



epsc

**Escola Politècnica Superior
de Castelldefels**

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: A High Availability platform design using Heartbeat and integration in a production environment

MASTER DEGREE: Master of Science in Telecommunication Engineering & Management

AUTHOR: Abraham Iglesias Aparicio

DIRECTOR: Jordi Guerrero Fau

SUPERVISOR: Jesús Alcober Segura

DATE: May, 29th 2008

Title: A High Availability platform design using Heartbeat and integration in a production environment

Author: Abraham Iglesias Aparicio

Director: Jordi Guerrero Fau

Supervisor: Jesús Alcober Segura

Date: May, 29th 2008

Overview

Nowadays, the number of services in the Internet grows more and more. Hardware vendors bring to market more powerful and stable servers. Operative systems are becoming more robust and flexible offering loads of possibilities. Nevertheless, service outage might happen if one of these components crashes.

When critical applications come to concern, it is a must to consider high availability techniques. By applying them, critical applications would be still available even in case of hardware, connectivity components or operative system failure.

On one hand, functional description, component architecture and a comparison between three software-layer high availability solutions is written.

On the other hand, it is necessary to enable SNMP protocol in every server, as the platform must be installed in a production environment within mobile operator's network. Integration with SNMP manager used by the customer must be achieved. A brief study of the protocol and its components are explained.

Platform design and implementation have been done in a development scenario. Beside, the client has another identical scenario to approve software and platform. This demo scenario has the same configuration than the development and production scenario. When the system has been approved, then production scenario must be configured in the same way than development and demo scenario. To deploy configuration and software releases, install scripts were implemented and packaged in RPM.

Lastly, a high availability platform running critical services with SNMP monitoring has been achieved. Moreover, integration within mobile operator's network has been done successfully. Since October 2007, the system is offering services running on the high availability platform implementation based on this project.

Título: Diseño e implementación de un cluster de alta disponibilidad con Heartbeat e integración en un entorno en producción

Autor: Abraham Iglesias Aparicio

Director: Jordi Guerrero Fau

Supervisor: Jesús Alcober Segura

Data: 29 de Mayo de 2008

Resumen

Hoy en día, los servicios online accesibles desde Internet crecen cada vez más. Fabricantes de servidores sacan al mercado servidores más potentes y más estables. Los sistemas operativos ofrecen cada vez más posibilidades y son más robustos. Sin embargo, un servicio podría estar no disponible si uno de estos componentes falla.

Cuando la criticidad de la aplicación aumenta, es conveniente considerar técnicas de alta disponibilidad. Aplicando dichas técnicas, la aplicación seguiría funcionando incluso con fallos del hardware, conectividad y sistema operativo.

Por una parte, se describe el funcionamiento, la arquitectura de componentes y una comparativa de cada una de las tres soluciones de software de alta disponibilidad.

Por otra parte, la plataforma diseñada deberá estar en un entorno en producción dentro de la red de un operador. Además, se debe integrar la monitorización con el software de gestión SNMP del cliente. Por consiguiente, es necesario estudiar las posibilidades que ofrece el protocolo SNMP.

El diseño y la implementación de la plataforma se han hecho en un entorno de desarrollo. Además, el cliente tiene un escenario de maqueta, idéntico al de desarrollo, para homologar el funcionamiento del servicio que se implantará en el entorno en producción. Una vez se ha homologado el sistema, se deberá configurar el entorno en producción, que cuenta exactamente con los mismos componentes que el entorno de maqueta, y se tendrá que configurar de la misma manera. Para configurar e instalar nuevas versiones de software, se han desarrollado scripts de instalación que se empaquetarán en RPM.

Finalmente, se ha conseguido la integración de una plataforma de alta disponibilidad con monitorización SNMP en la infraestructura de red del operador móvil.

Agradecimientos

Al final de mi etapa universitaria me gustaría agradecer el apoyo incondicional de mis padres, Celedonio y Rufina, que han estado a mi lado tanto en los malos como en los peores momentos. Sin ellos, muy probablemente, no hubiera podido realizar este proyecto.

También me gustaría agradecer el apoyo de Genaker, que me ha ofrecido la oportunidad de realizar el proyecto de fin de carrera. Me gustaría agradecer a Jordi Guerrero y Miquel Teixidor la confianza de asignarme un proyecto de esta importancia y al equipo de proyecto integrado por Carlos Pérez, Carles Asensio, José Miquel Penadés, César Hernández, Oriol Ribera y Marc Martín, ya que juntos hemos logrado que este proyecto haya tenido éxito.

Por último, un agradecimiento especial a Jéssica Molín, que sufrió el día a día de mi carrera universitaria.

TABLE OF CONTENTS

INTRODUCTION	1
CHAPTER 1. REQUIREMENTS	2
1.1. Context	2
1.1.1. Actors.....	2
1.1.2. Components	3
1.1.3. Hardware description.....	4
1.2. High availability	4
1.2.1. Node high availability.....	4
1.2.2. Application high availability.....	4
1.2.3. Data integrity between nodes in frontend cluster	5
1.2.4. Data integrity between nodes in backend cluster.....	5
1.3. Operations and management	5
1.3.1. Simple Network Management Protocol	5
1.4. Deployment and tests	6
1.4.1. Development, demo and production scenarios	6
1.4.2. Installation process.....	6
1.4.3. Stability tests.....	7
1.4.4. Platform auditory	7
CHAPTER 2. ANALYSIS	8
2.1. High Availability concepts	8
2.1.1. Avoiding the single point of failure.....	8
2.2. HA cluster solutions comparison	9
2.2.1. The Linux-HA project: Heartbeat.....	9
2.2.2. HP Serviceguard for Linux.....	14
2.2.3. Red Hat Cluster Suite	18
2.2.4. HA Cluster software comparison.....	22
2.3. O&M Components	23
2.3.1. SNMP components.....	23
2.3.2. SNMP implementation for Linux.....	25
2.4. Tools to test and monitor the platform	25
2.4.1. Sysstat.....	25
2.4.2. Bash scripts	26
2.4.3. SNMP based monitors.....	26
CHAPTER 3. DESIGN AND TESTS IN DEVELOPMENT SCENARIO	28
3.1. Production environment	28
3.2. Heartbeat	29
3.3. The Frontend HA Cluster	29

3.3.1.	Cluster resources	31
3.3.2.	Constraints.....	31
3.4.	The Backend HA Cluster.....	32
3.4.1.	Cluster resources	33
3.4.2.	Constraints.....	33
3.5.	SNMP daemon.....	34
3.5.1.	Parameters to monitor via SNMP	34
3.5.2.	Extending the snmpd agent.....	35
3.5.3.	SNMP traps	36
3.6.	SNMP scenario.....	37
3.7.	Monitoring the environment with cacti	38
3.8.	Stability test environment.....	39
3.9.	Installation process building blocks	40
3.9.1.	RPM packages	40
3.9.2.	Image of the system	40
3.9.3.	Automated way to fine tune the system	41
3.9.4.	Security level configuration.....	41
3.9.5.	Automated installation of the cluster environment.....	41
3.9.6.	Automated installation of a new software release	42
CHAPTER 4.	IMPLEMENTATION.....	43
4.1.	Installation procedure	43
4.1.1.	Ghost4lin.....	43
4.1.2.	RPM installers.....	43
4.1.3.	RPM creation	44
4.2.	Configuring the clusters	48
4.2.1.	Pre-requisites to configure the cluster	48
4.2.2.	Frontend cluster configuration	48
4.2.3.	Backend cluster configuration	49
4.2.4.	Web application deployment	49
4.3.	SNMP daemon configuration	50
CHAPTER 5.	EVALUATION IN PRODUCTION ENVIRONMENT	51
5.1.	Disc failure	51
5.1.1.	Procedure	51
5.1.2.	Service Check.....	51
5.2.	Cluster node failure	51
5.2.1.	Procedure	52
5.2.2.	Service Check.....	52
5.3.	Application failure	52
5.3.1.	Procedure	52
5.3.2.	Service Check.....	52
5.4.	Connectivity failure	53
5.4.1.	Procedure	53
5.4.2.	Service Check.....	53

CHAPTER 6. CONCLUSIONS.....	54
CHAPTER 7. BIBLIOGRAPHY.....	55

INDEX OF FIGURES

Fig. 1.1 Genaker's application use case	3
Fig. 2.1 Heartbeat GUI	13
Fig. 2.2 Heartbeat GUI toolbar	13
Fig. 2.3 HP Serviceguard for Linux software architecture	14
Fig. 2.4 HP Serviceguard for Linux GUI	18
Fig. 2.5 Cluster administration tool – Cluster configuration tab	21
Fig. 2.6 Cluster administration tool – Cluster management tab	22
Fig. 2.7 SNMP hierarchical structure	24
Fig. 3.1 Production environment scenario	28
Fig. 3.2 Frontend cluster scenario	30
Fig. 3.3 Backend cluster scenario	33
Fig. 3.4 SNMP plugin architecture	36
Fig. 3.5 SNMP actors in development scenario	38

INDEX OF TABLES

Table 2.1 High availability cluster solutions comparison	22
Table 3.1 Interfaces description in frontend cluster	30
Table 3.2 Interfaces description in backend cluster	33
Table 3.3 SNMP traps specifications	36

INTRODUCTION

Genaker is a company that develops PoC technology based software for PC platform. These applications make use of a set of critical services such as licensing and software upgrade service. The server side software is highly critical and must be always available. If the server is not available, then software installed on the client side will not work. Therefore, it is pretty important that the server side platform is based on high availability techniques on both hardware and software layer.

In this project, three main blocks are covered. First of all, the high availability platform design. Then, SNMP management on every server will be studied. Finally, the way that server is installed and updated must be done in a way that it results easy to recover a broken system or install another platform.

This document is structured in five chapters. The first one introduces the project explaining who the actors are and their relationships. Moreover, there are some definitions regarding to node and application high availability. In this chapter, some concepts about SNMP and all environments where the platform will be installed are also presented. Finally, requirements based on all those concepts introduced in this chapter will be written.

In second chapter, there is a detailed analysis based on three different high availability software solutions: Linux-HA Heartbeat, Red Hat Cluster Suite and HP Serviceguard for Linux. There is also some analysis regarding to SNMP protocol and its implementation in Linux operative system. SNMP will be used to retrieve some specific values and monitor the system in development scenario using SNMP based frameworks.

In third chapter, frontend and backend cluster design will be explained. It includes three main aspects: how each cluster must work, which cluster resources must run and which constraints must have. On one hand, the frontend cluster will receive queries from software running in PC on the client side to the different services that it offers. On the other hand, backend cluster will receive queries from frontend cluster nodes. Backend cluster is located in a secured zone which is not reachable from the Internet. Moreover, in this third chapter, parameters that are going to be monitored via SNMP will be listed and the way to extend the MIB will be explained. The SNMP environment used in development scenario and how it is going to be visualized is explained here. Finally, the installation method using RPMs is explained as well as the installation process used.

Lastly, in chapter five a set of test cases are written in order to check that the system is behaving correctly.

In short, this project has enabled the high availability platform implementation with SNMP support.

CHAPTER 1. REQUIREMENTS

In this chapter the project is presented and main building blocks of it are explained.

First of all, context actors and components interacting in the whole project are described. Afterwards, two core components of the project are introduced: high availability concepts and remote monitoring using SNMP.

Finally, requirements for installation and deployment on different scenarios are explained.

1.1. Context

In October 2007, a mobile operator started to offer a new mobile service based on PoC (Push-to-talk over Cellular). This technology allows group voice communication and other next generation services, using a mobile phone in the same way than a walkie-talkie, over a cellular phone network.

The mobile operator also offers to its clients a PC application which was developed by Genaker. This application can help a company to improve and optimize communications with all its PoC lines sold by the mobile operator.

Every software installation needs a license provided by Genaker. In order to control the number of licenses and provide some extra services to the end user, Genaker needs to design and maintain a high available platform integrated within the mobile operator's production environment.

Services offered by this platform are carried by four web applications developed by Genaker running on different tomcat instances on different ports. Moreover, an apache web server will serve static contents.

1.1.1. Actors

There are two actors in this project. Genaker is the software supplier and maintainer and the mobile operator is the distributor of the application developed by Genaker and PoC infrastructure owner.

Genaker was founded in October 2003 in Barcelona as a spin-out of Mobile Software R+D of Nokia. It owns a wide experience in mobile environments developing multimedia SIP based services. The company is a reference in PoC (push-to-talk over cellular) area solutions, and has four years experience developing software based on this technology.

The mobile operator offers residential and enterprise mobile advanced services. It recently started to offer PoC service and distribute software developed by Genaker to companies who need quick coordination between all its lines.

