High Availability Services

Arjan Dekker, Remco Hobo

July 4, 2005

Abstract

We live in the information age, to run a company you need information and it needs to be available all the time. If for some reason, information becomes unavailable this could mean great losses for the company. This article will discuss various ways of making sure data stays available by investigating so called High Availability solutions.

Contents

1 Preface

Today many organizations make use of computer systems. When there is something wrong with these computer systems the organization will lose profit and customers will lose their trust in the professionalism of the organization. Many organizations depend on reliable computer systems. E-commerce company's, for example, must have a website that is always available. There are several solutions to get the availability of the computer systems as high as possible. Many organizations have spare parts. When there are problems, the computer system can be examined and probably be fixed using the spare parts. This solution brings some downtime with it because when the computers system is being examined it will not be available. HA solutions assure availability even when some components of the service have failed. Examples of services are websites, e-mail, LDAP and online calendars.
This project was part of our education at the University of Amsterdam and our supervisor was Jaap van Ginkel. Hereby we would like to thank Jaap for his feedback, support and encouragement.

2 High Availability

A HA cluster can be used to keep a service available when some components of the server cluster are unavailable. This can be due to failures in hardware, software, scheduled downtime etc. A HA cluster can also be a solution for human errors as a misconfigured server can automatically be excluded from the cluster. Also during maintenance a server can be administratively taken offline while the service is still available. When the maintenance to the server has completed, the load balancer will automatically add the server to the cluster again.

Not all computer systems have to use a High Availability solution. Some services are very important for the organization, others can be unavailable for a while without any consequences. Its important to visualize which mission critical services there are and which dependencies they have. Mission critical systems are a good candidate for a High Availability solution.

3 HA challenges and requirements

High Availability sounds very interesting, but there are some drawbacks when using it. This chapter describes the drawbacks of High Availability. Further on in the document ways of overcoming these drawbacks will be described. The goal of this chapter is to provide a clear picture about the strengths and weaknesses of HA.

3.1 Costs

When mission critical services are down the organization will lose profit. Investing in equipment is therefore important. High Availability solutions costs money as additional equipment is needed and the implementation costs time. Also, staff has to be trained in managing the system. To make sure the HA solution is cost efficient, an analysis has to be made to see if the heightened availability is worth the cost.

3.2 Sessions

As will be described later on, there are several High Availability solutions. Some of these solutions have difficulties with sessions. A session is a two-way connection to some kind of service. The client connects to a server and a session will be created. During this session the client and server send and receive data. When the client or server ends the connection, the session will be closed. HA load balancers are located at the border of the service and can redirect a client from a failed server to another server. The service will therefore still be available, but any state that was present on the failed server has been lost. State information has to be encoded in the request message (URL) or the state has to be preserved at the client side (cookie). When one of the load balancers failes, the other one will take over. Some implementations allow for two load balancers to share TCP information. This means that for instance, downloading of a file will continue.

3.3 Encryption

High Availability solutions can have difficulties with encrypted sessions. Examples of services with encrypted sessions are HTTPS (encrypted website sessions) and LDAP. Some High Availability solutions can not preserve sessions when they are encrypted. During this project we will investigate the possibilities of encrypted session preservation.

3.4 Every added component reduces stability

HA is supposed to make a service more reliable but the argument against HA is that for HA more components in hardware and software are needed and thus more points of failure are introduced. Some High Availability solutions use more systems and software then others. It is important to investigate the level of availability the service must be and which systems and software must be used.

3.5 Installation / Configuration

Installation and configuration of a High Availability service is not very easy. There are multiple computer systems which have to be installed and configured. Every High Availability solution must be configured differently. During this project we installed and tested multiple High Availability solutions. In this document we describe the difficulties of the installation and configuration process. It is also important to do some research to see if there is enough documentation available about the chosen High Availability solution.

3.6 Flexibility

Certain High Availability solutions are more flexible than others. In this document we describe which solutions are highly flexible and which solutions are static. When a static solution is chosen, difficulties will occur when upgrading the servers. It can occur that the whole service must be taken down while upgrading the servers. Therefore it might be important to choose a flexible High Availability solution.

4 HA solutions

The main requirement for High Availability is that every component should be redundant. Therefore a system has to be devised so that a single failure of any component will not affect the whole service. Even when multiple failures might occur, it is preferable the service will notify clients of temporarily difficulties instead of remaining silent.

4.1 ARP-Spoofing

The ARP-Spoofing method is used to mimic a server in case the main server becomes unavailable. This means that two servers are equipped with the same services so that when one goes offline due to maintenance or due to failure, the other will take its place. It will send out an ARP reply telling the network the IP address for the failed server is now reachable at its own MAC-Address. This

is done using a Gratuitous ARP Broadcast. This means that an ARP packet is sent out without someone asking for it. This way, the failed master cannot answer with a rogue ARP-reply. If this where the case a race condition could occur when both servers would answer the same ARP request. When a failover occurs, a client does not have to reconnect to the network and will not notice the change of servers as long as the servers are stateless. If the servers are stateful, the state has to be encoded into the URL or in a cookie otherwise the session will be lost.

4.2 High Availability clustering with virtual servers

There are two common types of HA clusters available: HA IP and HA application clusters.

High Availability IP cluster

A High Availability IP cluster uses a virtual server mechanism that supports virtual IP addresses. It uses the same techniques as ARP-Spoofing but has some extra features. One virtual server will be the active server and the other one will be hot-standby. When the hot-standby virtual server notices the other server has gone down or it has less functionality in terms of connectivity as the active virtual server, it will become the active virtual server. It does this while retaining its own unique physical IP address, through a process referred to as IP failover. Clients will automatically be reconnected to the other server without reconfiguration. The two virtual servers will transmit a heartbeat message to each other on a predefined interval to announce they are still running. The active virtual server will attach to a predefined shared IP address and will become the host for that address. Some implementations can load-balance certain types of applications when their contents are replicated across a pool of application servers. A HA IP cluster will be concerned with maintaining connectivity at the border of the provided service. It has no way of telling an underlying service has failed, it will just use a job scheduling mechanism to forward requests to predefined servers regardless of their state.

There are numerous scheduling algorithms, below the two that are most widely used are explained.

- Round-Robin Scheduling

  A round-robin scheduling algorithm will have a list of servers (server A,B and C). When C is reached, the next request will overflow the list, so A is used again, completing the cycling or 'round-robin' of servers. It treats all servers as equal regardless of incoming connections, response times etc. The algorithm has a few advantages over traditional round-robin DNS. With normal round-robin DNS, a server resolves a single domain for different IP addresses and scheduling granularity is host based. Also, due to caching of DNS queries, significant dynamic load imbalances among the servers could be expected. The scheduling granularity of round-robin used in a HA cluster is network connection-based, and is much superior to round-robin DNS due to the fine scheduling granularity.

