Cedrus: a PostgreSQL management solution

Rodrigo Hjort [a)]
Companhia de Informática do Paraná
Curitiba, PR, Brasil

In the present work it will be shown the analysis, design and technical details on the development of the Cedrus project, a software solution that aims to manage and monitor PostgreSQL DBMS servers. Its main objectives rely on the environment difficulties PostgreSQL administrators have to live with and the lack of software tools that could help their work become easier and most productible.

I. INTRODUCTION

a. Subject

The main subject concerned in this study is Database Management Systems (DBMS), although Operating Systems, Software Engineering, Network and other knowledges on Information Tehcnologies are also involved.

b. Object

The project, named Cedrus (pronounced "*Kedrus*") - see **Figure 1**, has basically these tasks: to monitor and manage PostgreSQL database systems.

Figure 1. The Cedrus project logo.

c. Issue and propositions

Currently there is no tool designed to manage and monitor PostgreSQL [1,4] environments. When something goes wrong with the server, the administrators are alerted when it has already been discovered by the database users.

Today, PostgreSQL servers on the company are monitored via Unix native programs, such as: *top*, *sar*, *vmstat*, *tail -f*, *mail*, and *cron*. Besides the fexibility these programs provide, when the number of PostgreSQL servers arise, the administrator starts to feel the difficulties in maintaining several distinct instances.

Another point to be noted is pro-active monitoring on certain resources related to the server. On the company, there is already a home-made solution (GOPMP) to monitor general server and specific services availability. It is based on SNMP (Simple Network Management Protocol), and answers for multi-platform environments, and can handle things like: disk utilization, swap area utilization, memory and CPU usage. However, as it is a generic server solution, it might not handle database specific counters, such as: tablespace usage, lack of indices on tables resulting on bad queries, paging/swapping, among others.

It was recently made an effort in order to analyse PostgreSQL logfiles to search for: most frequent and burdenble queries, intermittent error messages, least (or never) used indices on tables, most resource consuming databases on an instance, and so. The problem is that is not an easy configurable, flexible and joined solution, as it is based on scripts in Shell that make use of *awk*, *sed*, *grep*, *sort*, and other utilities.

The last point to be discussed concerns to backup strategies. Since it was possible to perform physical backup with LOG archiving on PostgreSQL 8.0, the task of doing so is on the DBA hands. Besides this Full Online backup, we still execute logical (Dump) backup, mainly for historical and auditing purposes. After each backup, the script sends an e-mail to the server administrators telling its success (or cause of fail). With few PostgreSQL instances, it was still easy to control. The problem started when lots of

[a)] E-mail address of the author for contact:
rodrigo dot hjort at gmail dot com

servers were installed and all them needed those two types of backup. It became difficult to keep in touch with that. With backups, one could also remind of the so-called "jobs". As PostgreSQL does not have it natively, like proprietary DBMSs do, tasks are implemented using Shell Script and tools like *crontab*, which are not integrated to the database system.

Therefore, the main goal on this project is to develop a software tool which could fulfill all those needs pointed out. Some examples of programs that make this role on proprietary RDBMS include:

- Oracle Enterprise Manager 10g [2] (for Oracle)
- Microsoft SQL Server Profiler and Query Analyzer [3] (for SQL Server)

d. Specific aims

Since the issues faced by the administrators on the company were listed on the later topic, here are the specific aims that Cedrus has to be able to achieve on the servers (i.e: the targets):

- **Availability monitoring**: a continuous and repeated test if the machine is online – *ping*, and whether the services are responding – *nmap*;
- **Pro-active monitoring**: with this feature, the system aims to detect future lack of resources before they become a real problem, e.g: disk or partition utilization or some kinds of bottlenecks;
- **Storing operating system counters and database statistics information**: with a central repository, collecting data regarding to all servers would made it possible to properly detect anomalies on them;
- **Storing relevant entries made to the logfiles**: with this feature, it would be possible to study stuffs like top consumers SQL queries, full scans made to the database, to detect frequent errors happening on the instance and also to warng from vital or suggestion messages (e.g: checkpoint frequency);
- **Trends analysis**: with the server counters stored in the repository, one could, for example, obtain the best date and time for a maintenance blackout, or check wheter a specific counter is above normal conditions or not. Besides, we could make linear progressions in order to know how much a specific table or index grows monthly – this could help on organizing tablespaces;
- **Obsolete and missing objects detection**: by reading the last database statistics, the system could point unused indices, which might then be candidated for destroying. Yet, based on the most frequent and longest-time queries, Cedrus could analyse the SQL statement and suggest the creation of indices for the involved tables based on the execution plan;
- **Executing and controlling automated tasks**: with this, backups and jobs in general could be scheduled in Cedrus, which then would trigger and control them, releasing this charge from *crontab* or the administrators;
- **Remote instance management**: instead of doing an SSH to the server and or using several applications, we could also perform administrative tasks (e.g: creating users and databases, modifying postgresql.conf and pg_hba.conf) on Cedrus interface.

