Clusters de alta disponibilidade – uma abordagem *Open Source*

Filipa Ferreira, Nélia Santos, Mário Antunes
Escola Superior de Tecnologia e Gestão de Leiria
Instituto Politécnico de Leiria
Morro do Lena - Alto Vieiro - 2401-951 Leiria - Portugal
Telef: +351 244 820300; Fax: +351 244 820310;
e-mail: eic09037@student.estg.ipleiria.pt, eic09052@student.estg.ipleiria.pt, mario.antunes@estg.ipleiria.pt

***Resumo* — Este artigo descreve a construção de um *cluster* de alta disponibilidade recorrendo apenas a componentes de *software* de código aberto. A solução apresentada assenta na implementação da arquitectura *Heartbeat* num *cluster* com dois nós, testando a sua adaptabilidade a serviços comuns da Internet, como o Web e o Proxy. Os cenários implementados e os testes efectuados pretendem demonstrar o potencial desta arquitectura em ambientes Linux, tirando partido do baixo custo associado, da facilidade de configuração e da actualização permanente do *software*.**

1. Introdução

A infra-estrutura de suporte aos sistemas de informação de uma organização é constituída por diversos componentes, destacando-se os servidores, unidades de armazenamento de dados, acedidos pelos utilizadores através de uma rede. Os equipamentos envolvidos, nomeadamente servidores, discos e equipamentos activos de rede, são constituídos por componentes electrónicos que podem avariar, provocando uma falha no equipamento em causa. Atendendo à dimensão da rede, a probabilidade de ocorrência de falhas em equipamentos aumenta, bem como as suas paragens não planeadas, afectando a disponibilidade global da infra-estrutura e o acesso às aplicações pelos utilizadores. Especialmente em aplicações do tipo *mission critical*, que asseguram as actividades do negócio, este problema é particularmente delicado.

Actualmente todos os fabricantes dedicam muita atenção à garantia de disponibilidade dos equipamentos, implementando mecanismos redundantes que garantem a continuidade das operações em caso de falha e eliminem o maior número de pontos críticos de falha. Por exemplo, é comum ter servidores com fontes de alimentação redundantes, bem como implementar sistemas de RAID (*Redundant Array of Intelligent or Inexpensive Disks*) [1] para assegurar a escrita dos mesmos dados em dois ou mais discos diferentes. Do ponto de vista de desenho da rede, há várias estratégias que podem igualmente ser implementadas no sentido de aumentar a sua disponibilidade. Por exemplo, instalar duas interfaces de rede em cada servidor, ligadas a equipamentos activos distintos e fisicamente distantes, garantindo assim o acesso em caso de falha de uma das ligações. No entanto, o conceito de alta disponibilidade (*High Availability*) [2] não se restringe apenas ao desenho da rede, coligindo todas as acções específicas levadas a cabo na organização (e.g. condições físicas, servidores, discos), dotando a infra-estrutura, no seu todo, de um alto índice de disponibilidade para o utilizador final.

Embora ao nível dos servidores haja um conjunto de pontos críticos de falha que são corrigidos individualmente (e.g. fontes de alimentação, uso de duas interfaces de rede), o próprio servidor também é um desses pontos críticos de falha. Por variadas razões (e.g. erros de *software*, erros humanos e causas naturais), o servidor pode simplesmente parar, tornando-se indisponível para os utilizadores. Nesse caso, é necessário que, de forma transparente, o utilizador seja reencaminhado para um outro servidor que desempenhe as mesmas funções. Neste caso, os servidores são "nós" de um conjunto, que se designa de *cluster,* e que disponibiliza serviços aos utilizadores de forma contínua.

O conceito de alta disponibilidade pode ser conciliado com o balanceamento de carga no *cluster*. Ou seja, o pedido de um utilizador enviado para o *cluster* é atendido por um nó que se encontre operacional e com uma carga de processamento mais baixa.

Actualmente a maioria dos construtores [3],[4] desenvolvem soluções integradas de alta disponibilidade proprietárias. Além do custo elevado, estas soluções assentam em arquitecturas proprietárias de um fabricante. Por outro lado, os sistemas de código aberto (*Open Source*) têm conseguido uma aceitação crescente nos últimos anos em várias áreas. Também na implementação de arquitecturas de alta disponibilidade é possível encontrar soluções de actualização quase permanente e a custos reduzidos.