- Weighted Round-Robin Scheduling

The weighted round-robin scheduling is used to handle servers that have different processing capabilities. All servers will have a weight assigned to hem. This weight is an integer that indicates the processing power of the server. The higher the weight of the server, the more new connections the server will receive. When we have servers A, B and C, and they have the weights, 4, 3, 2 respectively, a good scheduling sequence will be AABABCABC in a scheduling period.

The weighted round-robin scheduling is better than the round-robin scheduling, when the processing capacity of real servers are different. However, it may lead to dynamic load imbalance among the real servers if the load of the requests vary highly. This may mean that a lot of requests, requiring a large response, are directed to the same server while another server, with a lesser weight sits idle.

This algorithm can be used for load shaping, not load balancing.

A list of all scheduling algorithms[6] can be found in the bibliography.

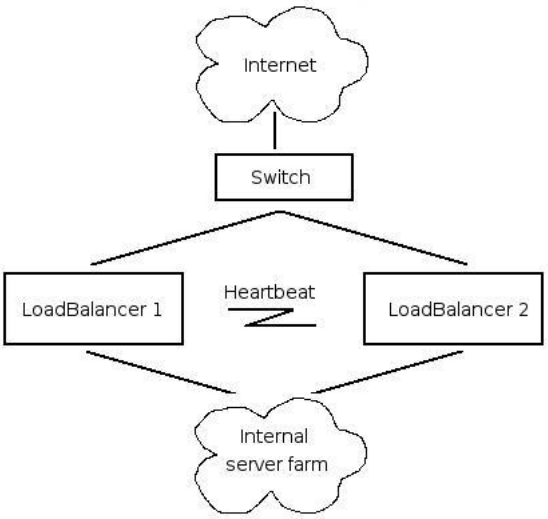

Figure 1: HA IP cluster

There are three different ways of forwarding packets; NAT, IP-IP encapsulation (tunnelling) and direct routing.

- Network Address Translation
  NAT can be used to manipulate the source and/or destination port and/or address of the packet. This means that a request from a client will be forwarded by the load balancer to an underlying server. The response form this server is then altered so the packet seems to be originated on the load balancer itself. This means packets pass through the load balancer twice so the load will be higher on the load balancer.

- Direct Routing
  With direct routing packets from a client will be forwarded directly to the

real server. As the IP packets are not modified the underlying server has to be configured to accept traffic from the load balancer's IP address. The underlying server can send replies back directly to the client so this is less demanding on the load balancer.

- IP-IP encapsulation
  Tunnelling is very similar to direct routing with the exception that packets that are sent to the underlying servers are tunneled. This means the underlying servers do not have to be in the same LAN or same geographical area. The underlying server replies to the clients directly.

High Availability Application cluster

The HA application cluster is used for stateful, transactional applications such as database servers, Web application servers and file servers. The HA application cluster will check the state of all underlying services and will determine which services are still available. It will then forward any requests and will act as a proxy for the underlying servers. If the HA application clusters proxies find no services are available, it can redirect traffic to an alternate website to, for instance, run a static web server locally apologizing for the trouble and advising people to come back later. In this approach, the proxy is the single point of failure, if it fails, the whole service will become unavailable.

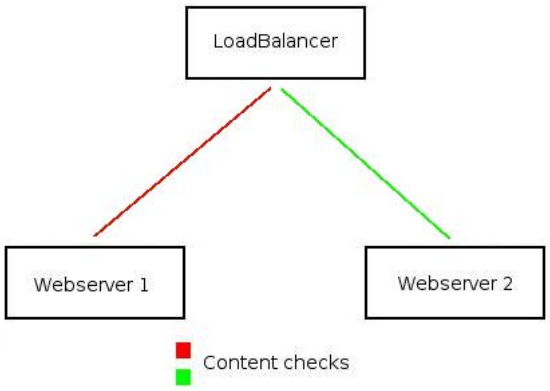

Figure 2: HA Application cluster

Combine both HA Cluster Types for a Multitier Solution

This is also called layer-4 switching. In most cases, High Availability and scalability are equally important for e-commerce or business-critical systems. When both types of HA clustering are combined, a Multitier application is possible. The virtual server/proxies will be the front-end of the service and will be backed up by a pool of application services. They in their turn can be backed up by a pool of database services providing full redundancy using the three-tier model.

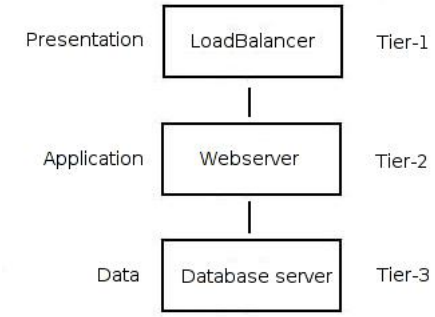

Figure 3: The 3-Tier model

If an implementation runs an active/active configuration for its load balancers, the load of the servers has to be kept in mind. Active/active means that for, both load balancers are active at the same time. If both load balancers should be experiencing a load of 60%, this would mean if one of the load balancers failed, the other would be overloaded. For this reason, an active/passive implementation might yield a higher availability rate.

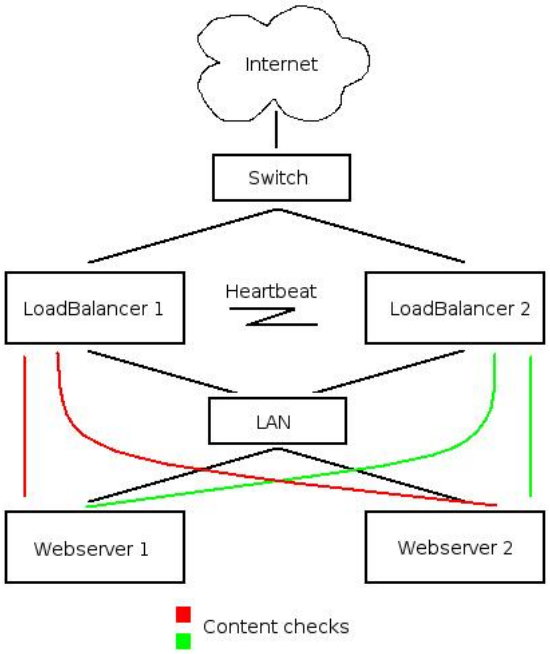

Figure 4: HA IP+Application cluster

5 Dedicated solutions and open source solutions

5.1 Hardware stability

Since all open source HA solutions run on normal PC hardware, this hardware has to be taken into account when advising on using an open-source solution. There are two types of PC hardware:

- Normal client hardware
  This hardware normally consists of consumer grade hardware that is less reliable and is much cheaper then server hardware. Normally, client hardware has no redundant components. Since this hardware is typically used to accommodate one user's needs it is not as important as a server. For economical reasons this type of hardware is used in most client machines.