e. Methodology

Based on the specific aims, here are presented possible manners for implementing each one of the listed items:

- **Availability monitoring**: there must be a continuous program running in background to periodically *ping* the targets and test for connection on the services (i.e: ports) they offer. The program must be coded on a programming language capable of connecting to the system database and be able of executing the proposed tasks;
- **Pro-active monitoring**: after collecting the operating system counters, the same program mentioned above should check for some rules

defined by the Cedrus administrator concerned to each target separately. If the condition is matched (called a baseline threshold), the administrators involved should be alerted immediately;

- **Storing operating system counters and database statistics information**: those informations should be arranged on a PostgreSQL database. It is very important to collect and save to the repository only relevant data, which should be used later for some kind of analysis. It is also important to observe and calibrate the ideal gathering frequency, as the repository could grow suddenly (e.g: indices or table statistics could be collected once a day or a week, as they do not change so much). There must be developed scripts to be executed on the target to gather operating system data, and an interface to call it from a program to populate the respective tables on the repository;
- **Storing relevant entries made to the logfiles**: working as a *tail -f* command, there must exist a target-side program that should read, filter, apply substitutions, parse and save the entries made to the PostgreSQL configured logfile. As it is common to use rotation (by time and size) on these text files, the script should be prepared to this feature. The lines should then be stored on a temporary repository, for further transferring to the central one;
- **Trends analysis**: taking on account counters filtered on a date range, one could calculate its averages, standard deviation and other statistics functions in order to know its normal behavior. Also, on the repository will be stored several snapshots of the targets, chronologically ordered;
- **Obsolete and missing objects detection**: these features should be invoked by the Cedrus administrator on a friendly interface, after selecting a specific target. The most frequent queries and object statistics must already be on the repository, and based on this the system would present the lack or excess of indices for the tables involved;
- **Executing and controlling automated tasks**: there must exist tables on the repository to store the schedules and execution informations for the jobs assigned to the target. To run it remotely, a mechanism must be developed and a program or script must be installed and executed on the target;
- **Remote instance management**: most tasks could be performed directly with the help of a super user or well-granted user on the PostgreSQL target instance. Other tasks, like those that modify PostgreSQL configuration files, must be accomplished by using other resources, such as native functions to read and write text files via SQL instructions.

f. Activities chronogram

In order to achieve the building of this software, there are many tasks that must be accomplished. They were arranged in a chronological order of execution:

1. Essential analysis to the necessary data to be collected in OS and DBMS [21];
2. Syntax and functioning of every collecting program (ie: sar, vmstat, iostat) [11-14];
3. Repository database modeling on DBDesigner [5] (open source visual database designer for MySQL);
4. Creation of Shell Scripts and PL/sh functions to collect data from the Operating System [15,16,20];
5. Agent module environment: Shell Script and Cron or a background program [17];
6. Development of scripts to the Agent gather the data and save them into a Repository;
7. Collecting periodicy definition;
8. Definition of mechanism to read and store the lines written in the DBMS logfile (“tail -f” or script in Perl);

9. Development of a program to monitor downtimes;
10. Development of scripts to manual collecting of old version logfiles - importing;
11. Definition of ways of sending alerts to the administrators;
12. Raising of reports and graphs needed;
13. Definition of functions and flows in the WEB frontend (Manager);
14. Development of the Manager module in Ruby on Rails [9];
15. Search for a solution for creating graphics and charts and sending it via stream (gnuplot, bind in Ruby)

II. CONSIDERATIONS

a. Target

Each monitored server is called a "target". It consists at least of a PostgreSQL server.

b. Domain

A set of two or more targets is called a "domain". Its goal is to agilize tasks which have to be executed on several targets considered a group of common purposes (eg: two Application Servers and PostgreSQL server).

c. Administrator

A person who is responsible for, or participates in the administration and monitoring of a target, is called an "administrator". This could include the DBA and AD staff.

d. Super Administrator

Someone who is in charge to maintain the monitoring environment is a "super administrator". It has powers to create and maintain administrators and targets, assign administrators to targets, and change general settings. Yet, a super administrator can perform anything an administrator does, but on every target. The interaction between the *Administrators* (live party) is made by e-mail (i.e: SMTP) and Internet / Intranet (i.e: HTTP).