Este artigo descreve a implementação de um *cluster* de alta disponibilidade com o *software* de código aberto *Heartbeat*, utilizando apenas servidores Linux. Foram instalados no *cluster* os serviços Web e Proxy e efectuados diversos cenários de teste, avaliando o seu impacto no acesso do utilizador final, bem como a escalabilidade da solução e a aplicabilidade em ambientes empresariais.

O artigo está dividido em três tópicos: breve enquadramento dos conceitos de alta disponibilidade e principais meios para a alcançar; descrição da arquitectura *Heartbeat* e de alguns cenários implementados. Por fim apresentam-se as principais conclusões da realização do projecto [5], bem como alguns pontos que podem ser desenvolvidos tendo como base o trabalho elaborado.

2. Alta disponibilidade – conceitos e arquitectura

O bom funcionamento de uma empresa implica que os seus serviços estejam sempre operacionais e disponíveis aos utilizadores. No entanto, a ocorrência de paragens planeadas (e.g. substituição de um disco) ou não planeadas (e.g. desastres naturais) colocam em causa esse bom funcionamento, podendo implicar a indisponibilidade dos serviços para o utilizador final.

A disponibilidade é uma medida, calculada como sendo a percentagem de tempo que um determinado componente da arquitectura (e.g. disco, servidor) está operacional para o utilizador final, conforme se ilustra em (1).

$$\text{Disponibilidade (\%)} = \frac{\text{Total Tempo Disponível}}{\text{Total Tempo}} \qquad (1)$$

De uma forma mais específica, a disponibilidade do sistema pode ser calculada com base nos tempos de paragem e recuperação de um componente (2) ou serviço (3) da rede.

$$\text{Disponibilidade} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}} \quad , \qquad (2)$$

em que MTBF (*Mean Time Between Failure*) é o tempo médio de paragem entre falhas de um componente e MTTR (*Mean Time To Repair*) é o tempo médio de reparação desse componente.

$$\text{Disponibilidade} = \frac{\text{MTBSO}}{\text{MTBSO} + \text{MTTSR}} \quad , \qquad (3)$$

sendo MTBSO (*Mean Time Between Service Outage*) o tempo médio de interrupção de um serviço e MTTSR (*Mean Time To Service Repair*) o tempo médio de reparação desse serviço em caso de falha.

Surge, então, o problema de contornar a possível indisponibilidade de um serviço de forma a que esta seja mínima. A redundância de *hardware* nos equipamentos e nas suas ligações à rede pode ser uma solução. Outra solução passa pela construção de um *cluster* onde os serviços são fornecidos por um servidor principal. Um *cluster* é um conjunto de computadores independentes, designados de nós, que colaboram entre si para atingir um determinado objectivo comum, podendo classificar-se como de alta disponibilidade ou de alto desempenho. Aqueles asseguram boa escalabilidade, possibilitando a introdução de novos componentes no *cluster* (e.g. serviços, nós) de forma quase transparente para o utilizador final. A figura 1 apresenta um exemplo típico de uma arquitectura de *clustering* com dois servidores que partilham dados e têm serviços configurados. Os servidores podem classificar-se como sendo activo (também designado de principal) ou *standby* (secundário). O primeiro assegura prioritariamente a disponibilidade dos serviços configurados. O servidor de *standby* assume o papel de principal sempre que neste ocorre uma paragem. A comunicação entre os dois servidores é efectuada por mensagens de controlo, permitindo ao servidor de *standby* detectar a paragem do principal.

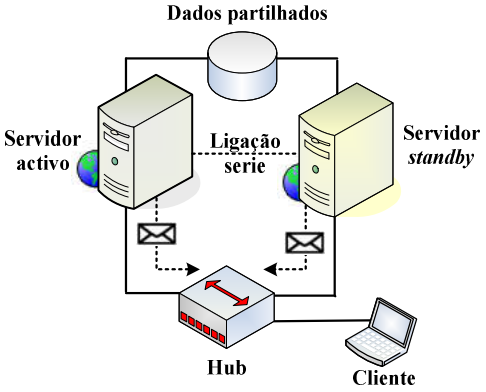

Fig. 1 - Exemplo de um *cluster* típico.

Para construir um *cluster* começam-se por eliminar os pontos críticos de falha (SPOF – *Single Point Of Failure*) de uma máquina individual. O primeiro passo consiste na redundância de *hardware* (e.g. discos, interfaces de rede) em todos os servidores pertencentes ao *cluster*. De seguida interligam-se os servidores em rede, onde também se incluam cuidados relativos à redundância das ligações. Por exemplo, recorrendo à utilização de duas interfaces de rede Ethernet que terminem em equipamentos activos distintos, ou implementando uma rede privada através de uma ligação série entre os servidores (e.g. SLIP). Desta forma assegura-se redundância ao nível físico, bem como ao nível protocolar, garantindo a comunicação IP sobre dois protocolos de ligação distintos.