1.1.2. Components

In this subsection, a use case of PoC communication using the PC application developed by Genaker is explained.

All workers in the company have a mobile phone which is connected to mobile operator's PoC infrastructure via GRPS or 3G network. Genaker's PC application is installed on a PC which has connectivity to licensing server and PoC infrastructure. It connects to the licensing server to check if it has a valid license. If the licensing server cannot find a valid license for the application that requested it, then it will not work.

A worker using Genaker's PC application creates a PoC group and sends a group invitation to a subset of mobiles: project manager's, head of carpenters' and head of brick layers'. At this moment, if they accept the invitation, they can start a group voice communication between them with just one click.

Figure 1.1 shows all components working together:

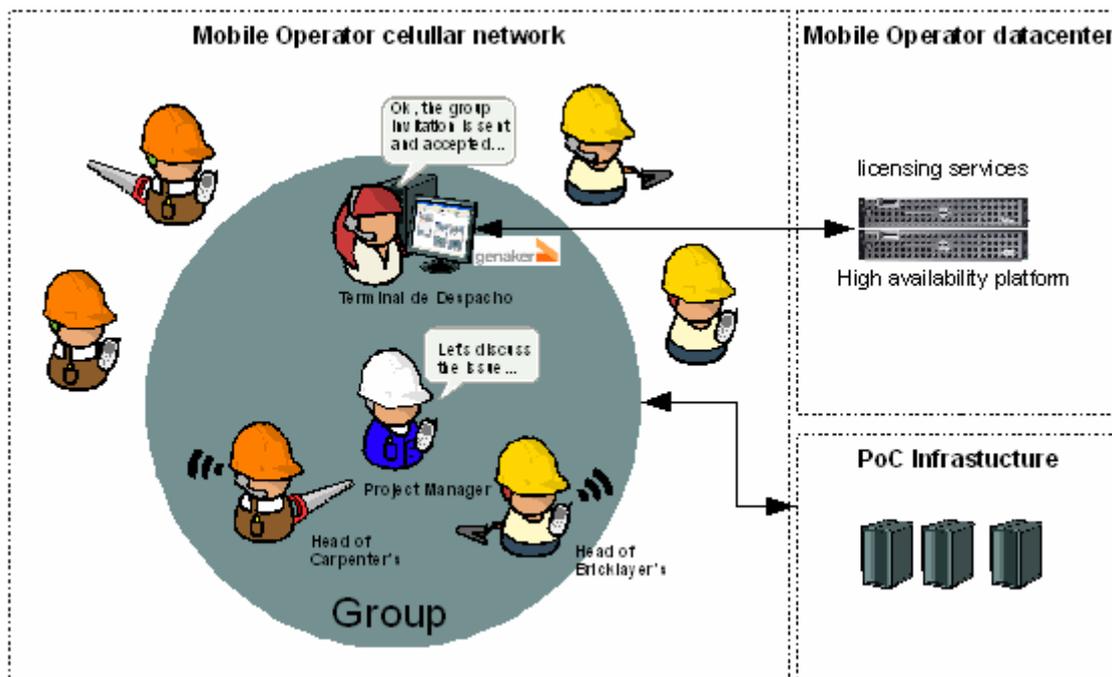


Fig. 1.1 Genaker's application use case

1.1.3. Hardware description

The platform where licensing server application is going to be installed is composed by four Dell Poweredge 2970 [1].

Genaker uses Dell servers to deploy its server side applications, and there have never been any kind of problem with it. Choosing Dell servers as the hardware layer for this project is an election based on Genaker's experience

Main features that these Dell Poweredge 2970 servers have are listed below:

- 2 processors PE2970 Opteron HE 2212 2.0GHz/2M 68W Dual Core
- 16GB DDR2 667MHz Memory - 2CPU (8x2GB dual rank DIMMs)
- 2x 73GB SAS (10,000rpm) 2.5in Hard Drive
- Chassis 2.5 HDD x8 Backplane
- PERC 5/i, RAID Integrated Controller Card (8 ports, 256MB cache, battery backup) x8 backplane
- Intel PRO 1000PT Dual Port Server Adapter, Gigabit NIC, Cu, PCI-E x4
- Broadcom 2Port TCP/IP Offload Engine (TOE) Not Enabled.

1.2. High availability

Services running on Genaker's platform are critical. If license server does not work, no Genaker's PC application installation will work. Therefore, mobile operator and Genaker agreed to design and implement a high availability platform where Genaker services will be running.

1.2.1. Node high availability

In case of node failure, that means, the node is no longer available to serve any query, the cluster must failover resources to the other node.

Node failure can happen in the following circumstances:

- Node loses connectivity to the default gateway. This usually happens because of a network error (nic failure, switch out of service or gateway out of service).
- Node is halted, hardware failure or any unexpected reason makes node be out of service.

1.2.2. Application high availability

When a critical application running as a cluster resource fails, there is no need to move all cluster resources to other node. For instance, if a server is overloaded and a request to a cluster resource fails, restarting the application can solve the problem.

However, this failure is a symptom that there is a problem in the cluster node. The policy will be to restart the application a finite number of times and if it continues failing to requests, then all cluster resources must be failed over to the other node.

In order to measure performance and user experience parameters, every web application will run in a different instance of apache tomcat. Therefore, every web application will be considered as a different cluster resource. This decision leads to an independent monitoring of every cluster resource and if a service fails, there is no need to restart all web applications, but only the sick one.

1.2.3. Data integrity between nodes in frontend cluster

One of the four web applications sends the clients an URL where the client can download a specific content. To serve this static content an apache web server must be installed and configured with client authentication.

To ensure that all nodes have the same information to serve, a hard disk replication method must be considered.

1.2.4. Data integrity between nodes in backend cluster

Both nodes in backend cluster run the database engine and receive queries from application running in the frontend cluster.

MySQL Enterprise server has been chosen to be the database engine. It provides a data replication method between database engines. Both nodes in the backend cluster must be configured to use MySQL replication method [2] to ensure data integrity in cluster.

1.3. Operations and management

The system is going to be installed within the client infrastructure. Operations & Management people will be monitoring hundreds of systems, so our system must support primary and secondary monitoring. That means, that O&M people may query the system status and what it is more important, the system must throw an alert in case of system error.

1.3.1. Simple Network Management Protocol

The SNMP protocol should be configured on every host so that system administrators are aware of the system. Send alarms and answer to SNMP queries will help O&M staff to know system status.

Therefore, A SNMP agent must be installed in every machine in order to monitor some variables within the Linux system.

In the development scenario, where stability tests run, it is necessary to add an extra machine and install network SNMP management software to monitor some values provided by the agent.

Moreover, an interface must be programmed to query certain values. That means that the system will be querying a defined value and comparing it to an expected value. If the expected value does not match the result of the queried value, then an alert must be send to the network SNMP manager.

1.4. Deployment and tests

After tests and configurations have been done in development scenario, it is necessary to provide a solid deployment process to install and configure any node in the cluster. In this subsection, three different scenarios where the cluster must be configured will be explained.

Installation method will be the same for all scenarios and testing the installation and configuration will be only done in development scenario.

1.4.1. Development, demo and production scenarios

There is a need to have three different scenarios in order to follow a strict deployment process.

First of all, development and testing staff must test a new release in the development scenario. It is located at Genaker's offices and it is used by developers to test new builds. It carries both development tasks and stability tests.

When a new release passes all tests, then it is deployed to demo scenario. Demo scenario is located at client's environment for demos. In this scenario first acceptance tests will be done. When the system is certified by the client, then the installation will be done in production scenario.

Production scenario is located at client's environment for production systems. This system must be only modified in case that demo system has passed all acceptance tests. These are the machines that will carry actual service to the end user.

1.4.2. Installation process

In production environment, there must be a way to upgrade the system with minimal service interruption. Moreover, in case that installation fails while

upgrading the system, there should be a way to roll back to the previous version.

It is also necessary to provide an automated installation method. There is not just one scenario where applications can be configured just once, but three scenarios: development, demo and production.

1.4.3. Stability tests

Before deploying a new release to demo scenario, some stability tests must be performed to check if the server software release is stable. Stability tests can also bring out problems which may not happen in normal operation mode, but they can occur in a stress situation.

Every time that a new build is released, some tests must be done in order to certify the quality of the build. If the application does not succeed in tests, then the build is not released and is forwarded back to the development team.

In a first stage, functional tests are done. This is a human procedure as it includes complex test cases which cannot be automated.

In a second stage, the system must be stressed during 5 days and it must behave as expected, that is, no memory leaks, system load within certain values and some other parameters which can show system status.

In short, a way to get these useful control parameters must be implemented.

1.4.4. Platform auditory

In the end of the project, the client performs two kinds of acceptance tests.

The first one is to check if the system accomplishes the security policy that is defined by the client. The second one is to check if the system behaves as it is described in the acceptance tests document explained in chapter 5.

Once the system has passed these tests in production environment, then the project is over.

CHAPTER 2. Analysis

In this chapter, main building blocks of the project are going to be analyzed. This chapter is a study of the main components in the platform. This chapter does not cover the final platform design, but the analysis of some possible solutions to design and implement the platform.

Moreover, not all requirements listed in chapter 1 need an exhaustive study. Some of them do not need any comparison, and the solution chosen to accomplish requirements listed in chapter 1 will be described in chapter 3.

This chapter contains the two important concepts of the project: how to achieve high availability and how implement remote management.

On the first part of the chapter, high availability concepts will be introduced. Afterwards, a comparison between three high availability platform solutions will be explained and compared.

On the second part of the chapter O&M components and SNMP monitoring will be described.

2.1. High Availability concepts

HA cluster solution is the main building block of the platform design. Therefore it is important to make a study of the different options.

Even large servers are exposed to have errors. Hardware failure and systems bad configured are main reasons for a server to become out of service. Therefore, it might not be worthy to buy a large and expensive server, but some commodity servers configured in a cluster.

2.1.1. Avoiding the single point of failure

To protect critical applications from service outage, the whole environment must be designed in order to avoid the single point of failure (SPOF) [3]. There are two building blocks in a system that needs a good design to avoid SPOF. The first one is addressed to hardware layer and the second one to software layer.

Some points that would help avoid SPOF are listed below.

- Redundant network components
- Channel bonding
- Redundant Heartbeats
- Redundant disk storage
- Storage path

- RAID
- Redundant power.

If no redundancy is provided in these components it might lead to a server failure and service outage.

However, hardware redundancy can not solve all failures. A software layer is necessary to identify and control a failure and transfer applications to a healthy server.

2.2. HA cluster solutions comparison

Hardware redundancy is out of the scope of this project. Only software layer will be analyzed in detail. There are multiple software solutions to implement software layer high availability [4]. In next subsections, 3 products are going to be analyzed and compared.

First of all, Heartbeat, from Linux-HA project is going to be described. Then, HP Serviceguard software for Linux will be analyzed. After that, Red Hat Cluster Suite will be presented. Finally, as a summary, a comparison will be done showing weaknesses and strengths of all 3 solutions.

Two main aspects will be covered on every solution: software architecture and manageability.

2.2.1. The Linux-HA project: Heartbeat

Linux-ha project (Linux High Availability project) [5] is an open source suite to build high available clusters. It can run on every known Linux flavor and in some Unices such as FreeBSD and Solaris.

Heartbeat software is the core component of the Linux-HA project and permits building high availability clusters from commodity servers, as it has no hardware dependencies.

The project started in 1998 and is maintained by an open developer's community. The project is lead by some companies like Novell or IBM. 10 years of development, testing and more than 30.000 production clusters are numbers that would reflect project maturity.

2.2.1.1. Introduction to Heartbeat

Heartbeat is the main piece of the Linux-HA project. An instance of Heartbeat must run on every node which periodically sends packets across the network or serial port to other instances of Heartbeat just as a keep alive signal. If no Heartbeat message is received from a node, the node is assumed to be dead

and cluster resources that this node was running fail over another node in the cluster.

Heartbeat can also deal with connectivity loss and several policies can be applied. For instance Heartbeat can be configured to run resources where it has better connectivity. To have better connectivity is a concept that can mean that a node has more ping nodes reachable, or that a node has connectivity to the outside world to the internet. In both cases, Heartbeat can move resources from the node with worse connectivity to the one who has a better connectivity. Ping nodes can be switches, routers, or whatever system that can assure a certain condition of connectivity or lost of it.

2.2.1.2. *Heartbeat cluster styles*

There are two different scopes when working with Heartbeat. Two styles of clusters can be configured. Let us say R1 cluster style and R2 cluster style.