III. ARCHITECTURE

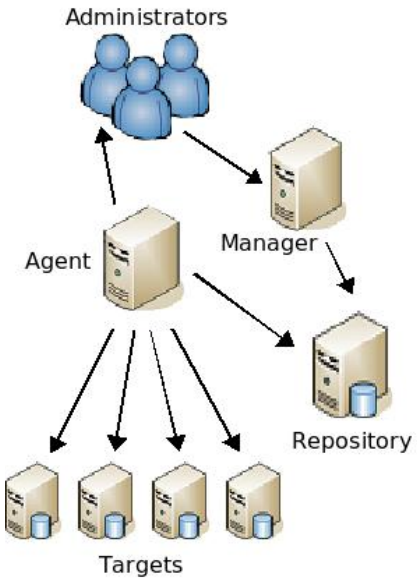

Figure 2. Architecture used in the Cedrus project. The *Agent* collects data from the *Targets*. These informations are stored in the *Repository*. *Administrators* receive notifications by the *Agent* and analyse their database servers using the *Manager*, which reads data from the *Repository*.

This section aims to explain what will be assigned to each party on the architecture shown in the **Figure 2**. The network protocol used on most of traffic is TCP. Between *Targets, Agent, Repository* and *Manager*, the PostgreSQL port, with a 5432 default number, is used to transfer data via SQL instructions.

a. Manager

This party is an application server which is an interface to the collected data in the repository. Hence, via web these functions can be performed:

- listing of most frequent queries;
- real time resources utilization on a target;
- trends analysis on a given target and period;
- remote scripts and SQL statements executions;
- database objects utilization granulated and general analysis.

b. Agent

This module is a continuous running program (i.e: a background process or daemon) which does:

- availability tests on the targets;
- service response tests on the targets;
- starting of jobs (operating system scripts or database stored procedures) in the targets;

- periodical gathering of data from the targets via SQL interface;
- analysis of metric baselines thresholds;
- creation of alerts assigned to targets;
- sending of notifications (e-mail, SMS, etc) to administrators based on triggered alerts.

c. Repository

The Repository is responsible for storing the information concerned to all targets. The data, arranged in a RDBMS (PostgreSQL), is composed of:

- historic of all counters collected from a target;
- historic of all operations done on a target;
- information of downtimes, alerts and notifications regarding a target;
- database, table and index specific informations from a target;
- specific configurations for inspection on each target;
- general settings for the whole environment.

d. Host (Target)

Is actually a set of programs and technologies installed on the target, being able to:

- collect hardware information (eg: cpu, memory, disk, network);
- collect database information on the PostgreSQL cluster;
- collect, filter and stream edit text lines sent to the DBMS logfile;
- start and monitor scripts (ie: running jobs).

It is necessary a PostgreSQL service running on the target to the functioning of the Host part, even though its main deal is another program (eg: JBoss Application Server or Apache Httpd).

On the PostgreSQL server, a database dedicated to the monitoring, holds views to access other database information and functions to collect operating system information. There is a mechanism to read the database system log messages (eventually sent to a file rotated periodically).

On hardware restricted environments, it is possible to have *Manager, Agent* and *Repository* parties can be installed on the same machine.

IV. TECHNOLOGIES INVOLVED

a. Manager (Client)

- **Web browser** (HTTP): supported by almost every browser in agreement with W3C patterns [6].

b. Manager (Server)

- **LightTPD**: a thin HTTP server that can interpret Ruby language and yet is designed and optimized for high performance environments [7];
- **Ruby on Rails (RoR)**: a web development framework in Ruby language based on Object Oriented Programming. A full stack, Web application framework optimized for sustainable programming productivity, allowing writing sound code by favoring convention over configuration [9];
- **Gruff**: a graphing library project to make graphs with Ruby language. Released also as a Ruby on Rails plugin, can produce inline PNG figures on webpages [10];
- **PostgreSQL client**: native connection to PostgreSQL databases [1,4].

c. Agent

- **Ruby**: the language which the agent daemon is written on - a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write [8];
- **Shell Script**: in order to maintain the *Agent* daemon always running, some scripts are assigned to make this role [8,15,16];
- **PostgreSQL client**: native connection to PostgreSQL databases [1,4];
- **cron**: daemon to execute scheduled commands on Unix [17];

- **exim4**: a mail transfer agent (MTA) developed at the University of Cambridge [18].

d. Repository

- **PostgreSQL server**: PostgreSQL, the highly scalable, SQL compliant, open source object-relational database management system [1,4];
- **DBLink**: a set of functions in C returning results from a remote database to use in PostgreSQL [4];
- **PL/pgSQL**: default procedural language in PostgreSQL. Some functions designed to collect database information on the targets are written on this language [19].

e. Host (Target)

- **PostgreSQL server**: the database instance itself to be monitored and analysed [1,4];
- **PL/pgSQL**: needed to filter and format the data collected [19];
- **PL/sh**: to create functions in Shell Script to collect hardware information [20];
- **DBLink**: used in functions that make access on other databases to collect their data [4];
- **sysstat**: the package which provides *sar* monitoring program [12];
- **iostat**: report Central Processing Unit (CPU) statistics and input/output statistics for devices and partitions [11];
- **gawk**: a pattern scanning and processing language. There are some AWK functions used in the *Host* that need this package [15];
- **sed**: stream editor for filtering and transforming text [16].