Para garantir a gestão e funcionamento do *cluster*, nomeadamente a eleição dos servidores, a troca de mensagens de controlo e a implementação de procedimentos de paragem e arranque das aplicações, é necessário instalar um *software* de *clustering* como o *Heartbeat*, descrito na secção seguinte.

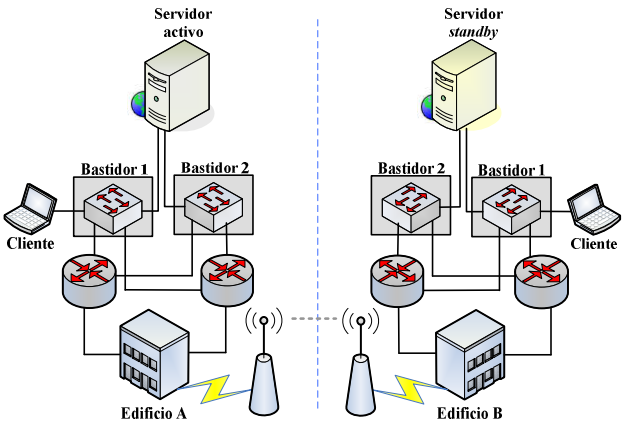

Fig. 2 - Exemplo de um *cluster* com alto nível de redundância.

A figura 2 ilustra um nível de redundância considerável, instalando os servidores e equipamentos de rede em locais distintos, nomeadamente em bastidores, edifícios ou cidades diferentes. Neste caso é necessário considerar uma ligação alternativa de alto débito que garanta a comunicação entre os servidores. Este tipo de soluções é de custo considerável e apenas implementado por *software* de *clustering* específico (e.g. HAGEO [6]).

3. Arquitectura *Heartbeat*

O *Heartbeat* é um *software* de *clustering*, associado a uma licença GNU [7], de código aberto e que efectua, entre outras funções, a gestão do *cluster* e configuração de automatismos de paragem e arranque das aplicações. Os nós do *cluster* enviam periodicamente mensagens UDP de *keepalive* por todas as interfaces de rede que estejam configuradas. Considerando um *cluster* com dois nós, cada um espera receber, num intervalo de tempo configurado (normalmente 2 segundos), mensagens de *keepalive* do outro nó. Ou seja, o nó principal aguarda por mensagens do secundário e vice-versa.

Se um dos nós não receber, em tempo útil, a mensagem de *keepalive* esperada, iniciará um conjunto de acções relacionadas com a sua função no *cluster*. Ou seja, se o nó secundário não receber o *keepalive* do principal, aquele inicia um conjunto de acções conducentes à sua passagem a nó principal, designadamente o arranque dos serviços configurados no *cluster*. A alteração da função de um nó no *cluster* (e.g. de secundário para principal) designa-se de *takeover*, que deve decorrer de forma transparente para o utilizador final, implicando o menor tempo de paragem possível dos serviços.

Os serviços disponibilizados por um *cluster* são associados a um grupo de recursos (e.g. aplicações, endereços IP, sistemas de ficheiros), designado por *ResourceGroup*, que é configurado em todos os nós do *cluster*, mas activo em cada momento apenas no nó principal. Em caso de falha do nó principal, esse grupo é activado no nó secundário, passando este a desempenhar o papel de nó principal.

A figura 3 apresenta as alterações de configuração dos nós do *cluster* após a ocorrência de um *takeover*. Neste caso, o acesso pelo cliente efectua-se pelo endereço IP 192.168.232.60, designado de virtual.

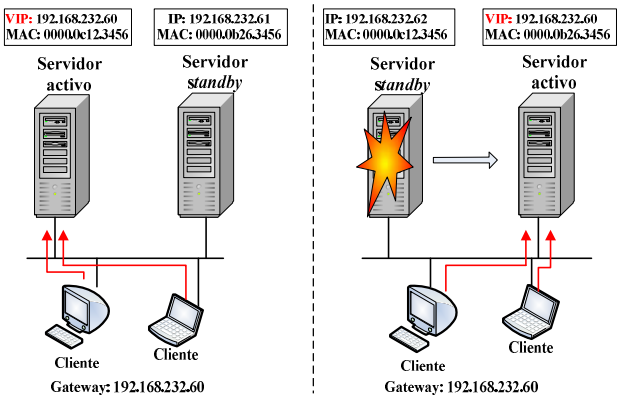

Fig. 3 – Esquema da ocorrência de um *takeover*.

