

# A reliable architecture for Asterisk Cluster

**Michele Bianchi**

Association Open Source In-  
novation  
email: michele.bi-  
anchi@openinnovation.it

**Roberto Grasso**

Association Open Source In-  
novation  
email: roberto.grasso@ope-  
ninnovation.it

## **Abstract**

*IT technicians have to face reliability if they want to enter into telecoms industry. The telco world applies no single point of failure criteria for HW systems and network in order to guarantee up to five-nines service reliability. The IT industry is still far from such a level of reliability but anyway is willing to enter the relative rich telco market by surfing the success of Open Source PBX projects such as Asterisk, SER, SipX.*

*This paper aims at presenting a robust cluster architecture to assess reliability based on Open Source software and native IP phone features by avoiding complex and expensive common cluster solutions. This solution currently is up and running to serve all business VoIP traffic of one of the most used credit card in Italy.*

## **Keywords**

Open Source, Asterisk Cluster, Failover, Real time architecture.

## **INTRODUCTION**

PBX manufacturers are supplying uninterrupted and continuous services with the highest Quality of Service (QoS) and maximum availability provided by hardware redundancy (double power suppliers, disk mirroring, state solid HD, double telecom cards, ...). According to this approach, the main idea to meet reliability requirements is to use physically separate and redundant devices in order to reduce dramatically the probability of system failure. So

doing the objective to obtain high reliability is reached but at increasing the cost of the HW, that also need dedicated drivers and Operating System requirements.

On the other hand, customers, facing a severe economical crisis, are continuously looking to optimize their revenues and reduce their capital and operating expenditures to be able to survive in today's competitive arena.

Open Source software PBX such as Asterisk, SER, SipX are based on the idea to move the computational power from (expensive) DSP (Digital Signal Processing) to commercial CPU.

But so doing the reliability of the solution, that becomes totally an informatics application, struggles to reach two-nines while for decades the five-nines reliability of the PBX legacy solutions was the common standard. Moreover people are used to expect voice calls to continue even in the face of a disaster.

A possible solution to increase the availability of such application is to use the failover cluster pattern [1] that makes it possible to design a highly available application infrastructure tier that protects against severe loss of service due to the failure of a single server or the software that runs on it. A failover cluster is a set of servers (at least two) that are configured so that if one server (the primary) becomes un-

available, another server (the standby) automatically takes over for the failed server and continues processing. If the secondary is already running, the transaction between the primary and the secondary can occur seamlessly. In such a way we are talking about secondary in *hot standby*. Only the transaction currently managed by the primary will be lost: the secondary can start processing the first new incoming transaction.

An application using a failover cluster usually needs special tools to ensure that when a failure occurs, the failover process is transparent to the user and the application remains available. The first requirement is to assess that data are replicated between primary and secondary. This task may be accomplished by DataBase clustering. The second and more delicate point is how to determine that the active server no longer running correctly and to make the secondary standby server the active one. The systems usually use a heartbeat mechanisms to accomplish this.

There are different type of heartbeats [2]. In the push heartbeat scenario, the active server sends specified signals to the standby server at a well-defined interval. If the standby server does not receive a heartbeat signal over a certain time interval, it determines that the active server failed and takes the active role. For *pull heartbeats*, the standby server sends a request to the active server. If the active server does not respond, the standby server repeats the request a specific number of times before taking over the active server.

The Linux-HA (High-Availability) project [3] provides monitoring of cluster nodes, applica-

tions, and implements a sophisticated dependency model with a rule-based resource placement scheme. When faults occur, or a rules change occurs, the user-supplied rules are then followed to provide the desired resource placement in the cluster. The most common basic configuration for Linux-HA is that of a high-availability server which simply provides a single IP address to be fail-over between the nodes of the cluster.

Whereas the most reliable DB-Cluster solutions are still legacy (Oracle) and quite expensive, the HeartBeat at the end adds a new point of failure to the system. In this paper we present a simple but reliable solution to implement an Asterisk Cluster without using Heartbeat and DB-cluster.