R1 cluster style was the first kind of configurations that could be done with Heartbeat. In this configuration there were some limitations such as:

- Limitation of nodes in the cluster (it can only accept 2 nodes)
- Cannot perform monitor operations of a cluster resource
- There were almost no options to express dependency information

First version of this R2 cluster style was 2.0.0. At the time of writing this document, Heartbeat latest stable version was 2.0.8. Therefore, this version will be the reference of study in this project.

R2 is fully compatible with R1 cluster style configuration. It supports up to 16 nodes cluster and an asynchronous logging daemon was added. Moreover, there were some improvements on message architecture.

With R2 cluster style, or next generation of Heartbeat, these limitations were overridden and software architecture changed. Adding more components and functionalities lead to a more complex system but more complete than the R1 cluster style.

Some new components appeared with R2:

- ClusterInformationBase
- ClusterResourceManager
- Modular PolicyEngine
- LocalResourceManager
- StonithDaemon

In next sections, the whole R2 style cluster software architecture will be covered.

2.2.1.3. *ClusterInformationBase*

ClusterInformationBase (aka CIB) is the component which stores information about the cluster. It is replicated on every node and stores two kind of information:

1. Static information. It includes definitions of cluster nodes, resources, monitor operations and constraints.
2. Dynamic information. CIB stores the current status of the cluster.

CIB is formatted in a XML document and must be built following DTD document included with Heartbeat specifications. Cib.xml files can be red in annex 2.1.

2.2.1.4. *ClusterResourceManager*

The ClusterResourceManager (CRM) component consists on the following components:

- ClusterInformationBase
- ClusterResourceManagerDaemon
- PolicyEngine
- Transitioner
- LocalResourceManager

The ClusterResourceManagerDaemon (CRMD) runs on every node and coordinates the actions of all other cluster resource managers. It exchanges information with the designated coordinator (DC) and it can be seen as a communication interface between DC and subsystem components such as CIB and LocalResourceManager.

The DC is a special instance of CRMD. It is elected with the help of the list of cluster nodes from the ClusterConsensusManager (CCM) and takes on the responsibilities of ResourceAllocation throughout the cluster. Therefore, DC is the one who delivers CIB information to all slave instances of CRMD.

The PolicyEngine (PE) module is the one who performs computation of the next state in the cluster. It is triggered by any event in the cluster, such as a resource changing state, a node leaving or joining, or CRM event such as a CRM parting or leaving.

The PE, takes the input from CIB and runs locally and does not do any network I/O, it only runs an algorithm. It describes the actions and their dependencies necessary to go from the current cluster state to the target cluster state.

When PE returns an action it is passed to the Transitioner. The goal of this component is to make PE's wishes become true. The new state computed by PE is passed to the Transitioner, which communicates with the

LocalResourceManager on every node and informs about the actions decided by PE (start, stop resources).

2.2.1.5. *ClusterConsensusManager*

The ClusterConsensusManager (CCM) is the one who establishes membership in the cluster. This membership layer is in the middle of Heartbeat, which is the messaging and infrastructure layer, and the new CRM which is the resource allocation layer. CCM information includes new additions to the membership, lost nodes and loss of quorum.

2.2.1.6. *Stonith daemon*

R2 cluster style features include stonith capabilities. Stonith is an acronym meaning "*Shot The Other Node In The Head*".

A stonith daemon is running on every node and receives requests from the CRM's TE about what and when to reset.

An example of when stonith daemon action is used is in a split brain situation. The DC will send a message to the Stonith daemon to reset the sick node. This node will reboot and then will join the cluster again in a healthy way.

2.2.1.7. *LocalResourceManager*

Finally, the LocalResourceManager (LRM) has the responsibility for performing operations on resources, by using ResourceAgent scripts to carry out the work. ResourceAgent scripts define a set of start, stop or monitor operations, as directed by the CRM, which are used to operate on a resource instance.

LRM also provides information about resources. It can provide a list of resources that are currently primary and its current state.

2.2.1.8. *Configuration and manageability*

Heartbeat GUI is a user interface to manage Heartbeat clusters. It is delivered in a separate package of Heartbeat core. It is an application developed with python and is able to add, remove nodes or cluster resources.

It is necessary to have connectivity to port 5560/tcp of the cluster node and password for hacluster user running on the cluster nodes.

Heartbeat GUI is executed from a shell and after being authenticated, the main screen of Heartbeat GUI appears. This is how GUI looks like:

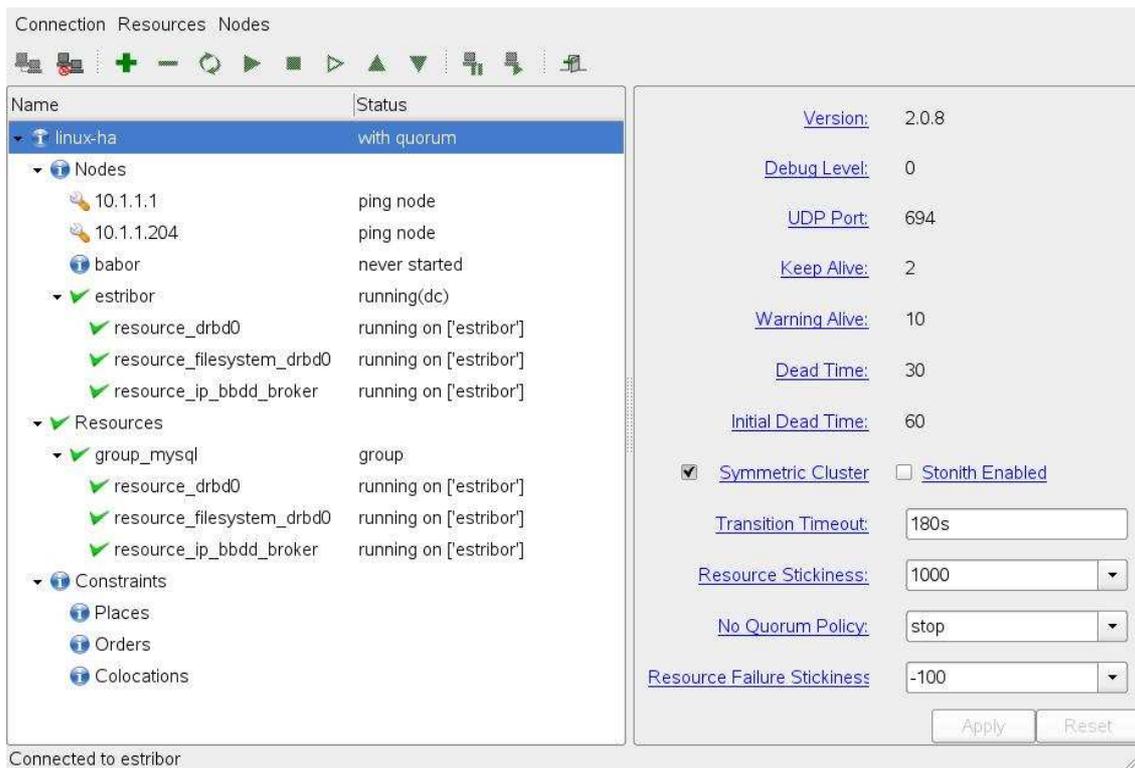


Fig. 2.1 Heartbeat GUI

Figure 2.1 is an example corresponding to a 2 node cluster named: babor and estribor. Babor has never been started, and estribor is the primary node running 3 cluster resources. Some cluster parameters can be configured such as resource stickiness, no quorum policy and many others as can be seen in the picture. Location, order and collocation constraints can also be configured.

On the top of the picture, the following tool bar can be seen:



Fig. 2.2 Heartbeat GUI toolbar

From left to right these buttons perform the following actions:

- Connect to a cluster
- Disconnect from a cluster
- Add a new cluster resource
- Remove an existing cluster resource
- Cleanup resources (reset failcounts)
- Start a cluster resource
- Stop a cluster resource
- Work as default button. It sets not default action to a resource
- Move up a cluster resource in the list. It affects to the order of starting resources
- Move down a cluster resource

- Makes the node standby
- Makes the selected node primary (opposite to the previous one)
- Exit the GUI

However, using the Heartbeat GUI to configure a cluster is not the only way to do it. Heartbeat packages provide command line tools such as cibadmin or crm_resource to get or set cluster or resource configuration. Moreover, cib.xml file can be edited and written, but this is a prone error method.

2.2.2. HP Serviceguard for Linux

Serviceguard for Linux [6] is a high availability clustering software designed by HP to protect mission-critical applications running on Linux from various software and hardware failures. Serviceguard offers availability at different levels. It offers node, application, service and network availability.

A Serviceguard cluster consists on a group of servers, which are named “nodes” and can be blades, servers or partitions. Nodes run packages, which can be started, halted, moved or failed-over. Every package consists on a group of resources. Some resources which can be packaged with Serviceguard are:

- IP addresses
- Volume groups
- File systems
- Network resources
- Application or system processes

In the following picture, software architecture can be seen:

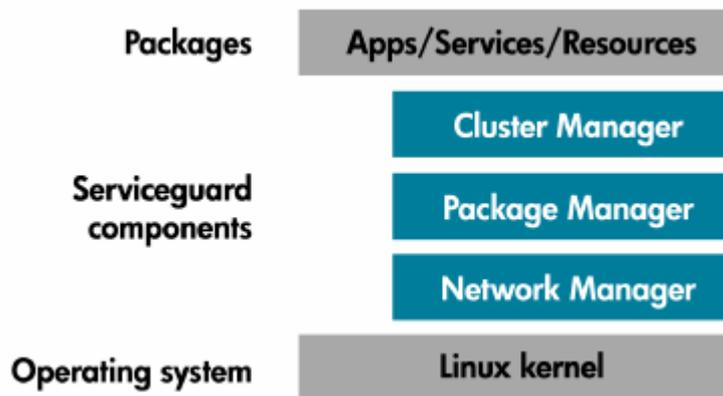


Fig. 2.3 HP Serviceguard for Linux software architecture

A cluster can contain up to 150 packages and every package can contain 30 services.

Every package also contains shell scripts to start, and halt application processes. In short, the package can be seen as the manager of those resources that were managed previously by the system.

Moreover, every package contains a list of prioritized nodes where it can be run. When a single service, node, network or other resource fails, then, Serviceguard automatically transfers control of the package to another node within the cluster. That reaction allows applications and services to have a minimal interruption and minimal downtime.

The architecture of HP Serviceguard for Linux consists of managers or software components that control vital cluster, package, and network functionality.

These are the software components which builds Serviceguard architecture: the cluster manager, package manager and network manager. In next sections all three components will be described.

2.2.2.1. The cluster manager

The cluster manager defines and determines cluster membership. It initializes a cluster, monitors its health, recognizes a node failure, and coordinates the recomposition of the cluster when a node joins or leaves it. The cluster manager runs as a daemon in every machine. However, there is always a coordinator which is the central point of the communication between nodes.

An important role of the cluster manager is to protect against split brain syndrome and protect data integrity. Split brain syndrome can happen if the primary node does not send a Heartbeat message to the other nodes in the cluster. This could happen because of a Heartbeat network channel failure or a primary overloaded node which cannot send Heartbeat messages on time.

Then, the secondary node will take over the packages which were running on the primary node and it would become the primary server. However, if the "old" primary server recovers Heartbeat network channel link or recovers from an overloaded situation, it will still try to act as the primary server which may potentially run two instances of the same application and it may probably lead to data corruption if both nodes acting as primary node try to access to the same data storage.

Another important role of the cluster manager is to exchange Heartbeat messages to the cluster coordinator over the network interface card configured for that purpose.

When a configurable number of Heartbeats are missed the cluster manager must re-form the cluster. After the cluster coordinator and cluster membership are established, information about the new set of nodes within the cluster is passed to the package coordinator. Packages running on nodes that are no longer in the new cluster are transferred to their adoptive nodes.

When a server does not respond to Heartbeat messages because it is temporally hung, the cluster can re-form without the server. However, the server may recover and continue with its normal operation and might cause data corruption. A deadman timer must be configured in order to avoid this situation. If a node does not respond to a Heartbeat message before deadman timer expires, then it will be reset as soon as it recovers.

2.2.2.2. *Package manager*

Every node on the cluster runs an instance of the package manager. The package manager that runs in the same node than the cluster coordinator is called package coordinator. The package coordinator decides when and where to run, halt, or move packages. Package manager is aware of resource status, and it determines what to do in case of status change on monitored resources.

In the package configuration process there are 3 aspects which have to be defined:

- list of services which runs in the package
- prioritized list of cluster nodes where the package may run
- type of failovers allowed for the package

Package failover is decided by the package coordinator. It involves halting the package on the node where it is running, and starting a new instance on the new node.