- Server hardware
  This hardware is military grade, which means it is more precise in terms of component accuracy. This type of hardware is more reliable and will stay reliable longer under less than perfect circumstances like high temperatures. Components in this type of hardware can be redundant. Typically redundant components will be power supplies, fans and hard drives. These are the components more prone to failures then others. Server hardware is normally rack mountable, which means it can be installed into a 19' rack.

Naturally, when building an open source HA solution, the latter hardware is preferred. Still, this type of hardware can not compete with a purpose built load balancer like Cisco's CSS 11500 Series Content Services Switches[4]. Purpose built solutions have several advantages on using PC hardware:

- No hard drive; Normally a load balancer does not need a hard drive, it only needs to store a small configuration script;

- Easy setup; When a load balancer has failed, a new one can be configured in seconds by pasting it's configuration script into a console.

One great advantage of an open source solution is it's adaptability. All programs needed to operate are available as source code. This way it is easy to adapt a program to behave in a different way and to fine-tune the application for the company's needs. Also, a lot of additions and modifications for programs are available in the open source community and when you're making your own modification there is always someone willing to help.

For a good comparison about stability, the MTBF failure and the expected life time is needed. Unfortunately, this was unavailable for the Cisco load balancer.

For the server we have chosen a HP ProLiant DL560[13]. This server has a MTBF of 120,000 Power On Hours. This means this machine will typically die after 120,000 hours of approximately 13,7 years. Unfortunately, the MTBF also requires an expected lifetime which is not available. We will set this value to five years.

Below is the equation for calculating the availability of this server, where A is availability, MTBF is Mean Time Before Failure and MTTR is Mean Time To Repair, which we will set to half a business-day, four hours.

$$A = MTBF/MTBF + MTTR \tag{1}$$

$$120000/120000 + 4 = 0.9999666677 = 99.99666677\% \tag{2}$$

Having a double failure, e.g. both load balancer servers die of hardware malfunction would be

$$At = 1 - ((1 - A1) * (1 - A2)) = 0.9999999988 = 99.99999988\% \tag{3}$$

This would give you an 8-digit availability rate for the load balancers, meaning a double hardware malfunction is highly unexpected. After the five years have passed, the failure rate will start to increase. Of course this calculation only takes a hardware malfunction into account. Software stability will differ per solution and hard numbers on this do not exist.

6 Software Installation

This chapter contains information about ARP-spoofing and High Availability Clusters. These are both methods of setting up a High Availability service. This chapter describes what software and hardware is needed to set up these High Availability solutions. Also the flexibility, manageability and stability are described in this chapter. Total time spent on installing and testing these solutions is about 120 manhours.

6.1 ARP-spoofing

ARP-spoofing is the most simple method of High Availability. The low costs of ARP-spoofing make it a popular method of High Availability. ARP-spoofing is mostly used for HA on the load balancers and not for HA amongst the underlying webservers.

Requirements

ARP-spoofing is a low cost High Availability solution, because there are only two identical computer systems needed. The active system and a backup system. Each system must have at least one Ethernet card installed. We connected the Ethernet cards of both systems to a switch. The connection to the Internet was also plugged into this switch. From the Internet there can only be one system connected through the shared IP-address.

Installation

As described earlier, the installation and configuration is simple. Fake[10] is an implementation of ARP-spoofing. It is designed to take over the IP-address of a failed system. Fake is used in combination with other applications. These other application are used to check if the server is up and/or the service is still available. When such an application has noticed something is wrong it will start Fake. In our situation we used Heartbeat[11]. Heartbeat is a package which contains Fake and an application which checks the availability of other systems. Heartbeat must be installed on both systems. Heartbeat will send information about it‘s own availability and receive information about the other load

balancer's availability. This information only contains data about Heartbeat. There is no communication about the availability of the service. Only when the system has failed Heartbeat will be notified. Heartbeat can be extended with a module called Ipfail. Ipfail pings certain systems and notices when a connection to the Internet is lost. When such a connection is lost Fake will be started. When more load balancers are used it will check which load balancer has the most active connections. The load balancer with the most active connections will become active.

Session Continuity

The ARP-spoofing technique uses two systems which will check each other. When the backup system notices that the other system is down it will switch to active. The backup system will check the availability of the other system at a predefined interval. When problems are encountered it can take up to twenty seconds before the backup system will take over. The TCP-sessions are broken and therefore closed. For Highly critical services this is unacceptable. When TCP-sessions are lost clients must start a new connection. It can take a while before the backup server will take over so there is a change clients will stop using the service. This could be devastating for an E-commerce website. When the availability is less important it is possible to use this method as an automatic recovery tool for a service. Encrypted sessions are also lost because it uses a TCP-connection which will die when the switching of load balancers occurs.

Management

ARP-spoofing is easy to manage. One server may be active while the other one could be updated with newer versions of software. When the active system fails the service will be down as there are no systems left which could take over. Some organizations use this method so that downtime during system maintenance is eliminated. One system could be brought down and be upgraded without the need to hurry. Also when something goes wrong during updating of the system the service will be available because the other system is still active. When the updated system becomes the active one it is possible the service is temporary unavailable during the twenty second period of transition. When this happens while the service is not used by users this will not be a problem, otherwise it is a good thing to wait until low hours before switching back to the primary systems.

Flexibility

ARP-spoofing is not very flexible. Many services will work with this High Availability solution which is a good thing, but it is not very extendable. When performance issues are encountered it is not possible to place an extra server to solve the performance issues. The current connections are lost because it will take a while for the backup server to take over. Therefore the flexibility of switching from the master server to the backup server is not very high as this can only be done when the system is not used much. Fake has no support yet for IPv6 and no attempts have been made so far to add it

Stability and Robustness

As mentioned earlier there are people who think High Availability solutions only introduce more points of failure. ARP-spoofing is easy to set up and there is less equipment needed as with other High Availability solutions. Therefore it adds some points of failure to the service, but the availability of the service is much higher with ARP-spoofing as it reduces the number of single point of failures. The only thing that can go wrong is the detection of the status of the service. When this detection goes wrong it is possible the switch to the backup server fails or the switch to the backup server was unnecessary. Of course there is a small change the ARP-spoofing or service failure detection software is flawed and will crash the computer system.