The main idea of the proposed solution is to exhaust the IP Phones feature of being able to support the configuration of multiple server for SIP registration. For our deployment we used hundreds of Polycom IP phones. These IP phones implement a mechanism in which the IP Phone are registered on multiple SIP registrar servers (according to the provisioned list defining primary server, secondary one, ..). The Polycom IP phone sends the INVITE message to tier up a new SIP call first to the primary server. If the working server does not respond correctly to the INVITE message, then IP Phone tries to make the call by using the next server in the provisioned list. Moreover the IP Phone will attempt to re-register on the primary server, typically at intervals of 30 to 60 seconds, until the registration is successfully, and the primary server will return to being the working SIP server. So doing when a fault occurs in the primary, all the phones will

automatically switch to the secondary while they keep trying to reach the primary server. When the primary returns to be active again, the phones will switch automatically without any need of further operational commands.

This approach appears to be intrinsically robust and conceptually simple to be implemented, by exhausting the increased computational power at peer level (IP Phones). So doing the server side of the solution requires only a relative simple replication *middleware* to propagate Asterisk provisioned data toward the secondary node: there is no need of an (expensive) DB Cluster, or of a system such as an HeartBeat, that, though robust, is anyway a new point of failure. The robustness of the solution is intrinsically provided by the peer architecture.

The following paper describes the detailed experience gained with the deploy of an asterisk cluster for all the business VoIP calls for a big premises of one of most widely used Credit Card company in Italy.

## **Architecture**

The choice of the proposed architecture is a compromise between a sufficient reliability and a not excessive software redundancy that would increase too much the complexity of the solution.

In Figure 1 we depict the architecture of the cluster. It comprises two virtual machines (VM) based on ESX VMWare virtualization layer. Each VM is configured as follow:

- Linux Operating System **Ubuntu** 8.01 server edition <http://www.ubuntu.com/>
- The stable version **1.4.21.2** of **Asterisk** (<http://www.asterisk.org/view.php>) configured in Real Time architecture. In our case we used PostgreSQL DB 8.3.7.
- The cluster comprises up to 1000 Polycom IP Phones (330, 450 and 670) connected by SIP trunks to two Avaya Voicegateways (VG) interconnected to PSTN. The *codec* used is G.711 and Asterisk is not requested to perform transcoding.
- The two Asterisk VMs are configured: the former as active primary Asterisk the latter as secondary in hot stand-by. To replicate the data of PostgreSQL DB is used the *middleware* *pgpool* (<http://pgpool.projects.postgresql.org/>). So doing all the updates to the primary Asterisk RT Database are propagated to the secondary in a single transaction fashion, they are kept always synchronized. If a data propagation error should occur, it will be immediately reported into the logs.
- The provisioning and the configuration of the Polycom Phones and Asterisk DB is made by a novel Java Web2.0 application, called **Contacta** developed by our association and published, with some user limitations, as OS application at <http://www.openinnovation.it/contacta>. It is not mandatory to install this application to implement the Asterisk RT cluster, but doing it, may help you to manage large installation of IP Phones.
- The synchronization of Asterisk configuration files, user voicemail files, is

made by another OS tool called *csync2*, <http://oss.linbit.com/csync2/> that implements an asynchronous file synchronization in the two nodes of the cluster. As these files are seldom modified, such approach is adequate.

In case of fault of primary Asterisk VM, Polycom IP phones switch off immediately to the secondary that is kept updated by *pgpool* tool and only current established sessions are eventually lost. *Csync2* application makes it possible to minimize user uncomfotableness by propagating also voicemail messages from primary to secondary and Asterisk configuration files. When primary Asterisk is recovered the IP Phones will switch automatically to it.

## **Configuration and installation**

### **Asterisk Real Time configuration**

In such configuration, all database specific code of Asterisk, usually configured by files, is moved to database specific drivers. The channel just calls a generic routine to do database lookup. There are lots of tutorials [5] that detail how configure and install Asterisk in RT architecture. Asterisk RT supports different DataBase (MySQL, ODBC, OpenLDAP, ...); we have chosen PostgreSQL because of its clear Open Source licensing and the proven stability and reliability. In this paper we suppose that the Operating System (in our case Linux Ubuntu), Asterisk and PostgreSQL 8.3 are already configured.