Serviceguard allows creating packages with parameters to create dependencies, perform load balancing of system resources or establish an order for starting/halting resources. These parameters can be used to define a failure policy, which means what to do when a node fails. If the primary node recovers from the failure, it has two options: take over the resources and become the primary node again, or just keep standing by.

There are many configurations which can be done. These, includes two traditional configurations. The first one is the primary/primary configuration, where all nodes in the cluster can run different packages at the same time. The second one is the primary/secondary configuration, where only one node is running packages, while the rest of nodes are backup nodes just in case the primary one fails.

All actions that can be performed by the package manager are defined in the package control script. In this script there is information about starting, halting, monitoring resources and reacting to a failure.

Whereas cluster manager monitors node availability, package manager must control package level interruption. If the package manager is configured with Serviceguard Manager GUI, deployment of the control script to all nodes is automated and an integrity check is performed on every package.

2.2.2.3. *Network manager*

Network availability is achieved by using Linux kernel bonding. Network manager gets network interface cards information and detects network failure. In case of failure, the network manager migrates all resources to another node.

One interesting thing about the network manager is the ability to perform load balancing. Different services can be configured in different packages. If the node gets overloaded, network manager can shift a package to another less burdened node.

An important feature of Serviceguard solution from HP is the Serviceguard Manager. In next section some features from this tool will be explained.

2.2.2.4. *Serviceguard manager*

Serviceguard Manager is a graphical user interface that provides configuration, monitoring, and administration of Serviceguard. It permits the system administrator to monitor clusters, nodes and packages built upon Serviceguard solution from HP.

The GUI can be run on HP-UX, Windows and Linux operating systems and provides several interesting features. Some of them are listed below:

- Cluster, node and package configuration (creation, modification, and deletion)
- Run, halt or move a package, including moving a package via drag and drop
- Run or halt a node or a cluster
- Monitor multiple cluster operation
- SNMP event browser displays all SNMP traps generated by Serviceguard nodes being monitored (this feature is only available on HP-UX operating system)

The picture below shows how Serviceguard Manager looks like:

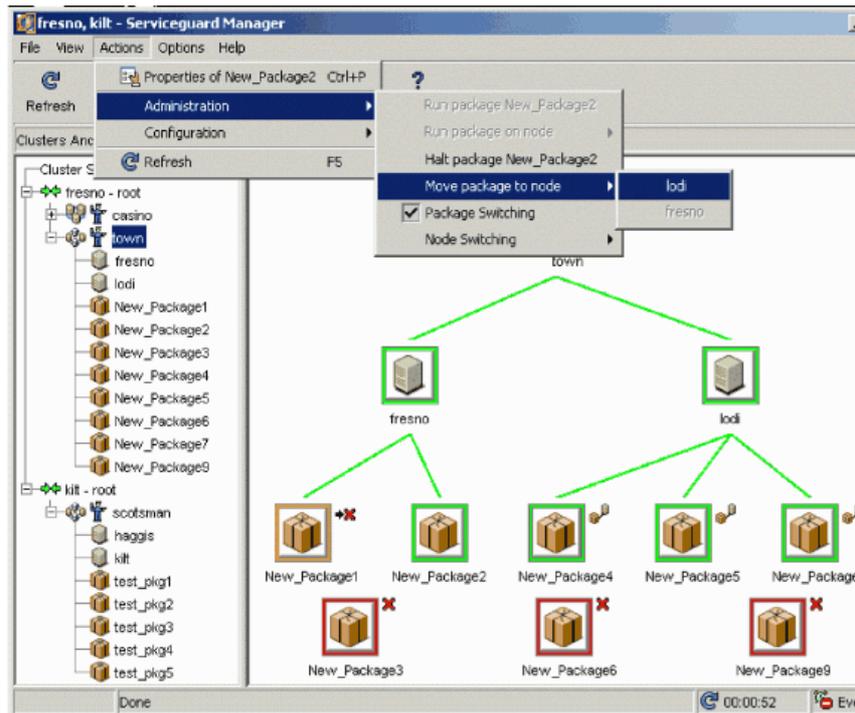


Fig. 2.4 HP Serviceguard for Linux GUI

2.2.2.5. *Extra features*

One of the main benefits of working with Serviceguard is the amazing support that you get from HP. HP provides a complete portfolio that includes online certification courses, access to documentation and consulting from their support engineers who have a vast experience in this product.

2.2.3. Red Hat Cluster Suite

Red Hat Cluster Suite (RHCS) [7] is a software solution provided by Red Hat Inc. to build high available clusters. RHCS is made of software components that can be deployed in different configurations to suit the needs of every scenario. It can be used to also build load balancing and performance clusters.

RHCS includes these components:

- Cluster infrastructure
- High Availability Service Management
- Cluster administration tools
- Linux Virtual Server (LVS)

2.2.3.1. *Cluster infrastructure*

That component provides some functions and operations to a set of Linux servers to act as a cluster. It performs cluster membership management, lock management, configuration file management and fencing.

RHCS offers two components which performs cluster management: CMAN (Cluster Manager) and GULM (Grant Unified Lock Manager). CMAN is a distributed cluster manager while GULM uses client-server architecture.

In one hand, if the cluster is configured to use CMAN, every cluster node runs a CMAN instance and cluster management is distributed across all nodes within the cluster.

On the other hand, if the cluster is configured to use GULM, then it will only run in those designated GULM server nodes.

The cluster manager keeps a track of cluster quorum. If more than 50% of nodes that run the cluster manager are primary, then the cluster has quorum. Otherwise, the cluster has not quorum and all activity is stopped. Cluster quorum is necessary to avoid split brain condition (two instances of the same cluster are running at the same time). In this situation, both instances might access to cluster resources at the same time which may cause corrupter cluster integrity.

In a CMAN approach, quorum is determined by exchanging Heartbeat messages. Heartbeat messages are exchanged using Ethernet devices or RS232 serial link. If 50% of node votes + 1 are reached, then there is quorum. By default, every node has one vote but the administrator can configure every node to have more than one vote.

In a GULM approach, quorum is determined when there is a majority of GULM nodes according to the number of nodes.

Beside, cluster infrastructure includes fencing capabilities which is the disconnection of a cluster node from the cluster. The fenced node has no access to cluster resources and I/O to shared storage is blocked. Fencing can happen when a node is misbehaving in the cluster due to a failure in the node or any other reason. Fencing avoids data corruption and it is performed by CMAN or GULM.

With CMAN, fence daemon performs fencing, while with GULM severs, GULM does. RHCS provides several ways to fence a node:

- Power fencing. If there is a power controller which can power off nodes.
- Fiber Channel switch fencing. This method disables fiber channel port that connects to the inoperable node.
- Other fencing. They are usually provided by server vendor. An example is DRAC/MC cards from Dell or HP ILO.

The Cluster Configuration System (CCS) is the component who manages cluster configuration and provides configuration information to other cluster components in a Red Hat cluster.

An instance of CCS runs on every cluster node. If the system administrator modifies configuration on one node, CCS will deploy the changes on every cluster node's configuration file.

The cluster configuration file is a XML document where some features are set. Some of them are cluster name, fencing method, information about nodes in the cluster (node name, node ID, fencing method for every node, quorum voting value), fence device, managed resources and definition of failover domains.

2.2.3.2. *High Availability Service Management*

The component which gives high availability functionalities to a Red Hat Cluster is rgmanager. A set of services or processes configured with other cluster resources within a Red Hat cluster are called cluster services. High availability cluster services can fail from one node to another without service interruption. The end user would not notice that the application running in the clustered server has changed from one node to another.

There are some things that need to be configured in the configuration file to achieve a high availability cluster service.

First of all, the cluster service, which is one or more cluster resources, must be defined. A cluster resource can be for instance an IP address, a service or a partition. Then, this cluster service can be associated to a failover domain. A failover domain is a subset of cluster nodes where the cluster resource can be failed over. Moreover, some priority nodes can be defined. That means that the system administrator can have a preferred order of failing over the list of nodes. If no failover domain is specified, then the cluster service can fail over to any node within the cluster.

Another important thing to know about cluster services is that only one instance can run on the cluster, that is, there cannot be 2 cluster nodes running the same cluster service.

2.2.3.3. *Cluster administration tools*

This component offers tools to configure, manage and setting up a Red Hat cluster. This set of tools is made to manage easily all components of RHCS.

First of all, Conga is a set of software components that provides centralized configuration and management of the cluster. Conga provides features such as:

- Web interface for managing the cluster
- Automated deployment of cluster data

- Integration of cluster status and logs

Two primary components in Conga are luci and ricci. Luci is the server part of the communication. It runs on one computer and can communicate with each cluster via ricci. Ricci is the client part of the communication. It is an agent running on every node and communicates with luci.

Luci can be accessed through a Web browser and can provide three main features: provisioning (adding, deleting nodes or users), cluster configuration and storage administration. Moreover, a system administrator can configure user permissions to allow, for example, an operator to configure cluster A, but not cluster B.

Another important tool to manage Red Hat clusters is the Cluster administration GUI. This GUI provides two major features:

- Cluster Configuration Tool
- Cluster Status Tool

The first tab in the GUI shown in figure 2.5 corresponds to the cluster configuration tool. It allows creating, editing and propagating the cluster configuration file. Cluster nodes, fence devices, cluster services, cluster resources and resources can be configured.

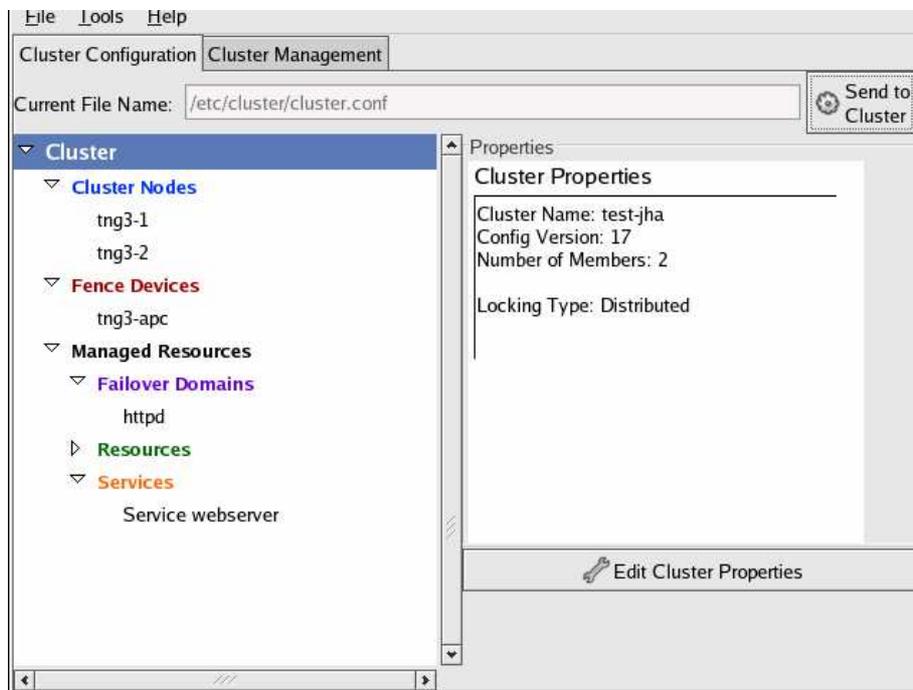


Fig. 2.5 Cluster administration tool – Cluster configuration tab

The second tab corresponds to cluster status tool in the Cluster Administration GUI and is shown in figure 2.6.



Fig. 2.6 Cluster administration tool – Cluster management tab

Cluster management tool allows the system administrator to enable, disable and restart or relocate a cluster service. Cluster nodes shown in the GUI are those who were set in the configuration file.

2.2.3.4. Linux Virtual Server (LVS)

This component provides IP load balancing using routing software. LVS runs on a couple of redundant servers which routes request to real servers that run behind them. Load balancing techniques is almost mandatory if the platform must be scalable. It is a good feature for the future, but it is out of the scope of this project.

2.2.4. HA Cluster software comparison

Among these three solutions discussed previously, only one will be used to build the high availability platform.