Maximum time of unavailability

ARP-spoofing can be realized with standard hardware. Therefore the time to repair a computer system is low when spare parts are available. Many organizations have spare parts which can be used to rapidly fix the system. When one server is under maintenance, the other server will take over. Taking over the other system takes a while, because the backup server has to notice the other system has a failure. Every given time the backup server will check if the other server is alive and kicking. The amount of time to wait between the checks can be altered. Besides this, there is an option which configures the number of times the check may fail before the switch to the backup server will be made. The lower the time between the checks is set, the higher the amount of needed bandwidth. Also there is a change the service is overloaded and will not answer on time so the check will fail. When the number of failed checks is set to low, there is a possibility of an unnecessary switch to the backup server. The problem of the high load is solved, but not very elegantly. The connections will be lost, which will decrease the happiness of the users. So it is important to fine tune these settings with the service in mind.

How is discontinuity of a service discovered?

ARP-spoofing has no way of knowing an underlying service has failed. It is only used to make sure one of the two load balancers is online and functioning properly.

What kind of services can be provided?

ARP-spoofing can be used between two servers to check if the other has failed. This means that you can install the same services on both of these machines. These services have to be stateless, so static HTML, FTP, SMTP etc will be fine. If the service involves some kind of state, some other means has to be devised to make sure both servers know this state.

Performance

ARP-spoofing uses one server which is exactly the same performancewise as when there is no High Availability solution installed. The main server and the backup server are probing on each other which costs an amount of bandwidth.

Thus the performance of this High Availability solution is a little lower as a single server setup. The actual performance depends on the service which is running on the server and the system itself.

6.2 HA-cluster (UltraMonkey)

An important point of availability is performance. When a server is overloaded with requests the service will become unavailable. High Availability Clusters can improve the performance of a service. That is why websites with many visitors use load balancers. Load balancers are used in High Availability Clusters. Linux Virtual Server (LVS) is a kernel module which extends the normal kernel so it can perform load balancing. The following websites, amongst others[9], use LVS-based (Linux Virtual Server) load balancers:

- Linux.com
- Sourceforge.net
- Real.com (Real networks)

Of course there are more websites which use LVS-based solutions but the ones above are highly visited and well known websites so they are good example of the usage of LVS-based High Availability Clusters. Besides an LVS-based application, an application which detects the failure of a service is also needed.

Requirements

High Availability needs more hardware then ARP-spoofing. At least three machines are needed, namely a load balancer and two systems (servers) which have the provided service installed and running. The load balancer is the single point of failure when using a total of three systems. To resolve this, two load balancers can be used for redundancy. When one load balancer encounters a problem the other one will take over. This works with the ARP-spoofing technique. During this project we have chosen to use two load balancers and two servers.

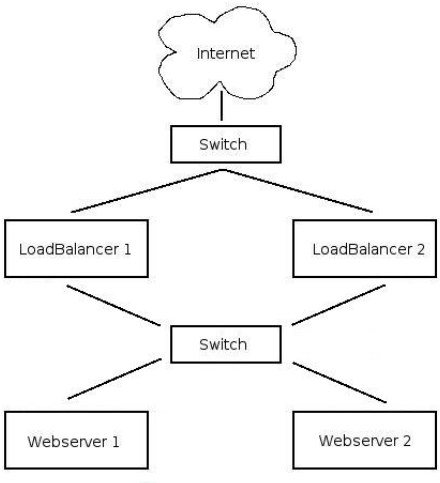

Figure 5: Our lab setup

Installation

High Availability Clusters use different pieces of software which have their own tasks. We have chosen UltraMonkey[18]. UltraMonkey is a combination of tools which can be used to set up a High Availability Cluster. UltraMonkey consists of Linux Virtual Server, Heartbeat and Ldirectord[12]. Linux Virtual Server is a kernel module which extends the normal kernel so it is able to load balance the connection attempts between the servers. Heartbeat is used to check if the load balancer is still alive. If Heartbeat detects the load balancer is not alive it will switch to the other load balancer. Heartbeat can be extended with a module, called Ipfail, which checks if the connection to the servers or the Internet is available. Ipfail pings some highly available machines and when this fails it assumes that the connection is down. These highly available machines could be some server somewhere on the Internet or the local servers. Ldirectord is used to investigate if a service is running without any problems. We made an HTML test page which contains some text, like "Service Up". Ldirectord will connect to the webserver and will get the HTML test page. When the test page is loaded, it will look for the words "Service Up". When these words are located inside the test page it will assume the service is up. These means if the service is down it will be detected because it will not return "Service Up" but an error page which does not contain "Service Up". The "Service Up" page can be self-written. It may check things like database connections, load and such before giving positive feedback to the load balancer. Ldirectord is highly extendable. It contains a number of checks for certain services. If it is required, custom scripts can be made to add a specific check. Another tool to test if a service is available is Mon[16]. Mon works like Ldirectord and has some minor improvements like sending out e-mails when an error occurs. Of course, Ldirectord could be extended so it also sends an e-mail when a problem is detected but does not has this option out of the box.

UltraMonkey delivers packages for Debian and Redhat Linux which simplifies the installation process. Of course there is a possibility to compile the software from source. There are three methods to route the packets between the servers and users. We have chosen to use the NAT routing method, because it is the easiest way to install and while it is not the fastest routing method it was sufficient for our testing purposes. The appendix contains the configuration files we used during this project.

Session Continuity

ARP-spoofing has the problem of losing connections when the master server fails. High Availability Clusters solve this problem much more elegantly. Services that use small sessions, like LDAP, DNS, SMTP, IMAP, HTTP, HTTPS and POP do not even notice when they get switched another server. When a server encounters problems (the "Service Up" page is not returned anymore) the load balancer does not use this server anymore and will remove it from the list of active servers. Ldirectord can be used to check which servers are running and which are not. Every given time the load balancer will check if the service on the server is up and running. When the output of this check is negative the server will be deleted from the list of active servers. When the server is available again it will be automatically detected and added to the list of active

servers. The time needed to switch from one server to another is so small it is not noticeable to users and applications. When the load balancer encounters problems it can take a couple of seconds to switch to the other one. The time to detect the failure of a load balancer can be configured.