O *software Heartbeat* faz parte de um projecto mais abrangente designado de "Linux-HA Project" [8]. O *download* do ficheiro binário específico para instalação encontra-se disponível no *site* do projecto. A instalação assenta no seguimento de passos simples descritos em [9]. A sua instalação e configuração são efectuadas, de uma forma simples, em todos os nós do *cluster*. A configuração da informação inicial (*input*) é definida em quatro ficheiros, nos quais se definem os parâmetros necessários para a comunicação entre os nós. Na figura 4 apresentam-se as principais directivas dos ficheiros de configuração do nó principal. No nó secundário (de *standby*) modifica-se unicamente a linha 1 do ficheiro `ha.cf` para "`ucast eth0 192.168.85.2`". Os restantes ficheiros mantém-se inalteráveis.

ha.cf

```
1 ucast eth0 192.168.85.1
2 auto_failback off
3 node servidor1
4 node servidor2
```

haresources

```
1 servidor1 192.168.85.3 httpd
```

authkeys

```
1 auth 1
2  1 crc
```

hosts

```
1 192.168.85.1 servidor1
2 192.168.85.2 servidor2
```

Fig. 4 – Principais directivas de configuração no nó principal.

A linha 1 do ficheiro `ha.cf` define o tipo de comunicação (`unicast`) e as linhas 3 e 4 o *hostname* de cada um dos nós do *cluster*. O parâmetro configurado na linha 2, `auto_failback`, permite definir qual o servidor que fica responsável pela disponibilização do grupo de recursos, depois da reparação de uma eventual falha no servidor principal. A activação ("on") deste parâmetro define que os serviços serão sempre disponibilizados pelo servidor principal. Ou seja, o servidor de *standby* apenas mantém os serviços activos enquanto o principal estiver parado. Logo que o servidor principal se torna operacional, assume imediatamente a sua função, iniciando os serviços. Por outro lado, um valor "off" no parâmetro indica que, em caso de falha do servidor principal, é efectuada a rotatividade entre os servidores do *cluster*, já que a posse dos recursos é alternada entre eles. A vantagem da activação deste parâmetro é particularmente visível se o servidor secundário possuir menor capacidade de processamento que o principal. Neste caso, torna-se necessário que o servidor secundário apenas assegure os recursos pelo menor período de tempo.

Os recursos (serviço Web e endereço IP virtual) a disponibilizar pelo *cluster*, juntamente com o servidor principal (`servidor1`), são definidos no ficheiro `haresources`. O ficheiro `authkeys` é utilizado para configurar a comunicação segura no *cluster*. O ficheiro `hosts` identifica os servidores existentes na rede.

Após a instalação e configuração do *Heartbeat* é necessário configurar o *cluster*, atribuindo os endereços IP definidos nos ficheiros de configuração dos respectivos servidores.

Para arranque ou paragem do *Heartbeat* basta exectuar o comando `service heartbeat start/stop` em cada servidor. Para automatizar o arranque, deve colocar-se o comando `service heartbeat start` no ficheiro de

inicialização do sistema (no caso da distribuição Mandrake 10.0, no ficheiro `rc.sysinit`). Desta forma a reinicialização é efectuada automaticamente sempre que o servidor arrancar. Toda a actividade do *Heartbeat* nos nós do *cluster* fica guardada em ficheiros de *log*, onde são especificadas todas as operações de comunicação e ocorrência de *takeovers*.

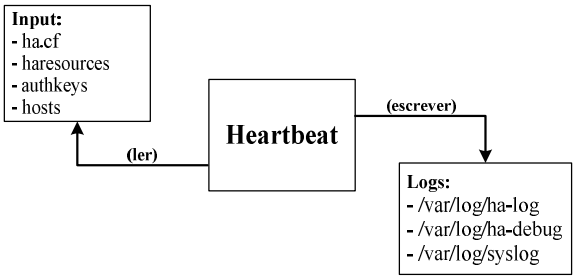

Fig. 5 – Ficheiros usados na arquitectura *heartbeat*.

A figura 5 esquematiza o fluxo de informação utilizada pelo *Heartbeat*, nomeadamente os ficheiros de *input* e *output* usados.