In table 2.1, all three candidates will be compared:

Table 2.1 High availability cluster solutions comparison

	HP Serviceguard for Linux	Red Hat Cluster Suite	Linux-HA Heartbeat
Supported platforms	Red Hat Enterprise Linux and SUSE SLES	Linux	Linux, FreeBSD, OpenBSD, Solaris and Mac OS X
Type of clusters	failover	Application	failover

		failover and load balancing	
Load Balancing support	No		
Community	150000 licenses	N/A	> 30000 clusters
Number of nodes	16	16	> 16 nodes
Shared storage solutions	HP shared storage and Red hat GFS	GFS/DRBD	DRBD
Upgrades	Yes (subscription needed)	Yes (subscription needed)	Yes
Price	\$1309 each node / 1 year	\$499 each node / 1 year	\$0
Support price	> 499 / year	> \$1499 (depends on RHEL subscription)	\$0
Support access	Telephone / HP documentation center	Telephone and Red hat network access	Mailing list
Extra features	Great support, documentation, training and certification courses	Integrated in new versions of RHEL 5	Simple and full of features. Development and usage is growing exponentially

2.3. O&M Components

Among all requirements listed in chapter 1 only one topic is covered in this chapter. Although it is not a service level component, O&M features are very important in a telecommunications environment.

In this section, a study of SNMP as management protocol is. Some aspects such as actors in a SNMP scenario or what kind of information can be obtained using SNMP queries will be presented in this section.

2.3.1. SNMP components

SNMP is a protocol included in the TCP/IP protocol suite [8]. Network administrators can monitor or configure a device remotely. A set of network management stations and network elements defines the protocol architecture.

Network management stations run management applications which monitor and control network elements such as routers, switches, servers, or any device who can act as a managed device [9].

2.3.1.1. The SNMP agent

The agent is the software which runs in the managed device. It provides an interface to all the items of their configuration. These items are stored in a data structure called a management information base (MIB).

It is the server side, as long as it maintains the information being managed and waits for commands from the client.

2.3.1.2. The network management system (NMS)

The manager is the software that runs in the monitoring station of the network, and its role is contacting the different agents running in the network to poll for values of its internal data. It is the client side of the communication.

2.3.1.3. SNMP MIB

A Management Information Base (MIB) is a set of information which is organized hierarchically. Only leaves on the tree do have an associated value with it. These leaves identifier are formally called SNMP OIDs.

As an example, next picture shows the hierarchical structure of the MIB tree for Cisco Systems.

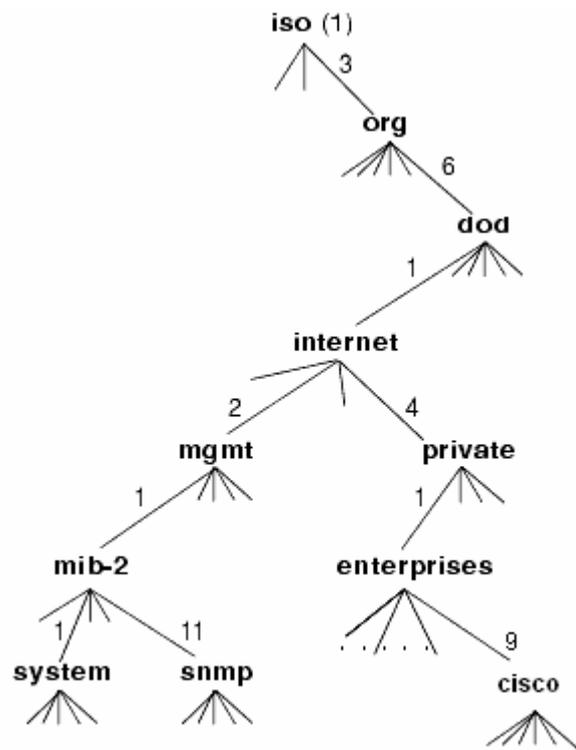


Fig. 2.7 SNMP hierarchical structure

An OID for Cisco sub tree would always start with the sequence: .1.3.6.1.4.1.9.X. where X would be different for every sub tree.

2.3.1.4. *SNMP notifications*

Notifications in SNMPv2 can be defined as v2traps messages or information messages. V2traps messages are not confirmed, while information messages are confirmed.

SNMP notifications are sent to port 162/UDP to the IP address of the SNMP manager. They are events that are notified to the manager using SNMP protocol.

2.3.2. **SNMP implementation for Linux**

The SNMP protocol is implemented in Linux via net-snmp project [11]. Net-snmp packages include a SNMP agent, a SNMP manager, a trap receiver and other utilities to build a SNMP management environment.

Net-snmp project history began on 1990 at Carnegie-Mellon University by a network group at CMU (led by Steve Waldbusser). The source code was made publicly available and several system administrators joined the project. In 1995 UCD-SNMP project was born. Eventually in late 2000, the project changed its name to Net-SNMP.

Net-SNMP project includes, after almost 20 years, an open source suite of applications which implement SNMP v1, SNMP v2c and SNMP v3.

Net-snmp Project is mature enough to be used on a real production environment providing both secure and stable SNMP management.

2.4. **Tools to test and monitor the platform**

In order to implement a stability test environment some aspects are studied. First of all, tools to monitor the system must be chosen. In next sections some solutions are presented.

The most important features of these solutions will be the way that data is collected and how data is displayed to the user.

2.4.1. **Sysstat**

Sysstat provides an easy way to get system status. It can provide CPU usage, memory usage, I/O writes and so on.

It provides a command line interface to get system information and can be configured to be run by cron daemon. Every machine is configured to run sysstat every minute and write data to its binary log file. This is a requirement in the POS auditory.

2.4.2. Bash scripts

Bash scripts can be programmed to perform some queries to “ps” process or top process and look into proc file system in order to get system status. This is a rudimentary way of obtaining system health, but it really works.

Nevertheless, programming these scripts can be quite time consuming. Reinvent the wheel is not a good solution, and some other solutions should be considered.

2.4.3. SNMP based monitors

There are lots of frameworks to graph data collected via SNMP. A framework that allows for easy configuration, web, and email notification, and incorporates other traditional monitoring methods, is vital.

In next section SNMP based frameworks for system monitoring are going to be presented. In chapter 3, the design of the monitoring framework will be explained.

2.4.3.1. Multi Router Traffic Grapher

The Multi-Router Traffic Grapher [12] is a graphing tool that will create web pages showing hourly, daily, weekly and yearly in-bound and out-bound packet traffic on one or many routers/ports.

However, this framework can be also configured to monitor any data accessible using SNMP.

It is easy to configure, and it generates images where collected data is represented.

Moreover, there are rich environments developed over mrtg core which presents information in a featured web site. Mrtg performs data collection and graphing task while these featured environments organize these graphs.

A cgi called routers2.cgi [13] is frequently used to display mrtg generated graphs.

2.4.3.2. Nagios

Nagios [14] is a program widely used to monitor services and hosts. It provides a web monitoring environment that shows host or service availability. Nagios core makes use of nagios plugin. A nagios plugin is an executable which performs the actual check.

A very interesting plugin is the check_snmp plugin. By means of this plugin, nagios can launch a SNMP query to a specific OID on a configured host. Therefore, nagios is aware of any data accessible via SNMP.

However, nagios does not provide a web interface to configure checks. Though there are several 3rd party web-based tools to do the work easier, it is a quite complex tool to administer for a snmp monitoring test environment.

2.4.3.3. Cacti

Cacti [15] is another SNMP based environment. It is quite similar to mrtg + routers2.cgi approach. However, it provides a backend to configure data queries, graphing methods and menu layout and is totally configurable using the backend web interface.

A powerful feature of cacti is that templates can be configured. On an environment where multiple similar machines are involved, developing a template can save lot of time.

Features in cacti include rich display of graphs including both menu creation for grouping data in a hierarchical way and usable zoom based on a mouse selection on the graph.

Moreover, adding, modifying and deleting monitored items is extremely easy with this environment.

This network architecture will determine how many interfaces will be used, and the kind and amount of traffic generated by each one.

Network interface cards are shown in orange bubbles with a number in it. Static IP addresses are in white boxes, while virtual cluster IP addresses are shown in yellow boxes.

3.2. Heartbeat

The first approach in the design and implementation of the environment was a Heartbeat R1 style cluster. At that moment, it was enough to build a high availability platform with IP failover. This version did not perform any status operations on cluster resources so there was a need to configure a Heartbeat R2 style cluster.

During the project, a new requirement was added. Those nodes that are reachable from the Internet cannot store any user confidential data, which means that the database engine must be installed on a secured zone which must not be accessible from the Internet.

Solution to this change request implied adding two extra machines in the environment which would also have to be configured in a high availability cluster.

At the time of designing the system, the stable Heartbeat release was 2.0.8-2. There were some newer releases, but they were too recent to be used in a production environment. Bugs solved in these newer releases were not critical and experience shows that testing or latest releases are not always the best choice for a production environment.

3.3. The Frontend HA Cluster

The role of the frontend cluster is to receive queries from the PC application running on the client side. The client will make the request to an IP address that might be owned by any cluster node. Clients do not need to know which the primary server is. Heartbeat will care about setting that IP address on the proper cluster node.

Where data integrity is concerned, the safest and most expensive way to achieve data replication is having an independent storage server where all nodes in the cluster would read and write at the same time. The lack of hardware to implement this solution leads to looking for other implementations such as DRBD [16].

DRBD is a kind of standby data replication. DRBD is a block device which is mounted only in the primary node. However, data is replicated over a network

link to the secondary node. In this way, if primary node fails, secondary node will have consistent data and will be able to take on service correctly. DRBD concept can be understood thinking in a “network-RAID1” device.

Figure 3.2 shows frontend cluster network topology:

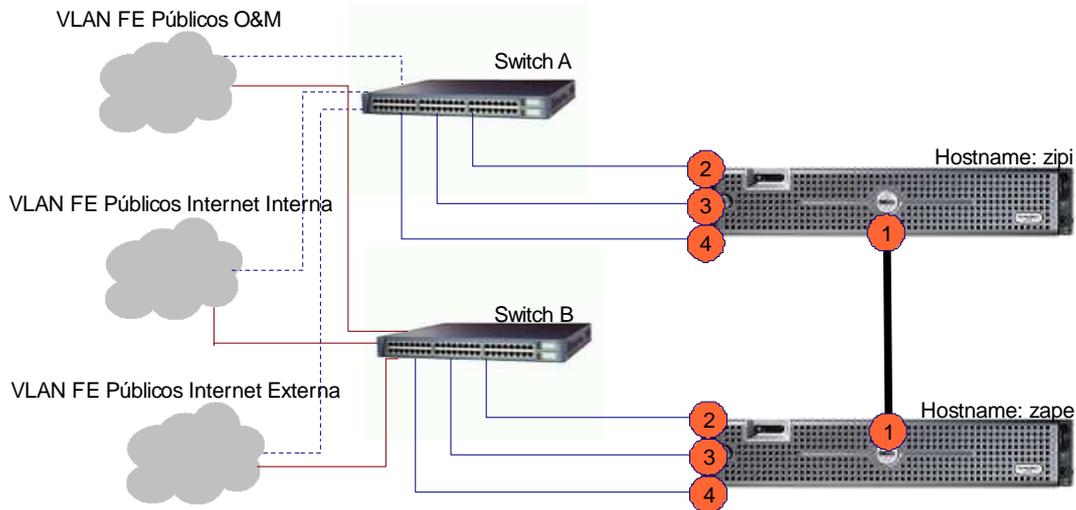


Fig. 3.2 Frontend cluster scenario

Two hosts named zipi and zape compose the high availability cluster. Host zipi is connected to switch “A” whereas host zape is connected to switch “B”. Both switches are configured to connect corresponding interfaces to 3 different VLAN. By using two switches, system availability increases as there are redundant paths. If both servers were connected to a unique switch, it would be a single point of failure and could cause service outage in case of switch failure.

Network interface cards are these orange numbered bubbles. In table 3.1, the role of every interface will be explained: Interfaces eth0, eth5 and eth6 are not currently in use, but they might be in the future.

Table 3.1 Interfaces description in frontend cluster

Interface	Name	VLAN	Description
1	Eth1	-	Exchange of Heartbeat messages
			DRBD replication data
2	Eth2	VLAN FE Públicos O&M	SNMP traffic
			SSH traffic
			Exchange of Heartbeat messages
			NTP traffic
3	Eth3	VLAN FE Públicos Internet Interna	MySQL queries to backend cluster
			Interconnection between servers
4	Eth4	VLAN FE Públicos Internet Externa	Incoming traffic from the Internet

Host zipi will take the primary role, while host zape will wait until zipi fails. If zipi fails, then zape will be the primary node and zipi will wait until zape fails.

3.3.1. Cluster resources

To accomplish requirements described above, following resources must be highly available. Both nodes in the cluster might run them, so they must be considered as cluster resources.