The following families of Asterisk data are moved to the DB: *sippeers*, *sipusers*, *voicemail* and *meetme*. We have created the database

schema called *asterisk.sql* by implementing for each family a table of data as described in [6]. We have modified the file *res\_pgsql.conf* that is present in */etc/asterisk/* by inserting the data used for authentication:

```
[general]
dbhost=127.0.0.1
dbport=5432
dbname=contacta_pooled
dbuser=<dbusername>
dbpass=<dbpassword>
```

Before creating the PostgreSQL DB please read before the configuration of *pgpool* in the next paragraphs. As well as, the file *extconfig.conf* of the same directory should be modified as follow:

```
sipusers => pgsql,contacta_pooled,stsip
sippeers => pgsql,contacta_pooled,stsip
voicemail=>pgsql,asterisk_contacta_pooled,
stvmu
meetme => pgsql,contacta_pooled,stmem
```

where *pgsql* is the driver to be loaded (please check that Asterisk loads it correctly), *contacta\_pooled* is the name of the DB, *stsip*, *stvmu*, *stmem* are the names of the tables of *asterisk.sql* schema file.

In our deployment we have implemented advanced services such as presence and voicemail (with MWI, Message Waiting Indicator). Asterisk RT requires some “tricks” to allow these services to work properly. In particular you should add in Asterisk configuration file *sip.conf*, the following lines in the [general] section:

```
rtcachefriends=yes ; To enable MWI
and presence in RT configuration
limitonpeers=yes ; To enable MWI e
presence in RT configuration
rtupdate=yes ; To enable MWI e pres-
ence in RT configuration
notifyringing=yes ; To send the event
"telephone ringing" for presence applic-
ation
```

## Clustering Asterisk

The first step is to configure PostgreSQL to accept external connections (the default comprises only connection requests by *localhost*).

As *root* user please edit

*/etc/postgresql/8.3/main/pg\_hba.conf* that has the following default structure

```
# TYPE DATABASE USER CIDR-ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all ident sameuser
# IPv4 local connections:
host all all 127.0.0.1/32 md5
# IPv6 local connections:
host all all ::1/128 md5
```

Add the line to configure the *network/mask* from which to access in the IPv4 section, e.g. 10.0.0.0/8 or 192.168.1.100/24. For single host access the network mask is 32, as an example 10.0.0.1/32. The METHOD should be *password* for pgpool:

```
[...]
# IPv4 local connections:
host all all 127.0.0.1/32 md5
host all all 10.0.0.0/8 password
[...]
```

After the user should edit */etc/postgresql/8.3/main/postgresql.conf* and search for the sting *listen\_addresses*, the default value is *listen\_addresses='localhost'*. Please modify this value to:

```
listen_addresses = '*'
```

Then restart PostgreSQL application by typing:

```
/etc/init.d/postgresql-8.3 restart
```

Now you can check to connect by typing:

```
psql -h <IP Address> -U username data-  
basename
```

where <IP Address> is host IP address, *username* is the owner of Asterisk database, and *databasename* is the database name used by Asterisk RT (in our example is *contacta\_pooled*). Please repeat the same procedure for the secondary VM machine.

## Pgpool Configuration

Please install the pgpool<sup>1</sup> from Ubuntu repository.

Then configure the application as following:

```
listen_addresses = '*'
port = 5430
backend_host_name = '<primary backend  
hostname>'
secondary_backend_host_name = '<second-  
ary backend hostname>'
secondary_backend_port = 5432
replication_mode = true
replicate_select = true
enable_pool_hba = false
```

where the backend names are the host name (please update your */etc/hosts* file) of the two nodes and 5430 and 5432 a possible choice for listening port numbers.

The default comprises that the logging is redirect to *syslog*, otherwise you should edit */etc/default/pgpool*.

Then restart pgpool applications by:  
*/etc/init.d/pgpool restart*

Now you can repeat the installation of pgpool on the secondary node, taking care of inverting the backend hostname with the name of the secondary.