During long lasting sessions, like downloading huge files with HTTP and streaming media, it depends on which problem occurs if the sessions will stay alive. LVS can be installed with a master and backup load balancer. The master load balancer synchronizes his connection information with the backup load balancer. When the master fails the backup load balancer will be activated and with help of the synchronized session information the active sessions will stay alive. When the master load balancer comes back online the session information will not be synchronized. So when the master load balancer is online the connections will be killed and the master load balancer will become active. There is an experimental application which synchronizes both ways. It is called ip_vs_user_sync_simple[14]. This is a solution to the shortcoming described above. The installation of this application is badly documented and it also needed some work to alter the source code to compile it.

When a server fails it will be removed from the list of active servers. When a user has been connected to this server the TCP-sessions are lost. Depending on the service it could be the application level session will overcome this problem. FTP is such an application as it supports resuming. Applications with small sessions, such as HTTP, will switch to another server without any problem. When the server delivers dynamic webpages the information about the session is lost when this information is placed on the server itself. To overcome this problem it is possible to encode the information into the URL or to alter the dynamic website so the location of the information is somewhere on an other system. For example, the information can be saved on the load balancers who must synchronize this information with each other. When a server fails the load balancer will divert the user to another server. This alternate server will look for dynamic website session information on the load balancer. If this information is found, it will be used. Also client-site cookies can be used, but these are not recommended because they could be altered by the user or a hacker which is active on the users machine. Also some users do not allow for cookies to be stored on their computer. High Availability Clusters can improve the availability, but before choosing for a particular setup all possible scenarios with this setup have to be taken into account.

Management

A High Availability Cluster is easily manageable. A server could be brought down without any problems, if the High Availability is well configured. Once a server is down it can be looked after without critical priority. Of course the other running servers will get some more connections to handle. Therefore it is important to investigate if the performance is good enough before taking the server down. Once the server is upgraded it can be brought online again. The load balancer will automatically add the server to the list of active servers. When this happens the next server could be upgraded. A High Availability Cluster uses at least three computer systems which must be managed. The servers and load balancers must be upgraded periodically. Often more systems are used, so this higher number of machines will take more time to manage in

comparison with ARP-spoofing.

Flexibility

The flexibility of a High Availability Cluster is outstanding. When the performance of the systems is low the number of servers could easily be extended. Install a new server and add it to the cluster is all that is needed to get it running. Adding a server to the cluster does mean editing the configuration file and restarting Ldirectord. When two load balancers are used, one of them can be reconfigured first, and by then forcing this load balancer to become active, the new configuration file will be used without any discontinuity of the service. Thereafter, the other load balancer can be updated. When a systems fails the impact for the service will be minor, because of the full redundancy. Many organizations buy hardware load balancers which are also flexible but are only usable for load balancing. When the load balancer is too slow a newer version of a load balancer has to be bought. Using standard hardware, which we use during this project, will make it easier to extend the performance of the load balancer. We could, for example, buy more memory or a new and faster motherboard. This is less expensive as buying a new hardware load balancer. When a hardware load balancer is too old to use, it can be thrown away while an load balancer made of standard hardware could be used for other purposes.

LVS is still in developing state which means not all functions are implemented. Ipv6 for example is still on the todo-list. The LVS programmers have started with modifications to make it possible to use Ipv6 with LVS, but it is still highly experimental and the latest version was released in 2002 which could mean that this project[1] is abandoned. Therefore, upgrading to Ipv6 is not recommended if LVS is used (at least not today).

Stability and Robustness

The stability of a High Availability Cluster depends on the hardware which is used. When cheap hardware is used there is a bigger change of encountering problems. Therefore we recommend the use of high quality hardware. A High Availability Cluster depends on many systems and software which could make it unstable. We have extensively tested our High Availability Cluster and encountered no strange behavior. The use of ip_vs_user_sync_simple is a little bit tricky because it is highly experimental but the other pieces of software are stable. The stability of the total setup is high even when there is more hardware which could fail. Every aspect of failure can be solved by the High Availability Cluster. The stability of the cluster could only be impaired if the software fails. When a load balancer announces it is up and running while it is not it could bring down the cluster. A High Availability Cluster allows for more time to carry out maintenance, but the stability of the cluster will be much higher in comparison with no High Availability solutions. The ability to add more servers while the service is still running without any problems improves the robustness of the service. A downside to this implementation is that when a switch has been made, the software will not check this switch has gone according to plan. We discovered this in the following scenario: We set up two load balancers. One had no connectivity and the other had only eth0 up. Naturally the latter became the active load balancer. When we also put eth1 up and restored con-

nectivity to the first load balancer, the program had no clue it had not attached to the eth1 interface of the second load balancer so all servers behind the load balancers remained unreachable.

What kind of services can be provided?

Not all applications could be used in combination with a High Availability Cluster. Some applications are smart enough to reconnect to the service and run without any problems. Other applications will lose the connection with the server and fail. Most applications that use small sessions, such as LDAP, DNS, SMTP, IMAP, HTTP, HTTPS, NTP and POP will work because if a switch to another server is made they will reconnect automatically to the new server. It is important to synchronize the data between the servers which is needed by the services. Dynamic websites could be altered to make this possible. Services which have long lasting sessions, such as FTP, downloading files from HTTP, SSH, TELNET and streaming media will have bigger problems during the switch to an other server. FTP has the ability to restart the session and let the download start at the location where it was when the problem occurred. Downloading a large file from a HTTP-server will stop when a problem occurs on the server because it is not intelligent enough to reconnect to the other server and continue(HTTP has no resume options). SSH is not useful with a High Availability Cluster because there is no way to now on which system you are working on.

LDAP

Many organizations use LDAP (Lightweight Directory Access Protocol). LDAP contains information about the users of the network. The authorization information is located within the LDAP-database. Many services use the LDAP-database to check if a user may or may not connect and use the service. When this service is unavailable many services cannot be used anymore. For this reason LDAP-service must be high availabile. High Availability Clusters could be used to make LDAP high available. LDAP itself contains some methods to set up a high available LDAP service but the flexibility could be extended when a High Availability Cluster is used.

