, etc). There are 112 replicated tables in the producción schema. Extranjeros is used to store information about legalized foreigner vehicles (vehicle information, legalization documents, frontier payments, etc).

Both schemas has many tables with critical data, SARVE is not allowed to fail and we must ensure the fast response to services. There are three main servers running Debian GNU/Linux [1]: REV, Ingestatales, and ILO. REV is the most important server, and its the main server of producción schema. Ingestatales is the main server of extranjeros schema (extranjeros is a not a heavy loaded system, but the information is critical). ILO is the backup server.

SARVE must serve to approximately 104 offices (each office has from two to twelve system users), and approximately 250 active users.

Typically, the configuration of PGSYNC in SARVE for producción is the shown in Figure 4. In this schema, REV is working as primary server, Ingestatales is the backup and load balance server, and ILO is just a simple up-to-date backup server.

1Sistema de Administración del Registro Vehicular del Estado
Ingestatales serves online-payments and information about debts of the taxes\(^2\), and needs an up-to-date tables (A big amount of requests are performed from online services). Furthermore in case of failure in REV, Ingestatales becomes the master. When REV returns, it serves as backup server. ILO is maintained as backup without generation of registers.

It is necessary to notice that ILO’s services are used to maintain integrity of the data against human errors (it preserve data against internal or external attacks or human’s mistakes) and forensic analysis.

SARVE solves more than 80000 requests per hour, in the Figure 5 a histogram of a week per hour is shown. The load is divided into REV and ingestatales, without the ingestatales replica the server REV could slows down performance in peak hours. Before PGSYNC, SARVE has frequent failures due to heavy load. Figure 6 shows the ingestatales load histogram per week. To create a full picture of the problem we need to say that every request could have several requests to the database, there are not requests without database access. The histograms are made based on the Apache (See [8]) access log, filtering the SARVE web application accesses.

Figure 7 shows the PGSYNC configuration to schema extranjeros. Ingestatales is the main server and REV is used as a backup server. ILO is used as a backup without regeneration of tables. In case of server failure of ingestatales, REV is not allowed to take its functionality, because REV is a heavy loaded server and the inner offices have preference over the online services. Unfortunately, ILO is not allowed to take the control because of the policies management of ILO.

5. Conclusions and Future Work

In prior sections we expose PGSYNC a new tool to replicate, and incrementally backup data of SQL tables for PostgreSQL. This new replication system was successfully implemented and executed in a real heavy loaded, critical service, such as Vehicle Taxes Collection “Ingresos Estatales del Gobierno del Estado de Michoacán” in México, one of

---

\(^2\)http://ingresos.michoacan.gob.mx
the most important kind of taxes in México.

PGSYNC has been working for an half year as a secure and flexible tool to ensure that the services will be provided.

PGSYNC is an open source software released under the General Public License and can be obtained from http://lsc.fie.umich.mx/~sadit/pgsync. It is multi-platform and is highly configurable to fill a wide range of necessities. It supports postgresql schemas. Additionally, PGSYNC does not need a tight integration with Postgresql neither needs a modified Postgresql version to work.

In contrast, PGSYNC does not support renaming of tables or schemas. In future versions we plans to support other RDBMS, like MySQL. Another challenge of PGSYNC is to find better representation of the replica registers, in order to optimize transfers and storage resources. Another desirable improve to PGSYNC is the propagation of structural changes in tables (e.g. addition, deletion and modification of columns). Finally, we need a better user and administrator interfaces to PGSYNC, currently only command-line and application programmer interfaces (APIs) are provided.

References

[9] E. S. Téllez, E. Chavez, and J. Contreras-Castillo. Spyro:
   Simple python remote objects. In Proceedings of the Fourth