These resources are:

- DRBD manager. Triggers DRBD to make the current node primary for the given resource.
- DRBD file system. This resource actually mounts the file system
- IP addresses.
 - IP address at VLAN FE Públicos Internet Externa. Servlet engines will bind to this IP address and will listen to client requests.
 - IP address at VLAN FE Públicos O&M. O&M people will query one servlet engine to enter to the management website.
- Servlet Engines. There will be 4 tomcat installations. Every tomcat instance will run a different web application at different ports.
- HTTP server. The apache HTTP server will serve the static contents and will limit the number of connections and will control the bandwidth on every connection.

3.3.2. Constraints

There are some constraints that Heartbeat must take into account. In next sections, these constraints will be explained.

3.3.2.1. Connectivity constraint

Heartbeat introduces the concept of ping node as a node which must be always reachable. If a cluster node cannot ping a defined host, then it is considered as a node which has not connectivity to the world. Therefore, a connectivity constraint is included in the cluster design. The primary cluster node must have connectivity to the world.

More than one ping node can be specified. It is highly recommended to configure a reliable and highly available machine (e.g. the default gateway) as the ping node

If the secondary node does not have either connectivity to its ping node, then it will not run any resources. The cluster will be out of service until one of the nodes will recover its connectivity to it. However, there must be 2 failures in the

whole system for this situation to happen. The system was configured to avoid single point of failure, but not more than one.

Connectivity constraints will avoid a service outage in case there is a cable, nic or switch failure. Moreover, it is recommended to have different switches for every node to avoid the single point of failure.

3.3.2.2. *Location constraint*

All cluster resources must run in the same node. Only one IP address is used to accept incoming connections to the cluster. Therefore, the cluster node that owns this IP address must run all servlet engines and apache web server.

Moreover, the primary node, which owns servlet engines resources, must have access to consistent data storage. That means that DRBD partition resource must be located in the same node than servlet engines.

This kind of behavior is implemented in Heartbeat putting all cluster resources within a group. A group is a set of cluster resources that belongs to the same group resource. In case of failure of a cluster resource belonging to a group, the whole set of resources will fail over to the other node.

3.4. **The Backend HA Cluster**

The backend servers will store user confidential data. They are located in a secured zone and must be only accessible by frontend servers.

If the database engine was not configured in a high availability cluster, it would be considered as a single point of failure. To avoid single point of failure, two servers will be needed to provide high availability.

Moreover, databases in both nodes must be consistent and every change on the primary node database must be written in the secondary node database. This requirement is accomplished using built-in replication capabilities of MySQL server database engine.

Figure 3.3 shows frontend cluster network topology:

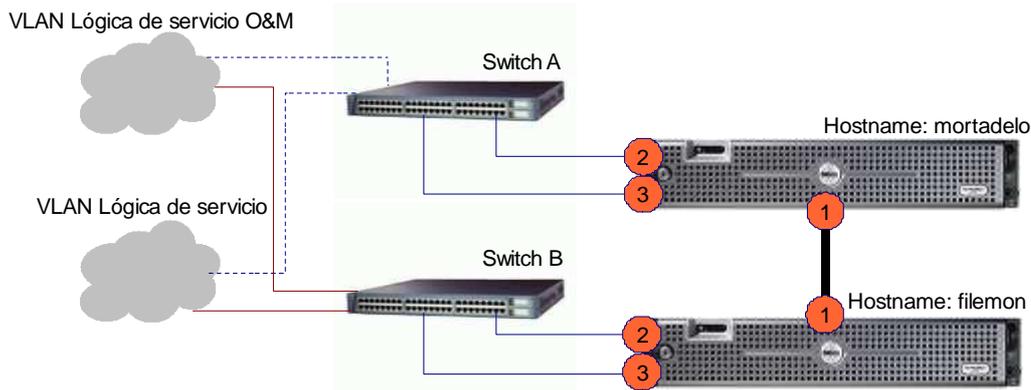


Fig. 3.3 Backend cluster scenario

In table 3.2, the role of every interface will be explained. Interfaces eth0, eth4, eth5 and eth6 are not currently in use, but they might be in the future.

Table 3.2 Interfaces description in backend cluster

Interface	Name	VLAN	Description
1	Eth1	-	Exchange of Heartbeat messages
2	Eth2	VLAN FE Públicos O&M	Exchange of Heartbeat messages
			SSH traffic
			SNMP traffic
			NTP traffic
3	Eth3	VLAN FE Públicos Internet Interna	MySQL responses to frontend cluster
			Interconnection between servers

3.4.1. Cluster resources

There are not many resources in backend cluster. In fact, this cluster only provides an IP address resource to listen to MySQL queries made by the primary node of the frontend cluster.

MySQL server is running on each cluster node so that replication can be performed. Both are configured to be master and slaves at the same time. In fact, from the functional scope, the master database server will be the one who is running the IP address cluster resource.

3.4.2. Constraints

There is only one constraint in this cluster. It must have connectivity to the default gateway. In the same way than the frontend cluster, if a node loses connectivity to its default gateway (the defined ping node), resources are failed

over the other node. If the other node has neither connectivity, then cluster resources cannot run anywhere.

3.5. SNMP daemon

Linux implementation for SNMP is achieved using net-snmp package. As far as I know, there are no other implementations for this protocol.

Two different aspects must be configured on every host:

- The agent, both primary and secondary monitoring.
- The primary monitoring / trap sending

The first step is to define a list of parameters which are going to be monitored via SNMP. Some of them can be queried using MIBs provided by the net-snmp package. However, there are some of them that cannot be retrieved by querying the default MIBs. Here the design of a new MIB module is presented. It includes both the MIB definition in ASN.1 and MIB module extension capabilities offered by snmpd.

3.5.1. Parameters to monitor via SNMP

In this subsection a list of parameters to monitor via SNMP will be listed [17]. Among all values which can be retrieved using SNMP, these ones are the most significant [18]:

- CPU Usage. Percentage of CPU used by a system, nice or user processes.
- System Load Average
 - In the last 1 minutes
 - In the last 5 minutes
 - In the last 15 minutes
- System Memory Usage. Memory used by the system.
- Free Memory. Available physical memory.
- Swap Memory. Amount of memory which is written to physical disk.
- Available disk space.
- Available space in a disk partition.
- Interface statistics. Incoming or outgoing traffic, administrative and operative status.

However there is a list of desired values which cannot be retrieved with the default installation of net-snmp. These values are specific values about applications running on the servlet engines or cluster and system status.

Linux-HA project provides a basic SNMP support in v1 cluster style. This SNMP support is made by using agentx capabilities of snmpd. However, there is not any support for v2 cluster style at the moment of writing this document.

The need of the desired values which cannot offer snmpd and the lack of SNMP support in the component that provides the high availability make necessary to implement an agent extension that may provide this data.

3.5.2. Extending the snmpd agent

Extending the net-snmpd agent can be done in three different ways:

- Using *ucdavis.extTable.extEntry*.
- Writing a plugin which is loaded to the snmpd agent in the enterprise tree
- Writing a MIB module to serve information described by an SNMP MIB and compiling it with net-snmp source code.

The first approach is useful if a few values want to be retrieved via SNMP. It basically maps a SNMP OID to an executable file which returns a value. When the OID is queried, then snmpd calls for the execution of the script and returns the value using SNMP protocol. However, if the enterprise tree is going to be used, the other ways are preferred.

The second option is to write a dynamically loadable object or plugin. This plugin is compiled and loaded using dlmod token. Then, snmpd will answer to queries offered by this plugin.

The third option means to get the net-snmp source code and compile the agent with the source code of a new MIB module. This recent snmpd daemon would support queries to the objects defined on this new MIB.

The preferred option is to write a loadable object which can be compiled once and copied everywhere without the need to recompile the agent again. In this way, tests could be done to the development scenario and the deploy to demo a production environment would consist on copying just the file and configuring the agent.

An interesting option is to write a plugin that reads a configuration file where there is a mapping between a numeric OID and script with one parameter. In this way, the number of OID could increase and moreover there would not be the need to recompile the plugin when a change in the way that values are retrieved is done (see also annex 3).

Figure 3.4 shows how this plugin would work with snmpd daemon

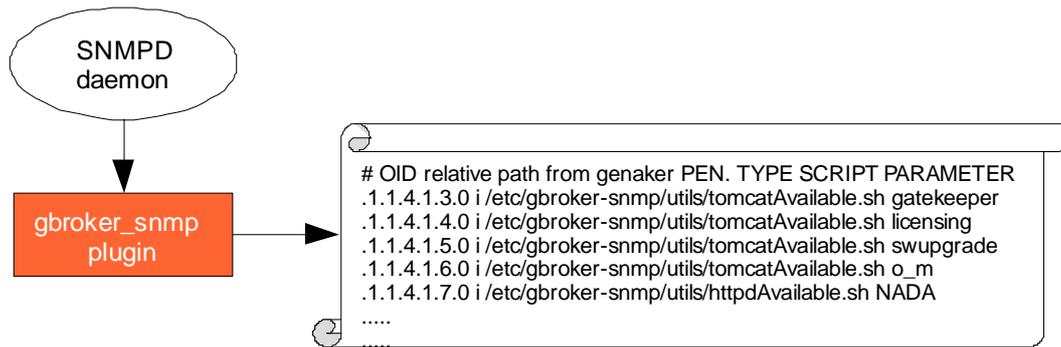


Fig. 3.4 SNMP plugin architecture

When the manager performs a `snmpwalk` or a `snmpget` to the enterprise tree, then the `snmpd` daemon will execute the corresponding script. This script will return a value that will be sent back by the `snmpd` agent to the manager using SNMP.

3.5.3. SNMP traps

SNMP traps are used to trigger a warning or an error of the system. `Net-snmp` provides default monitors which are triggered when a certain event is given.

`Snmpd` daemon monitors certain values periodically and compares the returned values to a configured condition. If this condition is equal to the returned value, then a trap is sent.

These primarily monitored values include percentage of available space in a partition, the use of swap memory or a process error.

These kinds of traps are only sent on the rising event. It means that it is only sent once until the condition that triggered the trap happens again. For instance, if a trap is sent because the disk is over 80% it will not be sent again until the disk space go below 80% and gets over 80% again.

However, traps offered by `snmpd` are not enough. There are system events that should be notified to O&M staff. Table 3.3 presents a list of extra traps that should be implemented:

Table 3.3 SNMP traps specifications

Trap name	Event that fires the trap
<code>connectivityLost</code>	The system has lost connectivity to one of the servers that it is connected to
<code>connectivityReady</code>	The system is fully connected to the other servers.
<code>powerSupplyRedundancyLost</code>	One of the power supplies is broken. Power redundancy is lost.
<code>failCountWarnLevel</code>	Power supply redundancy recovered. The

	power supply is fully redundant
failCountCriticalLevel	The failcount has reached 3. The node will stop running resources when the failcount reaches 5
failCountReset	The failcount has reached 4. The node will stop running resources when the failcount reaches 5
nodeOutOfService	The node is not available
nodeReady	The node is ready and is a member of the cluster
backupError	Backup process was not successful
licensePackUnderZero	Enterprise trap
backendReachable	Frontend has connectivity to backend
backendUnreachable	Frontend lost connectivity to backend
clusterTakeover	The primary node has failed over all resources to the secondary node
gatekeeperStarted	Gatekeeper module started
gatekeeperStopped	Gatekeeper module stopped
licensingStarted	Licensing module started
licensingStoped	Licensing module stopped
licensingdemosStarted	Licensing for demos module started
licensingdemosStopped	Licensing for demos module stopped
omStarted	O&M module started
omStopped	O&M module stopped
swupgradeStarted	Software upgrade module started
swupgradeStopped	Swupgrade module stopped

3.6. SNMP scenario

Development environment consists on the same architecture than production environment. From application point of view, four machines are all that is needed to develop and test some services. However, SNMP scenario must be integrated with this development platform.

Figure 3.5 shows SNMP actors in development scenario:

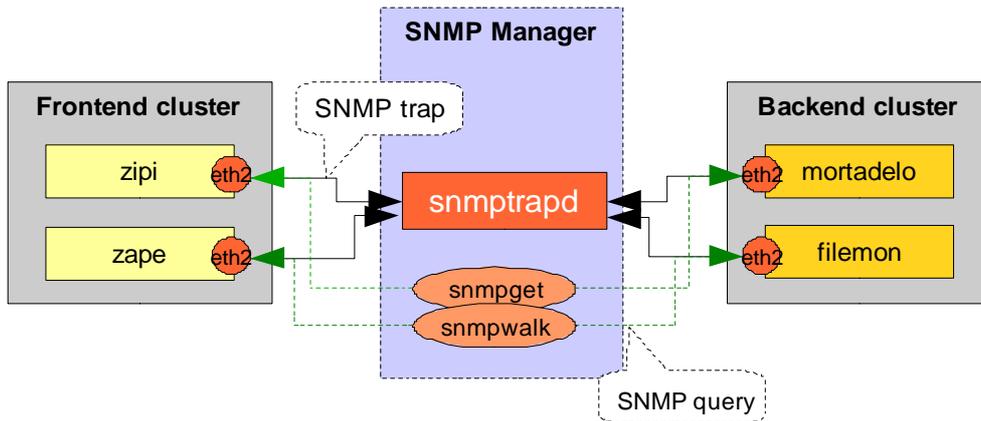


Fig. 3.5 SNMP actors in development scenario

The manager has been configured on a machine outside the cluster. It has connectivity to the O&M VLANs to 4 cluster nodes.