V. DATA COLLECTING

One of the first tasks in the project is to arrange what kind of counters should be collected stored and analysed. These might be relevant factors that could contribute to future studies in performance or failures in the targets.

According to inspections on what proprietary solutions offered and a basic knowledge on Operating Systems working, the informations were assembled categorically.

There are basically two slopes to be divided the counters:

- Operating System data;
- Database System data.

Considering Operating System, there are four important factors to take into account on DBMS:

- CPU;
- Memory;
- Disk;
- Network.

These are fundamental points when searching for a lack or poor performance on a server.

In the second part, the DBMS involves mainly:

- Data volume;
- Utilization and number of users;
- Logical and physical distribution.

Based on these preliminary assumptions and on what data could be extracted from the environment by the available tools, the counters were organized into the tables below.

a. Operating System

Counter name	Description
uptime	How long the system has been running.
run queue size	Run queue length (number of processes waiting for run time).
process list size	Number of processes and threads in the process list.
load average in the past minute	System load average for the last minute.
load average in the last 5 minutes	System load average for the past 5 minutes.
load average in the last 15 minutes	System load average for the past 15 minutes.
context switches	Number of context switches per second.

System load averages is the average number of processes that are either in a runnable or

uninterruptable state. A process in a runnable state is either using the CPU or waiting to use the CPU. A process in uninterruptable state is waiting for some I/O access, eg: waiting for disk. The averages are taken over the three time intervals. Load averages are not normalized for the number of CPUs in a system, so a load average of 1 means a single CPU system is loaded all the time while on a 4 CPU system it means it was idle 75% of the time.

b. Processing (CPU)

Counters collected towards the whole machine and individually on each processor (on SMP machines).

Counter name	Description
% user (applications)	Percentage of CPU utilization that occurred while executing at the user level. Time spent running non-kernel code (user time).
% nice (privileged)	Percentage of CPU utilization that occurred while executing at the user level with nice priority.
% system (kernel)	Percentage of CPU utilization that occurred while executing at the system level (kernel). This does not include time spent servicing interrupts or softirqs. Time spent running kernel code (system time).
% IO wait	Percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request. Time spent waiting for IO. Prior to Linux 2.5.41, included in idle.
% steal	Percentage of time spent in involuntary wait by the virtual CPU or CPUs while the hypervisor was servicing another virtual processor. Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown.
% idle	Percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request. Time spent idle. Prior to Linux 2.5.41, this includes IO-wait time.

c. Memory

Counter name	Description
free memory	Amount of free memory available in kilobytes.
used memory	Amount of used memory in kilobytes. This does not take into account memory used by the kernel itself.
used memory percentage	Percentage of used memory.
buffered memory	Amount of memory used as buffers by the kernel in kilobytes.
cached memory	Amount of memory used to cache data by the kernel in kilobytes.
free swap space	Amount of free swap space in kilobytes.
used swap space	Amount of used swap space in kilobytes.
used swap space percentage	Percentage of used swap space.
cached swap space	Amount of cached swap memory in kilobytes. This is memory that once was swapped out, is swapped back in but still also is in the swap area (if memory is needed it doesn't need to be swapped out again because it is already in the swap area. This saves I/O).

d. Disk Devices

Counters collected towards the whole machine and individually on each device (*).

Counter name	Description
transfers per second	Number of transfers per second that were issued to the device. A transfer is an I/O request to the device. Multiple logical requests can be combined into a single I/O request to the device. A transfer is of indeterminate size.
blocks read/s	Indicate the amount of data read from the device expressed in a number of blocks per second. Blocks are equivalent to sectors with 2.4 kernels and newer and therefore have a size of 512 bytes. With older kernels, a block is of indeterminate size.
blocks written/s	Indicate the amount of data written to the device expressed in a number of blocks per second.
blocks read	The total number of blocks read.
blocks written	The total number of blocks written.
total size*	The amount of disk space on the file system.
used space*	The amount of disk space used on the file system.
percentage of utilization*	

Disk space is shown in 1K blocks by default, unless the environment variable POSIXLY_CORRECT is set, in which case 512-byte blocks are used.

e. Network

Counters collected towards the whole machine (*) and individually on each network interface.