We have investigated the best method of setting up a high available LDAP-service. LDAP uses a master server and slave servers. The master server can handle write request which the slaves can not. When the master server is unavailable the service is still available because the slave servers will handle the connections. To extend the performance of the service it is possible to add more slave servers. Load balancers could be used to reduce the time needed to configure the new situation. With help of load balancers the clients only have to know the IP-address of the slave server farm. Adding a slave server is therefore very simple because this IP-address is already configured at the clients. The following picture explains our design:

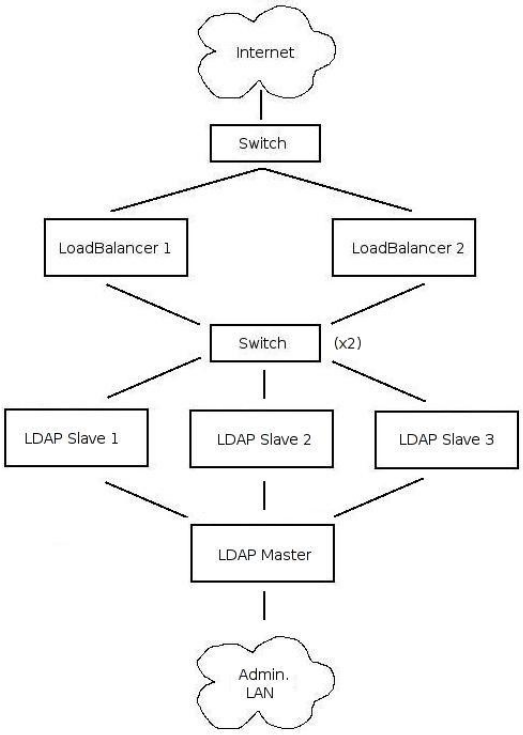

Figure 6: HA LDAP configuration

The master server could be connected through an administrator LAN which is shielded from the outside. The changes made to the master server will be replicated to the slaves. The number of slaves could be extended very easily. The load balancers are installed with Heartbeat and Ldirectord. When a load balancer encounters problems the other load balancer will take over. It is also possible to place redundant switches by using multiple network interfaces on the slaves and load balancers.

Streaming media
Today many radio and television related organizations offer the ability to listen or see their programs online. The load on the server can become quite high when a huge number of users are connected. Load balancers can be used to balance the load between multiple servers. With help of a High Availability Cluster it is easier to extend the performance because an extra server could be added without a long time to configure the new server. A new server can be added without discontinuity of the service. When ldirectord is restarted, the new configuration will become active. By switching to the other load balancer before ldirectord is restarted, an interruption of the service is prevented. Also when a load balancer has problems and the connection status is synchronized (with help of ip_vs_user_sync_simple) the other load balancer will take over and the media will still be streaming from the clients buffer. Depending on the client's buffer size, a small discontinuity could be experienced. When a streaming server has a failure, a client has to reconnect and will be forwarded to a functional server.

When custom services are made it is important to think about the future use of the service. When the service might be required to be highly available, this has to be taken into account when designing the service. Dynamic websites must be created with a High Availability Cluster in mind, because the location of temporary information can not be saved on the server itself.

Maximum time of unavailability

The maximum time of the service to be unavailable is very low if a server has a failure. When switching from servers there is no unavailability of the service. When the load balancer encounters a problem it has the same time of service unavailability as ARP-spoofing. When an application which synchronizes connection information is installed the connections remain so the time to switch from server is not a huge problem. High Availability Clusters have almost no time of unavailability which make it very useful.

How is discontinuity of a service discovered?

We have installed UltraMonkey on our load balancers which contains Ldirectord. Ldirectord investigates the state of an service. Ldirectord has the ability to control the list of active servers. When Ldirectord detects the failure of a service it removes the server from the list, also when it detects a server which could join the list it will be added to it. Ldirectord has several built in checks of certain services, such as HTTP, HTTPS, LDAP, POP, IMAP, SMTP, FTP, DNS and MySQL. Also we installed on our load balancers a plug-in to Heartbeat called Ipfail. Ipfail checks the availability of the network connections. When Ipfail detects a network connection which is down it will communicate with the other load balancer. The other load balancer may have more network connections which are available. If the other load balancer has more available network connections it will become the active load balancer.

Performance

One of the great advantages of High Availability Clusters is the easiness of extending the performance of the service. The load on the service is shared between multiple servers. When the servers are having too much connections to handle, it is possible to add an extra server without any troubles. ARP-spoofing only uses one server at the time so the performance is not balanced between multiple servers. We have tested if the load balancers could be overloaded with a script which simulates multiple users. Even with 10.000 simulations connections the load balancers were not overloaded but the webservers where. Huge websites with many visitors are using LVS-based load balancers without any performance issues. When websites are overloaded it is mostly due to the lack in performance of the servers. The documentation we used during this project said it is possible to handle 100.000 simultaneous connections. A 100Mbit link could be load balanced with commodity hardware. With higher-end hardware it is even possible to load balance a 1Gbit link and beyond.

6.3 DNS

By setting multiple A records for the same hostname, a website can be distributed amongst multiple servers. This way, a traffic shaping solution can be created. The DNS server will then use a round-robin like way to answer to DNS request so all servers will get approximately the same number of requests. This is a kind of traffic shaping as the load of a particular server and the type of request is not taken into account. By setting the TTL to a low value, in case of a failure, the DNS record will be picked up by a client after a short period of time. This approach has some downsides:

- Users might use the direct IP address instead of a host name.
- Users might use non-DNS methods like /etc/hosts to map server host names to IP addresses.
- A round-robin DNS solution cannot differentiate between a one-off hit, and a request that will result in many hits.
- There is no way to assign weights to servers thus their stability, load, resources are not taken into account.

DNS is not a good option for HA. It's strength lies more in creating a robust, easy to implement and transparent to end users method of distributing traffic to multiple server. DNS has no options to check connectivity of a server before it responds with its IP address to a request. When a server has failed, a system administrator has to manually remove the server from the DNS list.

6.4 Installation conclusion

DNS round-robin is not a real option for HA. It is an easy way to do some basic load shaping but it has no way to know a service has failed and no automatic way to exclude a server from the DNS list.

ARP-spoofing can be used to make two identical machines HA. Services on these two machines need to be in the same state. Content must be the same across both servers. When a session state has to be preserved, this information has to be stored on the client or encoded in the URL. Also, another means of synchronizing the states between servers can be implemented, but this is application dependent. It still is IPv4 only.

UltraMonkey can be used to create a fully redundant solution. Every component can be made redundant and is the best HA solution out there. It works quite stable, we have not had any unexplainable behavior with it. When multiple application servers are implemented, the state also has to be stored client side or encoded into the URL. Again, another means of synchronizing the states between servers can be implemented, but this is application dependent. UltraMonkey has no stable support yet for IPv6 and the documentation is insufficient. It can take a while before the cluster is setup to specific needs as not all options are documented.

7 Scenarios

When is what kind of HA solution applicable if any. In this chapter we describe three scenarios where High Availability solutions could be used to increase the availability of the service. This chapter aims at providing some examples on when to use which HA solution.