Net-snmp package provides utilities to make SNMP queries. Snmpwalk retrieves all OID values from the queried subtree while snmpget queries a single OID.

Moreover, net-snmp package provides a daemon that listens to traps sent by the other machines. This trap receiver is called snmptrapd. It can forward the trap to another machine, send an e-mail or execute a program on a given event.

3.7. Monitoring the environment with cacti

In this section cacti is going to be presented as the monitoring tool for the whole environment. After having tested all frameworks described in chapter 3, cacti is the one who accomplishes requirements such as easy installation, easy to configure and really good user experience.

This section does not describe how to configure cacti, but defines what and how cacti should monitor each server.

First of all is important to define which parameters are going to be monitored. Some of them are common for all machines and some of them are interesting only in frontend cluster nodes.

As cacti allows creating menus to organize the information, every machine will have a submenu called "System", where the following information will be displayed:

- CPU usage of the system
- Memory usage of the system
- Load average
 - 1 minute average
 - 5 minutes average

- 15 minutes average

Moreover, there will be another submenu regarding to disk partitions. There, information about the amount of available or used disk will be shown.

Another submenu on every host should show statistics of incoming and outgoing traffic on every network interface.

The most interesting thing for stability testing is to measure the amount of memory used by a servlet engine. As there are four servlet engines, there will be a submenu called "Tomcat Memory Usage". In this submenu there will be four graphs showing the memory used for every tomcat.

The expanded tree of all information should be like this (see also annex 3.9):

- fend1om
 - System
 - Tomcat Memory Usage
 - Disk partitions
 - Interfaces
- fend2om
 - System
 - Tomcat Memory Usage
 - Disk partitions
 - Interfaces
- bend1om
 - System
 - Disk partitions
 - Interfaces
- bend2om
 - System
 - Disk partitions
 - Interfaces

All these data will be obtained via SNMP. Default MIBs of snmpd support all disk, system and interfaces data. Moreover, the plugin developed will support queries to every tomcat memory usage.

3.8. Stability test environment

Once a software version is released, it is time to make the stability tests.

Jmeter will be used as traffic injector to the frontend cluster. Jmeter will query all 4 servlet engines running on the primary node of the frontend cluster.

Tests will be carried for 5 days. During these days, it is important to have a look at the graphs generated by cacti. Load average must not be higher than 4, as machines have 4 CPU. Moreover, memory used by every tomcat should be stable. If there is not an excessive load and memory leaks do not appear, then stability test is passed and the new release will be considered stable.

Design of stability tests is out of the scope of this project, so they are not going to be explained in this document.

3.9. Installation process building blocks

This section will cover the design of the installation process. A good design in installation process can help to reduce time in later installations.

If no installation method is used, then the system administrator would have to install and configure every node manually. Eventually, it would become a lost of time and a prone error method.

In this section, components implied in the installation procedure will be detailed.

3.9.1. RPM packages

RPM [19] is the Red Hat package manager. This program is designed to build and manage software packages which may include both source and binaries files. It is portable and can be run on different platforms.

Normally, in Red Hat Operating systems, it is very useful to pack the binaries in RPM files. In this way, you can safely upgrade or downgrade, get version information and dependencies of a package. RPM provides a wide list of benefits which make it be the standard software installation method for Red Hat operating systems.

Red hat provides the rpm utility to install rpm packages. However it is quite useful if up2date is used to manage dependencies. Up2date is a package manager provided by Red Hat to maintain, install or remove rpm packages. It solves packages dependencies and downloads rpm files from repositories configured on the up2date configuration file.

3.9.2. Image of the system

When hardware involved in a clustered scenario is exactly the same for every server, it is quite useful to have an image of the system with all requirements for a node in the cluster environment. This image must have all packages updated with latest fixes from RHEL repositories, and basic and common configuration for all nodes.

That image would help to speed up the installation of a new server or it can be also used to recover a broken system. That part of the installation covers the static configuration of the server.

Beside, every system has also a specific configuration. This part of the configuration is achieved by some RPM including some scripts.

3.9.3. Automated way to fine tune the system

It is quite interesting to have a package to fine tune the system after a default installation. After the image has been restored from the default operating system installation, this package will:

- Disable unnecessary services which are not used and would waste CPU and memory
- Configure init to start at runlevel 3
- Uninstall some packages which would conflict with future installations (e.g. MySQL version 4 or gcc-java).

3.9.4. Security level configuration

In order to pass the security auditory made by the client, every machine must be configured following client's security level document.

Configuring every machine would be a waste of time. Therefore, it would be desirable to have a package which configures the operating system to pass the security auditory.

This package will configure settings for:

- Login parameters (e.g. block system accounts for remote access)
- Sudo permissions
- /etc/issue message
- File permissions
- Apache httpd modules loaded
- Kernel parameters
- Configuring sysstat utility

3.9.5. Automated installation of the cluster environment

Linux-HA project provides 3 RPM to install software binaries. However, the configuration must be made for every node.

A RPM containing some scripts to configure every node would help to automatically configure the cluster environment. The design of this package

must take into account that there are 2 kinds of machines: frontend servers and backend servers.

Resources for every cluster are different, so there is a need to provide 2 separate packages. However the core scripts will be almost the same and most code will be reused.

IP addresses configuration is also included in this RPM. This package will provide automated configuration of the IP addresses to make the package independent of the network environment.

These packages would also provide resource agent LSB scripts and configuration files for every resource. This package will deploy servlet engines or will initialize MySQL server depending on the server role.

Eventually, the result of applying this RPM to the system is a cluster ready configuration.

3.9.6. Automated installation of a new software release

A RPM package will provide a script and a war file. After applying this RPM to the system, the following actions will be done:

- Deleting tomcat cache
- Deleting old war file
- Deleting old deployed directory of the web application
- Copying the war file in the *webapp* folder of tomcat

The most important feature of the RPM packages is applied at this point.

If new software release does not pass the acceptance tests, then the old version must be installed again. RPM provides the rollback functionalities to perform the step back or the step forward action.

CHAPTER 4. Implementation

In this chapter, all wishes designed in chapter 3 come true. Installation procedure and how the system is configured using RPM is described.

The goal of this chapter is to explain the procedure of implementing the design and some snippets of code will be written. However, the complete source code and implementation including configuration files will be attached in annexes.

Finally, SNMP daemon configuration is presented.

4.1. Installation procedure

The time to configure and set up a new server can be reduced by automating some steps. In this section the process to set up a new server is described.

Servers have a preinstalled Red Hat Linux Enterprise 4 system out of the box. The first thing to do was to update the system using up2date with the newest stable versions of RHEL repository packages. Multiple packages were updated including Linux kernel. Up2date wrote a new entry to grub configuration and the system could reboot correctly with the new brand kernel.

4.1.1. Ghost4lin

At that point, an image of the entire system was made and burnt in a DVD to future uses. Ghost4lin supports backup or restoring the whole disc from different sources. It includes DVD, an image file located in a partition of the system or FTP.

The easiest way to restore images to other machines was to setup a FTP server with anonymous user access. All machines had connectivity to that FTP server and downloaded the image to restore the whole disk with it.

This process took approximately 4 hours, so it only took 1 hour to achieve a server with all packages updated and ready to be configured.

4.1.2. RPM installers

Now that we have a complete clean system, some scripts and configuration files were programmed and packaged in RPM.

The complete set of RPM installers are listed below:

- gbrokertunning – This RPM contains a script to fine tune the system.
- gbroker-frontend – This RPM contains scripts and configuration files used to setup a frontend cluster.

- gbroker-backend – This RPM contains scripts and configuration files used to setup a backend cluster.
- gbroker-gatekeeper – This RPM contains the gatekeeper war file and a script that performs tomcat cache cleaning and new war file deployment.
- gbroker-licensing - This RPM contains the licensing war file and a script that performs tomcat cache cleaning and new war file deployment.
- gbroker-o_m - This RPM contains the O&M war file and a script that performs tomcat cache cleaning and new war file deployment.
- gbroker-swupgrade - This RPM contains the swupgrade war file and a script that performs tomcat cache cleaning and new war file deployment.
- gbroker-pos – This RPM contains configuration files and scripts to set up security parameters on the server.
- gbroker-snmp – This RPM contain scripts, plugin for snmpd and configuration files to set up SNMP support on the machine.

All these RPM described above have some tasks in common. The first step is to program some bash scripts to perform installation tasks. Then, these scripts, configuration files and all necessary data are packaged into a RPM.

This process is common to all installers, so in next subsection the RPM creation is going to be described just for one package. In annex 1, RPM creation scripts can be found for all installers described in the list above. Moreover, gbroker-frontend and gbroker-backend will be explained in detail in annex 4.

4.1.3. RPM creation

First of all a workspace is created to program the rpm creation scripts and spec files. This workspace is created in /etc/gbroker-build.

Moreover, the system where RPM creation is made must have installed the tool *rpmbuild*. In this project this tool was obtained by installing the package *rpm-build-4.3.3-22_nonptl*.

The process described below will create a RPM for the frontend cluster nodes and its name is gbroker-frontend.

Every RPM creation consists on 2 files. The first one is a bash script which gathers necessary files to create the RPM and then executes the *rpmbuild* command. The second file is a spec file which is the description file of the RPM. It contains version, dependencies, change log, and information about the package.

4.1.3.1. Script to create the RPM

The common process for all RPM files created is:

- Create a temporary folder and copy all files that are going to be packaged
- Create a compressed tgz file with all files in the temporary folder
- Copy the tgz file to /usr/src/redhat/SOURCES
- Copy the spec file to /usr/src/redhat/SPECS
- Execute rpmbuild with the spec file corresponding to the RPM

After executing this script, the rpm file will be located at /usr/src/redhat/RPMS/i386.

```
#!/bin/bash

VERSION=1.0
# Moves to workspace /etc/gbroker-build
cd /etc/gbroker-build

# Remove previous rpm creation files
rm -f -R gbroker-frontend-$VERSION

# Copy all files in /etc/gbroker
# These are the installation bash scripts, configuration files and
binaries
mkdir -p gbroker-frontend-$VERSION/etc/gbroker
cp -R /etc/gbroker/* gbroker-frontend-$VERSION/etc/gbroker/

# A README file is attached to the RPM
cp README_tomcat gbroker-frontend-$VERSION/README

# Remove unnecessary files
rm -fR gbroker-frontend-$VERSION/etc/gbroker/backup-*

# Create tgz.
# This will be the input for rpmbuild command as specified in gbroker-
frontend.spec
# Compress directory gbroker-frontend-$VERSION in a tgz file
tar -Pcvzf gbroker-frontend-$VERSION.tgz gbroker-frontend-$VERSION
rm -f -R gbroker-frontend-$VERSION

# Copy spec file to /usr/src/redhat/SPECS
# This is the default path where rpmbuild looks for spec files
cp -f gbroker-frontend.spec /usr/src/redhat/SPECS

# Copy spec file to /usr/src/redhat/SOURCES
# This is the default path where rpmbuild looks for sources
cp -f gbroker-frontend-$VERSION.tgz /usr/src/redhat/SOURCES

# Removes the tgz file located in /etc/gbroker-build
rm -f gbroker-frontend-$VERSION.tgz

# Executes rpmbuild command to create the RPM
rpmbuild -ba /usr/src/redhat/SPECS/gbroker-frontend.spec
```

4.1.3.2. *gbroker-frontend.spec* file

The spec file consists on different parts:

- Header. General information about the package (copyright, version, company) and dependencies. It also includes fields such as “source” which must be the same tgz file created with the rpm creator bash script. Another powerful feature of RPM packages is that dependencies and requirements can be set. For instance, gbroker-frontend RPM will require Heartbeat and jdk packages installed.
- Body. there are several sections:
 - %prep, %setup, these are the macros to create RPM. Here some previous actions to RPM creation are specified.
 - %build. This section is normally used to compile source code. This is not used here.
 - %install. In this section files are copied to the destination path.
 - %post. Commands within this section will be executed just after the package has been installed.
 - %clean. It is used to clean source code from the package.
 - %changelog. Log of latest changes in the RPM. It must follow a determined syntax