Counter name	Description
total sockets*	Total number of used sockets.
TCP sockets*	Number of TCP sockets currently in use.
UDP sockets*	Number of UDP sockets currently in use.
raw sockets*	Number of RAW sockets currently in use.
IP fragments*	Number of IP fragments currently in use.
packets received / sec	Total number of packets received per second.
packets trasmitted / sec	Total number of packets transmitted per second.
bytes received / sec	Total number of bytes received per second.
bytes trasmitted / sec	Total number of bytes transmitted per second.
compressed packets received / sec	Number of compressed packets received per second (for cslip, etc).
compressed packets trasmitted / sec	Number of compressed packets transmitted per second.
multicast packets received / sec	Number of multicast packets received per second.

f. Paging / Swapping

Counter name	Description
pages paged in / sec	Total number of kilobytes the system paged in from disk per second. With old kernels (2.2.x) this value is a number of blocks per second (and not kilobytes).
pages paged out / sec	Total number of kilobytes the system paged out to disk per second. With old kernels (2.2.x) this value is a number of blocks per second (and not kilobytes).
faults/s	Number of page faults (major + minor) made by the system per second (post 2.5 kernels only). This is not a count of page faults that generate I/O, because some page faults can be resolved

	without I/O.
major faults/s	Number of major faults the system has made per second, those which have required loading a memory page from disk (post 2.5 kernels only).
pages swapped in / sec	Total number of swap pages the system brought in per second.
pages swapped out / sec	Total number of swap pages the system brought out per second.

g. Processes

Processes are the programs in execution in the operating system. They can be on several states, eg: in a sleep or running condition. Every process has an unique identifier, most of time called simply the PID.

Counter name	Description
PID	Process ID number of the process.
user name	Effective user name. This will be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.
program	Command name (only the executable name). Modifications to the command name will not be shown.
command line	Command with all its arguments as a string.
% CPU utilization	CPU utilization of the process. Currently, it is the CPU time used divided by the time the process has been running (cputime/realtime ratio), expressed as a percentage.
% memory utilization	Ratio of the process's resident set size to the physical memory on the machine, expressed as a percentage.

h. Database System Activity

These are counters that indicate activities on a database instance.

Counter name	Description
sessions	Number of connected users.
running queries	Number of running queries.
locks	Session locks.

i. Databases Statistics

Here are informations regarding to a whole database.

Counter name	Description
database size (MB)	Size in megabytes occupied by the database (all objects).
database cache hit ratio (%)	The rate in which data are found on memory (logical read) instead of going into the disk (physical read). The higher, the better.
transactions per minute	The amount of transactions made to the database. Few transactions indicate a static database.

j. Tables Statistics

These are counters specific to tables on a database.

Counter name	Description
table size (MB)	Size in megabytes occupied by only the table.
table total size (MB)	Size in megabytes occupied by the table (including data, toasted and indices structures).
table cache hit ratio (%)	The rate in which data are found on memory (logical read) instead of going into the disk (physical read). The higher, the better.
table index scan rate (%)	The rate that indicate wheter index scans (generally faster) are executed more than full scans. The higher, the better.

k. Indices Statistics

These are counters specific to indices on a table.

Counter name	Description
index size (MB)	Size in megabytes occupied by the index.
index utilization (%)	When using index scans, this counter reflects the percentage of utilization of this particular index regarding to other indices from the same table.

l. Logging

The partial reads on the logfiles are arranged in counters to advice different situations occurring in the DBMS.

Counter name	Description
errors	Error level messages sent to the server logfile.
warnings	Warning level messages sent to the server logfile.
notices	Notice level messages sent to the server logfile.
timed out queries	Statements that take more that a certain time to execute are logged to further analysis.
WAL recycling frequency	Each time a write-ahead LOG (i.e: PostgreSQL transaction LOG) is recycled, a line is appended to the logfile. Frequent recycling indicates huge load of modifications to the database instance.

VI. DATABASE MODELING

The modeling of such a complex environment is divided on two branches:

- **local**: data stored on the Repository;
- **remote**: data stored temporarily on the Target, for future sending to the Repository.

Furthermore, the local data is logically federated in: general settings, target configurations, histories, and violations, alerts and actions. Following is an extended explanation on each of these subject areas.