7.1 Scenario 1: A small business's email-service

A small company has an email server which send and receive the email of the company. Also it contains a POP-service so the employees can read and send email from their homes. Email is used as a way to communicate with customers, but the telephone is still the primary way of communication. Therefore it is not very devastating if the email server is down for a couple of hours. Also an email message will be hold for a couple of hours when an email server ca not be reached. When the server is back online the email message will be resend. The usage of the email-service by employees which are at home is very high. In the weekends there is no system administrator which can fix the email-server when it encounters problems. Therefore they use ARP-spoofing to take over the problematic email-server with a backup email-server. This way the email service is still available when problems are encountered. They use Heartbeat and Ldirectord to check if the SMTP and POP services are available. When problems are detected the software will let the email service switch to the backup server without any activity of a system administrator. It is a relative cheap solution to setup a High Availability service. The email messages located on the active server are replicated with the backup server.

7.2 Scenario 2: An ISP's secure Webmail service

An medium-sized Internet provider offers a Webmail-service to it's customers. This Webmail-service uses encrypted HTTP-sessions (HTTPS). The Webmail-service is a huge success and the usage was highly underestimated. Therefore they wanted a setup whereby the performance could be easily extended. All the email messages of the users are located at an IMAP-server. The provider has chosen to use a load balancer and two servers with Apache/SSL and a customized version of Squirrelmail[17] which connect to the IMAP-server to make use of the email messages. When in the future some performance issues will arise, it is possible to add multiple servers which could be used by the customers. When a server fails, the customer automatically will be redirected to another server. The customer does not even notice he/she has switched of servers. Squirrelmail does encode information into the URL and with help of Apache::Session[7] it is possible to save session information at another location besides the server itself which will make it possible to switch to a different server without any problems. Optionally they can add a second load balancer in the future for redundancy.

7.3 Scenario 3: An on line travel agency

An online travel agency has a website on which one can search for flights to a specific location and search for accommodation in the destination area. The

availability of this service should be as high as possible as it reflects the integrity of the travel agency. Any outages will have a negative effect on this and if outages are common, customers may lose faith and will book their holidays and journeys elsewhere.

Therefore the travel agency investigated which High Availability solution is the best solution to keep this service available. They had investigated the option to use load balancers with multiple servers. The outcome of this investigation was that they could use it, but not out of the box. They had to use the 3-tier architecture. The first tier is the presentation-tier which presents the front-end of the service to the customers. Load balancers are used to balance the load to multiple servers. These servers use SSL to provide a secure connection so no confidential information will be leaked. When a server encounters problems the costumer will be redirected to another server. This server has no information about the state of the previous session and therefore it redirects the customer to the login-page and starts a new session. When a customer wishes to book a holiday it will be send to one of the application-servers. This application-server does several checks. The application-server communicates with a database-server which contains information about the available seats in an airplane etc. When all checks out the booking can be finalized and stored in the database. When this was done successfully a message will be send to the presentation-tier. When the application-server encounters problems during a customer session it will be noticed by the presentation-tier. The server on which the customer is active will show a message to the customer which informs him/her of about the problem. After the server has become deactivated the customer will be redirected to another server. Here the customer has to login again and reenter the information needed to book the holiday.

By placing multiple servers the travel agency could expand the performance of their service. The availability of the service has grown, because the customer could always connect to one of the servers. The only negative situation is when a application server or database -server fails and the customer must reenter his booking information.

8 Findings

This chapter describes the problems we encountered during the installation and testing of our High Availability Cluster.

Poor documentation

We have chosen to install UltraMonkey as this project has several documents about the installation of High Availability Clusters. There are several projects which create software to setup an High Availability Cluster, but they are all using Linux Virtual Server as the basic tool. We looked at several of these projects and chose the one which offers the most documentation. The installation manual of UltraMonkey was useful but incomplete. It contains some information about the technique of Linux Virtual Server and how to install it. Also some information about Ldirectord and Heartbeat was available. With help of this installation manual we were able to install a working High Availability Cluster. Unfortunately the documentation only gives some scenarios and how to config-

ure them. It is unusable reference material as it lacks information about some options. Specific documentation is not available or very poorly written.

The documentation belonging to ip_vs_user_sync_simple is even worse. The word "ip_vs_user_sync_simple" only gets 56 hits at google at the moment. The only line of documentation told us to "run a script" which did not work. It seemed there where some kernel dependencies which where never documented. The installation procedure was described in a few lines and we where unable to find a complete list of the features it offers. After a couple of days experimenting with ip_vs_user_sync_simple we where able to get it running.

Heartbeat has the ability to be extended with a module called Ipfail. Ipfail must be started as a user called "hacluster". During testing we discovered the software fails to start because of insufficient rights. So we tried to get it working with root-privileges but this also did not work. After a day of searching on the Internet we found a web page which contains information about Ipfail. It turned out some option must be altered. The option "auto_fallback on" has to be changed into "nice_fallback on". The "auto_fallback on"-option was a legacy option which creates a nicer fallback method between the master and backup server. This is undocumented in most howto's so it took us quite a while to figure it out.

Poorly written source code and installation manual

Load balancers have the ability to be activated both on the same time. This is called active-active and could be used to increase performance on the load balancers. The application needed by Linux Virtual Server to make this possible is called Saru. Saru contains a patch to the kernel which extends the possibilities of the load balancer. During the installation of Saru we had to change the source code to get it working. The source code contained the following strange code:

```
/* Sillyness:
 * The varible is defined (in ha_msg.h) but need never be used.
 * But -Werror is turned on by default so we have to
 * do something with it, else we can't compile.
 */
if(_ha_msg_h_Id){ ; }
/* End of sillyness */
```

After we removed the "if(_ha_msg_h_Id) ; " from the source code it was able to compile. This problem occurs because we used a newer version of the GCC-compiler which does not turn "-Werror" on by default.

SSL and Load balancing

When using SSL and load balancing it may be useful to let the load balancer handle the SSL-connection. When a client connects to a server which uses SSL it is possible the client is unable to connect to another server when the original one fails. When the load balancer handles the SSL-connection the client could switch to another server as the SSL traffic is translated to normal traffic (HTTPS to HTTP). This will decrease the load balancer's performance. Handling a SSL-session is very demanding for the load balancer. The load balancer must handle a great number of connection attempts and can be overloaded when it also must

handle the SSL part of the connection attempt. Therefore an SSL offload card could be used. This card can handle SSL-connections in hardware which makes it a lot faster. Many proprietary load balancers could be extended with an SSL offload card. Also Linux Virtual Servers could be extended with a SSL offload card. HP, for example, offers such cards[8].