This is the spec file for gbroker-frontend RPM creation:

```
# First section: Header
Summary: This RPM set up a HA cluster environment for gbroker project
Name: gbroker-frontend
Version: 1.0
release: 1
Copyright: GPL
Group: Productivity/Networking
URL: http://www.genaker.net
Vendor: Genaker
Packager: Genaker
Source: gbroker-frontend-1.0.tgz
AutoReqProv: no
BuildRoot:    %{_tmppath}/%{name}-%{version}-build
Requires: jdk
Requires: httpd = 2.0.52
Requires: Heartbeat = 2.0.8
Requires: Heartbeat-pils = 2.0.8
Requires: Heartbeat-stonith = 2.0.8
Requires: gbrokertunning

# Second section: prep
%description
Summary: This RPM set up a HA cluster environment for gbroker project
%prep
%setup

### Third section: build
%build

### Fourth section: install
%install
if [ -n "%{?buildroot}" ] ; then
    [ %{buildroot} != "/" ] && rm -rf %{buildroot}
fi

# Install all files which where packaged from /etc/gbroker
mkdir -p %{buildroot}/etc/gbroker
cp -R etc/gbroker/. %{buildroot}/etc/gbroker

### Optional section: After installation of the package these commands
will be executed
%post
echo "Run /etc/gbroker/scripts/install.sh"

### Fifth section: clean
%clean

### Change log
%changelog
* Mon Jul 09 2007 V.1.0 Genaker.
- This is the first tomcat release

### This is the list of files installed with this package
%files
/etc/gbroker/*

### Optional field

%doc README
```

4.2. Configuring the clusters

This subsection is a quick guide on how to configure both frontend and backend clusters. Let us assume that all RPM described above were built correctly and they are ready to be installed.

4.2.1. Pre-requisites to configure the cluster

There are several packages that must be installed on every machine in order to run applications correctly.

On one hand, in the frontend cluster the following packages must be installed:

- httpd = 2.0.52
- jdk = 1.5.0_12

On the other hand, the backend cluster must have the following packages installed:

- MySQL-server-enterprise = 5.0.46

Additionally, there are some common packages which must be installed on both clusters:

- Heartbeat = 2.0.8
- Heartbeat-pils = 2.0.8
- Heartbeat-stonith = 2.0.8
- gbrokertunning
- net-snmp-5.3.0.1-1
- MySQL-client-enterprise = 5.0.46
- gbroker-pos

Installing gbroker-pos package ensures that the server is ready to pass security tests made by the client and to be installed in the production environment.

4.2.2. Frontend cluster configuration

The RPM named gbroker-frontend-R1.0-1.rpm will configure a cluster with the following resources:

- DRBD partition
- Virtual IP Addresses
- Tomcat instances
- Apache web server

This package contains a script called `install.sh` that will execute following scripts:

- `config_apache.sh`
- `config_drbd.sh`
- `config_eth.sh`
- `config_heartbeat_comun.sh`
- `config_heartbeat_frontend.sh`
- `config_java.sh`
- `config_network.sh`
- `config_tomcats.sh`

These scripts will configure apache web server, DRBD, network interfaces, heartbeat resources, Java environment variables and tomcats. Specific values for every node are set in file `gbroker.conf` included in the RPM package.

After installing this RPM, frontend cluster will be running correctly.

4.2.3. Backend cluster configuration

In the same way that it was done with the frontend, a RPM named `gbroker-backend-R1.0-1.rpm` can be built from source code.

This RPM will configure heartbeat parameters and cluster resources. Moreover it will configure MySQL database engine replication method and network interfaces with values retrieved from the file `gbroker.conf`.

After installing this RPM, backend cluster will be running correctly.

4.2.4. Web application deployment

Let us assume that both clusters are running correctly. Now it is time to install and deploy Genaker's software on every tomcat instance.

Four tomcat instances are running on the frontend cluster. Four RPM packages are needed to install every web application on a different tomcat instance. These packages are named:

- `gbroker-gatekeeper-R1.0.0-1.rpm`
- `gbroker-licensing-R1.0.0-1.rpm`
- `gbroker-o_m--R1.0.0-1.rpm`
- `gbroker-swupgrade-R1.0.0-1.rpm`

When all RPM packages are installed, Genaker's software will be running on a high availability platform.

4.3. SNMP daemon configuration

There are several parts in the configuration file of snmpd (/etc/snmp/snmpd.conf) which can be configured. These directives can be classified into four categories:

- Those controlling who can access the agent
- Those configuring the information that is supplied by the agent
- Those controlling primary monitoring of the local system
- Those concerned with extending the functionality of the agent.

As a plugin was developed to answer queries to Genaker's enterprise tree, it has to be specified in this configuration file. After starting the daemon it will start answering queries to OID under Genaker's tree.

The package gbroker-snmp-R1.0-1.rpm must be installed in order to deploy configuration files, MIB definition file, MIB module and support scripts that retrieve system parameters.

CHAPTER 5. Evaluation in production environment

This section covers a set of tests that can be done in order to test system high availability. Both frontend and backend clusters are implied in these set of tests.

If the system does not pass these tests, it is considered unstable and the project will not be accepted. In next subsections, these tests will be presented.

Every test includes a procedure and a service check to verify that the system behaved correctly.

Lastly some conclusions are exposed, resuming and emphasizing the main concepts of this project.

5.1. Disc failure

Every machine owns 4 physical disks organized in 2 RAID-1 volumes. The second pair of disks contains the DRBD partition (/dev/sdb1 partition). In this section, DRBD partition failure will be tested.

5.1.1. Procedure

The goal of this test case is to make DRBD partition unavailable. To achieve that purpose, the second RAID-1 volume must be destroyed. That can be done just by unplugging the physical disks that belong to the RAID-1 volume where DRBD partition is mounted.

First disk extraction will not make any system failure. However, when the second disk is unplugged, DRBD partition will get unavailable, and Heartbeat monitor will fail. At that moment, all resources will fail over the secondary cluster node and this one will become the primary node.

5.1.2. Service Check

DRBD resource can be monitored with the command “cat /proc/drbd” to annex 2.2 for further information on DRBD.

5.2. Cluster node failure

Primary node in the cluster is powered off. It can be halted, rebooted or whatever method to make the node out of service.

5.2.1. Procedure

The procedure is quite simple. Just halt the machine with the shutdown node.

5.2.2. Service Check

All resources will fail over the secondary cluster node, which will become the owner of all resources. Refer to annex 2.3 for further information about Heartbeat status monitoring.

5.3. Application failure

This test simulates a failure of the servlet engine. There are four different tomcat resources where this test case can be done.

5.3.1. Procedure

To simulate a tomcat failure, stopping the service is enough. To stop the service resource agent scripts can be used. In this example, test with the tomcat that runs gatekeeper module will be used.

```
# /etc/ha.d/resource.d/tomcat_gatekeeper stop
```

As soon as Heartbeat performs its monitor operation on the tomcat resource and it realizes that it is not running, it will restart the service using resource agent script. Then, failcounts for this resource will increase. For further information on resource failcounts monitoring refer to annex 2.3.2.

If the total score of failcounts for all resources reaches 6, then all resources will failover the other cluster node. If there are 6 resource failures on a cluster node, then Heartbeat considers it as a sick node. If resources fail on the new primary node for 6 times, then resources will not be able to run anywhere and the cluster will be out of service.

5.3.2. Service Check

Open a browser and introduce the correct URL where tomcat instance is bounded. If it responds to the default http site, then it means that tomcat is running again normally.

5.4. Connectivity failure

If the primary cluster node loses connectivity to the configured ping node, which is its default gateway, then it makes no sense to keep this node as the primary node. Resources running on this node will failover to the secondary node.

5.4.1. Procedure

Unplug the cable on eth4 interface. Default gateway is reached by means of this interface and unplugging the cable will cause a ping node status failure. Automatically, all resources running on the node without connectivity will fail over to the secondary node.

If both nodes lose connectivity to the ping node, then the cluster will remain out of service. When connectivity is recovered, resources will be started again on the node decided by Heartbeat.

5.4.2. Service Check

Use `crm_mon` tool to monitor Heartbeat status and see that all cluster resources are stopping on one node and starting up on the new primary node.

CHAPTER 6. CONCLUSIONS

There are some concepts that should be remarked in this project. First of all, high availability helps to increase service uptime and protect critical applications. However, having a high available platform does not avoid service outage. In fact, high availability concepts are based on 3 “R”: Redundancy, Redundancy and Redundancy. Proper hardware, operative system configuration and high quality network components are needed too. If the application running on a high availability cluster is not stable enough, the cluster will probably be out of service soon. It is said that having a two-node cluster increases availability in a “9”. It means, that if a server is 99.9% available, configuring a high availability cluster gives the platform a 99.99% of availability

Another important point is remote management. It makes no sense to configure high availability clusters if the system administrator has no knowledge about what it is happening in every machine. That is the reason why SNMP is almost a must in every server. It may be quite hard to configure a machine monitoring specific values, but in the end it helps to be aware of the complete system. The lack of SNMP functionalities in Linux-HA heartbeat version 2 clusters is a topic which is under discussion in Linux-HA mailing list.

There is something that should be pointed. This project covers the design of a high availability and not load balance architecture. However, in a near future the platform will scale by adding a load balancer based on two servers configured with Linux-HA Heartbeat. Therefore, all knowledge about Heartbeat achieved with this project will be applied in next load balancing architecture.

Lastly, enterprise services will not make sense without a high availability platform. Does anyone want a non available platform? Linux-HA project community is growing exponentially, and in my humble opinion, it has a quite promising future

CHAPTER 7. BIBLIOGRAPHY

- [1] DELL POWEREDGE SPECIFICATIONS [online]
http://www.dell.com/downloads/global/products/pedge/en/pe2970_specsheet_final.pdf [visited: 6th June 2007]
- [2] REPLICATION IN MYSQL [online].
<http://dev.mysql.com/doc/refman/5.0/es/replication.html> [visited: 27th June 2007]
- [3] GETTING STARTED WITH LINUX-HA [online] <http://www.linux-ha.org/GettingStartedRevisedV2> [visited 6th June 2007]
- [4] COMMERCIAL HIGH-AVAILABILITY SOFTWARE FOR LINUX [online]
<http://www.linux-ha.org/CommercialSoftware> [visited: 6th June 2007]
- [5] LINUX-HA [online]. <http://www.linux-ha.org> [visited: 6th June 2007]
- [6] HP SERVICEGUARD FOR LINUX [online].
<http://h18026.www1.hp.com/solutions/enterprise/highavailability/linux/serviceguard/documentation.html> [visited: 11th June 2007]
- [7] RED HAT CLUSTER SUITE FOR RED HAT ENTERPRISE LINUX 4.6. 2007 [online]
http://www.redhat.com/docs/manuals/csgfs/pdf/Cluster_Suite_Overview-4_6.pdf. [visited: 11th June 2007]
- [8] INTERNETWORKING TECHNOLOGY HANDBOOK [online]
<http://www.cisco.com/en/US/docs/internetworking/technology/handbook/SNMP.html> [visited: 18th May 2008]
- [9] NET-SNMP PROJECT. [online] <http://www.net-snmp.org> [visited: 6th June 2007]
- [10] SNMP SOFTWARE [online].
<http://www.snmpink.org/snmpsoftware/forenduser/#7> [visited: 6th May 2007]
- [11] THE MULTI ROUTER TRAFFIC GRAPHER [online].
<http://oss.oetiker.ch/mrtg/> [visited: 6th August 2007]
- [12] ROUTERS2.CGI [online].
<http://www.steveshipway.org/software/index2.html> [visited: 6th August 2007]
- [13] NAGIOS [online]. <http://www.nagios.org/> [visited: 18th July 2007]
- [14] CACTI SNMP FRAMEWORK [online]. <http://www.cacti.net/> [visited: 6th August 2007]

[15] DRBD [online]. <http://www.drbd.org/> [visited: 16th June 2007]

[16] NETWORK MANAGEMENT & MONITORING WITH LINUX [online].

<http://www.david-guerrero.com/papers/snmp/> [visited: 8th June 2007]

[17] LINUX MEMORY MANAGEMENT [online].

<http://tldp.org/LDP/sag/html/sag.html#MEMORY-MANAGEMENT> [visited: 28th May 2007]

[18] HOW TO BUILD RPM [online]. <http://rpm.org/RPM-HOWTO/build.html>

[visited: 20th June 2007]