Please also refer to:

*/usr/share/doc/pgpool/README.gz*

---

<sup>1</sup>On Ubuntu we faced some problems to run pgpool2 (deb was version 1.3, last version is 2.2) so we installed pgpool version 3.4.1.

## Pgpool configuration

Now you should create a new database for Asterisk RealTime managed by pgpool, To connect to this Database you have to use 5430 port.

As an example let's create the new DB that we call *contacta\_pooled* by:

```
createdb -U username -W -p 5430
contacta_pooled
```

after having create the DB you have to apply the configuration and the data defined for Asterisk RealTime Database:

```
psql -U username -W -p 5430 -f asterisk.sql contacta_pooled
```

where *asterisk.sql* is the database schema and data of Asterisk RT DB. If you have installed **Contacta** please update *contacta.properties* file in Contacta installation directory with the following URL:

```
jdbc:postgresql://HOST:5430/contacta_pooled
```

## Configuration testing

When a node of the cluster faults, pgpool will use only the second node. Such working condition is called: *degeneration mode*. The operator before reintroducing the replication mode should check that the DB in the primary node and the one on the secondary are update each other. Otherwise the operator should manually synchronize them by making a dump from the primary and updating the secondary.

This is the output (please type: `tail -f /var/log/syslog` as root user) of the log if a node fails:

```
ERROR: pid 11816: health check failed.
master chrome at port 5432 is down
LOG: pid 11816: notice_backend_error:
master: 1 fail over request from pid 816
LOG: pid 11809: starting degeneration.
shutdown master host chrome(5432)
LOG: pid 11809: degeneration done. shutdown master host chrome(5432)
```

## Csync2 installation and configuration

Now you should install and configure *csync2* on both the VM nodes and generate the keys for SSL:

```
# apt-get install csync2 sqlite3 openssl
xinetd
# openssl genrsa -out
/etc/csync2_ssl_key.pem 1024
# openssl req -batch -new -key
/etc/csync2_ssl_key.pem -out
/etc/csync2_ssl_cert.csr
# openssl x509 -req -days 600 -in
/etc/csync2_ssl_cert.csr -signkey
/etc/csync2_ssl_key.pem -out /etc/csync2_ssl_cert.pem
# csync2 -k /etc/csync2.key.linuxCluster
# scp /etc/csync2.key.linuxCluster
root@<secon_cluster_node_IP>:/etc
```

Note that Csync2 requires the installation of the application Sqlite that is a light file DB and OpenSSL libraries.

Please configure the file */etc/csync2.cfg* as following showed. You should insert as hostname the two node cluster hostname (i.e., the output of hostname command). Moreover it is necessary that both the */etc/hosts* files contain the inserted hostname.

```
group linuxCluster
{
  host <hostname_primary>
  <hostname_secondary>;
  key etc/clustername.key;
  include /etc/asterisk/;
  include /home/contacta/polycom/roms/;
  include /home/contacta/contacta/conf/polycom/;
  include
  /var/spool/asterisk/voicemail/
  ;
}
```

```
auto younger;
}
```

So doing all files in the directories indicated in the Linux cluster are propagated from the primary to the secondary. The directories that start with `/home/contacta` are worth only if Contacta application is installed. We decided to configure Csync2 with the option `younger`: in case of a conflict between two files on the two cluster nodes, the file that was lastly update will be replicated.

Now copy the configuration on the secondary VM:

```
# scp /etc/csync2.cfg
root@<secondary hostname>:/etc
```

Run the application on both the nodes by following command:

```
csync2 -cr / ; csync2 -T
```

The command will produce as an output the list of the files that are present on the primary but not on the secondary node. Moreover SQLite DB will be populated. To propagate the updates from primary node to the secondary, type:

```
csync2 -xv
```

Now we can update the cron table with `csync2` synchronization command (we choose to synchronize every 6 minutes):

```
0-59/6 * * * * root csync2 -cr / ;
csync2 -T ; csync2 -x
```

Another solution is to write a shell script including the following commands: `csync2 -cr / ; csync2 -T ; csync2 -x` Operator can add logs command and other outputs

```
0-59/6 * * * * root
/usr/local/sbin/sync_nodes.sh
```