9 Conclusion

A High Availability Cluster is the best way to setup a service which must be high available because several failures may occur and the service will still be available. Upgrading servers is a lot less stressful if a High Availability Cluster is used because there is always another server which will take over the connection to the service. Before setting up a High Availability Cluster it is important to think about the difficulties we described in this document. Also not every application or service could be used in combination with a High Availability Cluster. Some applications must be altered to function in combination with a High Availability Cluster. Therefore an investigation must be done to check if the application or service should be used in combination with a High Availability Cluster.

The usability of an open-source High Availability Cluster is not easy to say. Open source software offers some great advantages, like openness, large communities and freedom of use. Of course there are some disadvantages, like poor documentation and the lack of support. Also, we encountered abandoned projects like LVS-IPv6[1] and Saru[15]. We advise not to use the open source solution if there is not enough knowledge about Linux/Unix available within the organization. The installation and configuration of a High Availability Cluster can only be done by someone who has experience with Linux/Unix. The documentation contains just enough information to setup a basic High Availability Cluster, but the maintenance and customization are poorly documented. Also the installation and configuration takes more time in comparison to a hardware load balancer. A great advantage of an open source High Availability solution is the possibility to customize the cluster. A great advantage of hardware load balancers is they are easier to setup by people who are less experienced. The purchase price of the hardware load balancers is higher, but a huge part of this investment will be regained from the lower maintenance costs. An open source HA cluster could be cheaper to setup and maintain as a hardware load balancing HA solution, because the hardware could be re-used after disabling the cluster. Also, the software is free to use. If there is enough knowledge available to setup a High Availability Cluster with open source software it could be a good choice to use it. Every organization is different so we could not give a generic conclusion about the usability of an open source High Availability Cluster. When load balancing is implemented one has to know the load of all servers. When two servers are under 60% load and one fails, the other will become overloaded. This way all HA effords will become useless.

10 Recommendations

During this project we have come up with some recommendations about the High Availability software. We recommended to the programmers of High Availability software to write more documentation which will increase the usage of this software. Also it was not always clear if the software project was abandoned or still alive, like LVS-IPv6[1] and Saru[15]. A project which looks abandoned will not be used by companies in highly critical situations. LVS is a stable application but it is not clear how well it performs. The creators of LVS should make a chart of the performance. This way it is possible to show future users how well it performs and which hardware they need to use to create a High Availability Cluster suited to their needs. There are some commercial companies which make use of LVS, like Red Hat's Enterprise edition, which have nice GUI's. UltraMonkey has no GUI at all. Perhaps it is a good idea to create such a GUI. This GUI must show the status of the servers and load balancers. Also some configuration options must be changeable with help of this GUI. While an experienced system administrator experienced in Linux/Unix knowledge can install UltraMonkey, the GUI can be used by less experienced personnell to maintain the cluster. This way the experienced system administrator is not the only person who can maintain the cluster. IPv6 is at this moment not supported by LVS. In the future this will be used more often. Therefore we recommend to start a project which develops a stable IPv6 load balancing solution or continue the current one[1].

References

[1] An IPv6 attempt for LVS:
http://www.yggr-drasill.com/LVS6/

[2] Architecting Linux High-Availability Clusters - Part 1:
http://www1.us.dell.com/content/topics/global.aspx/power/en/ps4q00_linux?c=us&l=en&s=corp

[3] ARP spoofing explained:
http://www.vergenet.net/linux/redundant_linux_paper/talk/html/node3.html

[4] Cisco CSS 11500 Series Content Services Switches:
http://www.cisco.com/en/US/products/hw/contnetw/ps792/index.html

[5] Creating webfarms with linux:
http://www.vergenet.net/linux/has/slides-stuff-jp/has-slides-jp.pdf

[6] Different scheduling algorithms explained:
http://www.linuxvirtualserver.org/docs/scheduling.html

[7] The Apache::Session webpage:
http://search.cpan.org/dist/Apache-Session/Session.pm

[8] The AXL600L SSL accelerator card:
http://h18004.www1.hp.com/products/servers/security/axl600l/

[9] The deployment of LVS servers:
http://www.linuxvirtualserver.org/deployment.html

[10] The Fake website:
http://www.vergenet.net/linux/fake/

[11] The Heartbeat website:
http://www.linux-ha.org/

[12] The Ldirectord website:
http://www.vergenet.net/linux/ldirectord/

[13] The HP proliant DL560:
http://h18004.www1.hp.com/products/quickspecs/11595_na/11595_na.html

[14] The ip_vs_user_sync_simple download site:
http://www.ultramonkey.org/download/conn_sync/

[15] The Saru active-active module for Ultramonkey:
http://www.ultramonkey.org/papers/active_active/active_active.shtml

[16] The Service Monitoring Daemon:
http://www.kernel.org/software/mon/

[17] The Squirrelmail website:
http://www.squirrelmail.org/

[18] The UltraMonkey website:
http://www.ultramonkey.org

11 Appendix

Configuration files
Here you can find the configuration files we used during this project. With help of these files it is possible to setup a High Availability Cluster which consists of LVS, Ldirectord and Heartbeat.

HA.CF

```
logfile /var/log/ha-log
keepalive 2
deadtime 10
serial /dev/ttyS0
node fake1
node fake2
auto\_failback on

ping 213.73.255.52 //dutch DNS server
ping 192.168.0.1
ping 192.168.0.2
respawn hacluster /usr/lib/heartbeat/ipfail
```

HARESOURCES

```
fake1 192.168.0.10/24/eth1
fake1 145.92.26.80/24/eth0
```

AUTHKEYS

```
auth 1
1 crc
```

LDIRECTORD.CF

```
# Global Directives
checktimeout=3
checkinterval=1
fallback=127.0.0.1:80
autoreload=yes
logfile="/var/log/ldirectord.log"
quiescent=yes

# HTTP
virtual=1.2.3.4:80
        real=192.168.0.1:80 masq
        real=192.168.0.2:80 masq
        service=http
        request="test.html"
        receive="Page Up
        scheduler=rr
```

```
        protocol=tcp

# HTTPS
virtual=1.2.3.4:443
       real=192.168.0.1:443 masq
       real=192.168.0.2:443 masq
       service=https
       request="test.html"
       receive="Page Up"
       scheduler=rr
       protocol=tcp

# LDAP
virtual=1.2.3.4:389
       real=192.168.0.1:389 masq
       real=192.168.0.2:389 masq
       service=http
       checkport=80
       request="testLDAP.html"
       receive="Page Up"
       scheduler=rr
       protocol=tcp


# Streaming media
virtual=1.2.3.4:9000
       real=192.168.0.1:9000 masq
       real=192.168.0.2:9000 masq
       service=http
       checkport=80
       request="testStreaming.html"
       receive="Page Up"
       scheduler=rr
       protocol=tcp
```