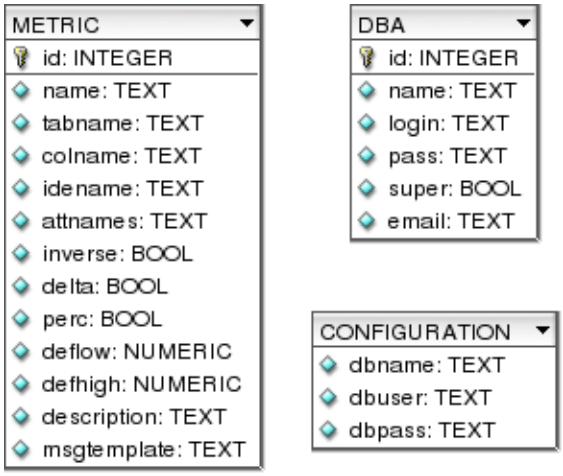

Figure 3. Modeling: General Settings subject area.

a. General settings

These tables are concerned to no specific target. They serve all the system, such as the users. See **Figure 3**.

- **CONFIGURATION**: general settings used by the software, such as host, database name, user and password to connect to the targets.
- **DBA**: holds name and contact information from the database administrators. Still, there are fields like login name, password and super user indication, all used to connect to the Manager module.
- **METRIC**: these are common counters which could be assigned to the targets monitoring. Some examples should include: CPU utilization percentage, swap disk utilization, and number of concurrent transactions on a database.

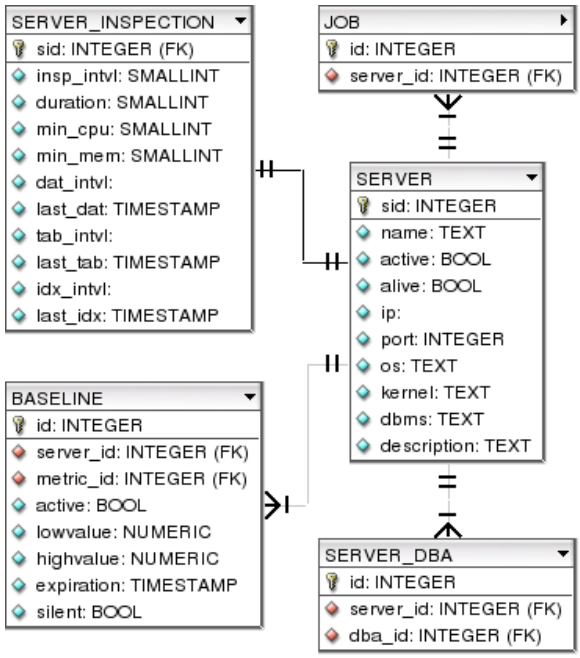

Figure 4. Modeling: Target Configurations subject area.

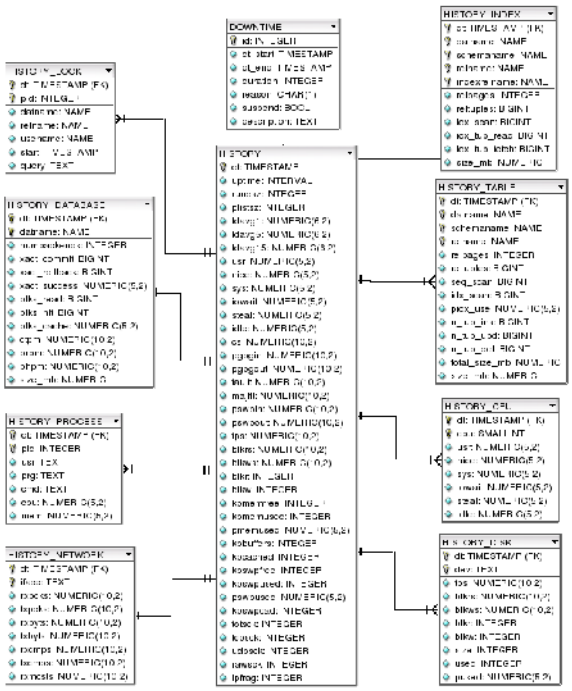

Figure 5. Modeling: Histories subject area.

b. Target configurations

These are settings regarding to the targets, which will be used for inspection and control. See **Figure 4**.

- **SERVER**: holds the statical records of the targets, such as: name, IP address, port, OS, DBMS and kernel versions.
- **SERVER_INSPECTION**: this table contains informations regarding inspections for each target. For example: what is the frequency of collecting indices information and when is the next collecting of these data. The frequencies of databases, tables and indices can be specified, and different configurations can be assigned to the targets.
- **SERVER_DBA**: is the link between a target and administrators involved with it. Each DBA have (i.e: can manage) several targets, and each target may have several administrators.
- **BASELINE**: in addition to the metrics, it is a threshold trigger assigned to each target, defining a warning and a critical limiar.
- **JOB**: this table stores the services a target should perform, executing remote scripts or stored procedures.

c. Histories (collected data)

These are the most large in volume tables, due to the frequency of insertion on them. See **Figure 5**.