To copy the database from one node to the other one the operator may use the script:

`pg_dump` and `pg_restore`. It is safe to run these tools when the secondary node is disconnected.

```
export PGPASSWORD=<password>
export t=<nomefile>-<date>.psql.sql
pg_dump --clean -U <username> -h local-host -p 5432 --format=c contacta_pooled > $t &&
pg_restore --clean -U <username> -h 10.99.99.99 -p 5432 --dbname=contacta_pooled $t
```

## Troubleshooting

The format of the log of the application is the following: date (Mar 1 14:04:48, the format depends on the configuration of `syslog`), hostname (`jstar`), "`pgpool:`", `isodate` (2009-03-01 14:04:48), `level` (ERROR), `pid` (13630), and message. An example:

```
Mar 1 14:04:48 jstar pgpool: 2009-03-01
14:04:48 ERROR: pid 13630:
do_clear_text_password: failed to read
password packet "p"
```

If such error message appears please stop `pgpool` and restart it with the following options: `pgpool -d -n` in order to enable the debug mode. To stop it press `Ctrl-c`

Then type the following command:

```
* psql -U username -W -h <IP Address> -p
5430 contacta_pooled
```

The following log is not a real problem for `pgpool/postgresql`, but it only indicates the client

connection broke suddenly, as an example for a Kill or SIGHUP:

```
ERROR: ProcessFrontendResponse: failed to read kind from frontend. frontend abnormally exited
LOG: do_child: exits with status 1 due to error
```

The error in the syslog in case of an inconsistency is the following. Please note that this requires a manual synchronization of the DataBase:

```
ERROR: read_kind: kind does not match between backends master(67) secondary(68)
ERROR: pool_process_query: kind does not match between backends master(
LOG: do_child: exits with status 1 due to error
```

For more details please refer to:

<http://www.postgresql.org/docs/8.3/static/high-availability.html>  
<http://edoceo.com/liber/db-postgresql-replication>  
<http://pgfoundry.org/projects/pgpool>

## Conclusions

We have illustrated in this paper an architecture to assess high availability to an Asterisk Cluster in order to reduce significantly the overall system failure rate, while offering the same or better overall performance.

This solution requires standard replication tools without the need of using (expensive) DB legacy DB cluster solutions, and new point of failure as HeartBeat. Moreover it is exhausted native features of IP Phones such as multi server registration. All this makes robust the

proposed solution and suitable for large implementation of Asterisk. So doing this approach may be an ideal solution for many telecom applications in which extremely reliable operation, maximum availability, minimal maintenance and low life cycle cost are essential.

## Authors

**Roberto Grasso** holds the following degrees: one in Electrical engineer at Politecnico of Torino, a MS degree in Electrical engineer at the University of Texas at Dallas, a Bachelor degree in Economics at the University of Torino. With more than 10 years in ICT and telecommunication sector, he has served as a senior consultant in international companies such as CriticalPath, Lucent Technologies, Athos Origin, JnetX, and Accenture. He is an expert of important Open Source project such Asterisk, Ser, OpenSIPS and Mobicents. He is president and founder of the association Open Source Innovation.

**Michele Bianchi** holds a Laurea degree in Computer Science at University of Verona . With more than 10 years in IT and software development sector, he has served as a senior consultant in corporates such as Telcordia Technologies, Vodafone, RBS, GlaxoSmithKline and Unicredit. He is an expert on JEE and Agile methodologies and many well-known Open Source projects such Springframework, Maven, Liferay, jBoss and Mobicents. He is founder of the association Open Source Innovation.