- **HISTORY**: most important and relevant data concerned to the target hardware are arranged in the lines of this table. This includes: general CPU utilization, memory utilization, general disk utilization, general network traffic, paging and swapping, and run queue list.
- **HISTORY_CPU**: informations specifics to a specific CPU is stored there. On SMP (i.e: multi-processor) machines, there will be several entries for each inspection on this table. The CPU identifier starts from 0 to N – 1, where N is the total number of cores.
- **HISTORY_DISK**: data such as transactions (read, write, or both) made in certain times on a physical device, and its percentual utilization are stored on this table.
- **HISTORY_NETWORK**: separated into network interfaces, this table holds the number and size of packages sent and received by the network.
- **HISTORY_PROCESS**: most relevant processes are stored in this table. For each

target shoud be configured a minimum memory and CPU percentage utilization for filtering.

- **HISTORY_DATABASE**: data from the several databases of a target. This includes: number of active sessions (connected users), amount of commits and rollbacks made, blocks read and hit, and size of a database.
- **HISTORY_TABLE**: contains records of every table on a database. There are data such as: approximate number of tuples and pages, number of sequential and indexed scans made, amount of each DML statement performed and size of a table (including and excluding its toast structures and indices).
- **HISTORY_INDEX**: data concerned to indices of a table. We could list: approximate number of tuples and pages and the size of an index in megabytes.
- **HISTORY_LOCK**: informations concerned to locks on relations occurred in the database.
- **DOWNTIME**: when a target becomes inactive, data can not be collected. Therefore, is inserted a line on this table to indicate the time the target passed in unavailable state.

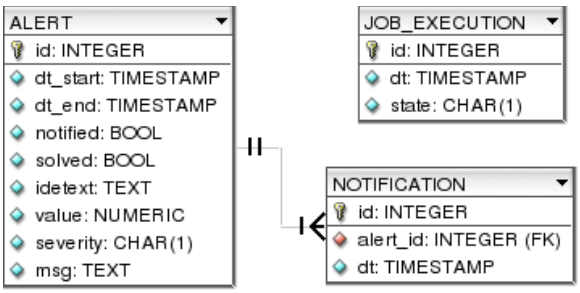

Figure 6. Modeling: Violation, Alerts and Actions subject area.

d. Violations, alerts and actions

These tables are concerned to events that happens on the targets, such as alerts, notifications and job executions. See **Figure 6**.

- **ALERT**: when a target gets unavailable or when baseline thresholds are reached, alerts are created. This table holds the historic of all alerts from a target.
- **NOTIFICATION**: when an alert is created, according to configurations and severity, notifications are sent to the target administrators.
- **JOB_EXECUTION**: this table stores the execution of every job configured for a target.

VII. RESULTS

After searching for the data to collect, analyse similar proprietary tools, design a technical solution and develop it, the application could be used in the running servers. See in the **Figures 7-11** some screenshots of the *Manager* module.

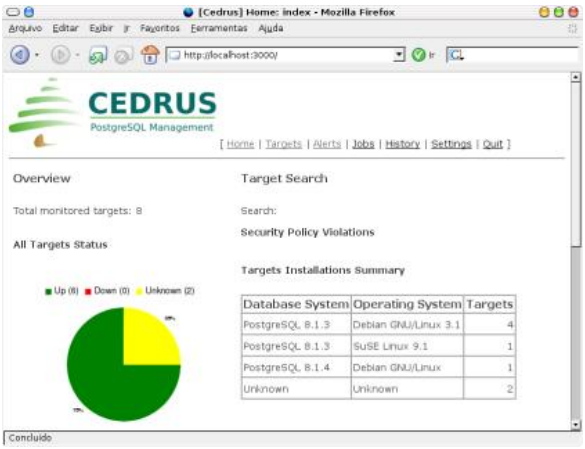

Figure 7. Snapshot: Home view.

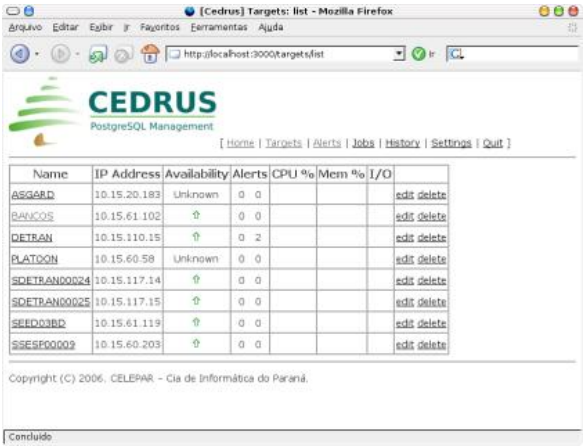

Figure 8. Snapshot: Targets view.

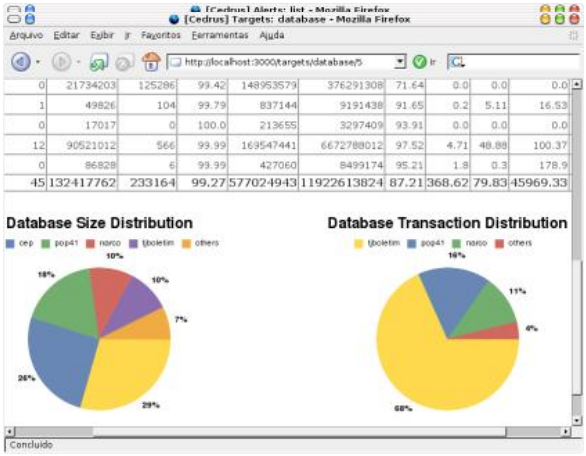

Figure 9. Snapshot: Target Databases view.

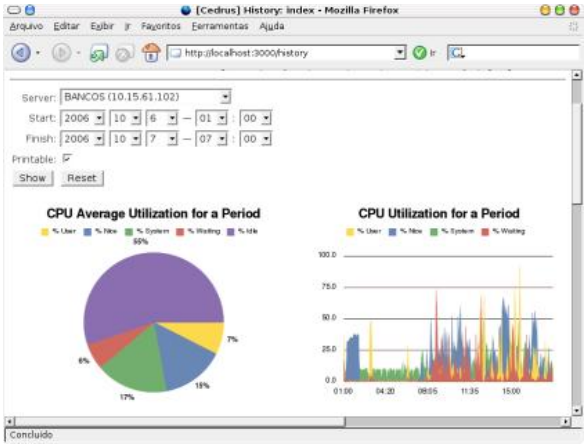

Figure 10. Snapshot: History view.

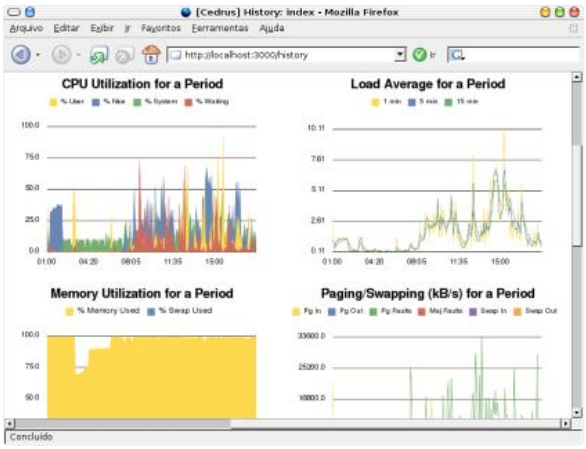

Figure 11. Snapshot: History view.

VIII. CONCLUSIONS

After having the software colecting the data from targets in development, test and production on the company, we were able to analyse better the behavior of each one. One of the main points was that, with Cedrus, one could definitely have something to measure and then determine whether or not a given PostgreSQL server is on heavy load.

IX. REFERENCES

1. PostgreSQL 8.1 Database Management System, on http://www.postgresql.org/
2. Oracle Enterprise Manager 10*g,* on http://www.oracle.com/enterprise_manager/
3. SQL Server Profiler and Query Analyzer, on http://www.microsoft.com/brasil/msdn/Tecnologias/aspnet/SQLProfilerQueryAnalyzer.mspx
4. PostgreSQL 8.1 Documentation, on http://www.postgresql.org/docs/
5. fabFORCE.net DBDesigner 4, on http://fabforce.net/dbdesigner4/
6. World Wide Web Consortium (W3C), on http://www.w3.org/
7. LightTPD webserver, on http://www.lighttpd.net/
8. Ruby Programming Language, on http://www.ruby-lang.org/
9. Ruby on Rails, http://www.rubyonrails.org/
10. Gruff Graphing Library, on http://nubyonrails.com/pages/gruff/
11. I/O Stat Manual Pages, on “man iostat”
12. SAR Manual Pages, on “man sar”
13. PS Manual Pages, on “man ps”
14. DF Manual Pages, on “man df”
15. Gnu AWK Manual Pages, on “man gawk”
16. SED Manual Pages, on “man sed”
17. Cron Manual Pages, on “man cron”
18. Exim4 Manual Pages, on “man exim4”
19. PL/pgSQL - SQL Procedural Language, on http://www.postgresql.org/docs/8.1/interactive/plpgsql.html
20. PL/sh - Procedural Language Handler, on http://plsh.projects.postgresql.org/
21. A. Tanenbaum, *Sistemas Operacionais Modernos*, 2a Edição, Prentice Hall.