## REFERENCES

- [1] Vivek Chopra et al., Professional Apache Tomcat 5, Addison-Wesley, 2008
- [2] Felber, P., D'efago, X., Guerraoui, R., and Oser, P., "Failure detectors as first class objects", In *Proceedings of the 9th IEEE Int'l Symp. on Distributed Objects and Applications(DOA'99)*, pages 132-141, Sep. 1999.
- [3] <http://www.linux-ha.org/Heartbeat>
- [4] <http://www.voip-info.org/wiki/view/Asterisk+High+Availability+Solutions>
- [5] <http://www.voip-info.org/wiki/view/Asterisk+Real-Time>
- [6] <http://www.voip-info.org/wiki/view/Asterisk+RealTime+PostgreSQL>

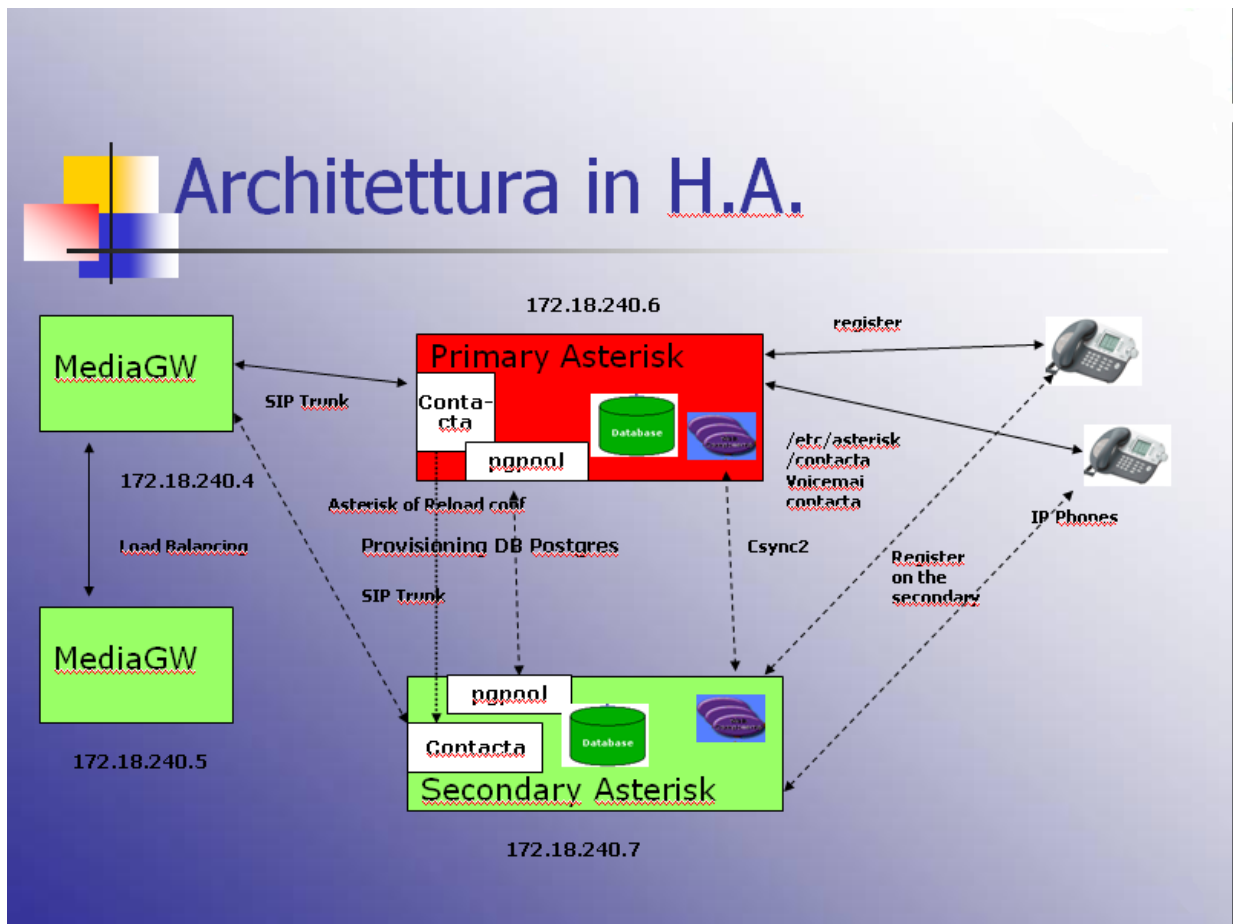


Figure 1: Architecture of the Asterisk cluster