4. Implementação

Os cenários implementados pretenderam testar a fiabilidade e eficiência do *Heartbeat*, configurando-se fisicamente os mais simples e virtualmente os mais complexos, recorrendo ao utilitário VMware® [10]. Nestes cenários são implementados diferentes *clusters*, com dois ou três servidores interligados através de um *hub*, disponibilizando um ou dois serviços (Web e Proxy). Para efectuar os testes ligou-se ao *hub* um terminal cliente (PC).
Os dois principais cenários apresentados simulam um *simple takeover*, onde o servidor em *standby* adquire um grupo de recursos pertencente a um *cluster*, e um *dual takeover*, onde o servidor em *standby* adquire dois grupos de recursos referentes a dois *clusters* distintos. No primeiro cenário (fig. 6), testado fisicamente, o *cluster* engloba dois servidores que disponibilizam o serviço Web e que estão ligados a um cliente através de um *hub*. Depois de efectuadas as configurações básicas de rede para a ligação entre eles, foram configurados os ficheiros referentes ao *Heartbeat*. Além de especificado o modo de comunicação *unicast* entre os servidores, é também necessário definir o serviço a disponibilizar (Web) e associar um endereço IP virtual de acesso ao *cluster*.

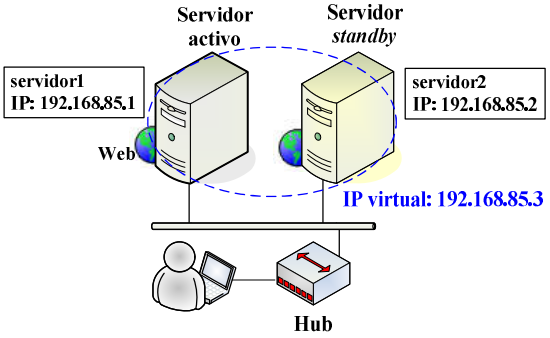

Fig. 6 - *Simple takeover*.

O cliente acede ao serviço Web disponibilizado recorrendo a um *browser*, onde insere o endereço IP virtual associado ao serviço. Neste caso, no *browser* é colocado o recurso "http://192.168.85.3". O primeiro teste foi realizado ao nível da ligação entre os servidores. Ao desligar esta ligação, por exemplo, retirando o cabo de rede, o servidor em *standby* assume que o servidor activo se encontra inoperacional e activa os recursos configurados (neste caso, o serviço Web). Do ponto de vista do cliente, o impacto é mínimo, havendo apenas paragem do serviço durante escassos segundos, relativos à ocorrência do *takeover*.
No entanto, ao restabelecer a ligação, ambos os servidores possuem os recursos activos. Este evento remete para o conceito de "síndrome de cérebro dividido" [11], incomportável num *cluster*. As soluções para resolver o problema passam pela implementação do método STONITH (*Shoot The Other Node In The Head*) [5] ou a utilização de uma ligação alternativa para o envio de mensagens de *keepalive*.
Numa segunda fase foi testada uma possível falha no servidor principal, com um cliente a utilizar o serviço, simulando uma paragem não planeada. Neste caso, o servidor secundário efectuou o *takeover* dos recursos (serviço Web e endereço IP virtual) de forma transparente para o cliente, visto que este mantém o acesso ao serviço.
O segundo cenário (figura 7) foi testado virtualmente, recorrendo ao VMware® com a implementação de dois *clusters*. Cada um possui dois servidores: um servidor activo e um em *standby*. No entanto, uma dessas máquinas foi configurada para desempenhar a função de servidor em *standby* para os dois *clusters*. Sendo assim este cenário é constituído por apenas três servidores. Cada um dos *clusters* possui um grupo de recursos associado, constituído pelo serviço disponibilizado (Web ou Proxy) e endereço IP virtual respectivo. Em caso de falha de um deles, ou de ambos, o servidor em *standby* efectua o *takeover* do(s) grupo(s) de recursos em falha garantindo a continuidade do(s) serviço(s) para o cliente.

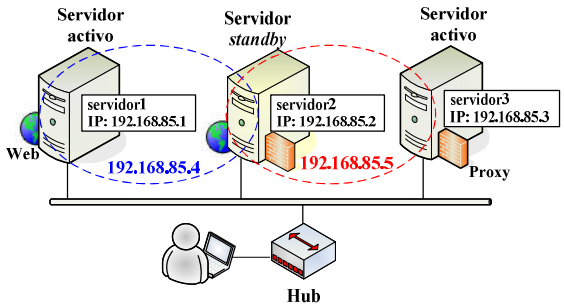

Fig. 7 - *Dual takeover*.

Os testes efectuados neste cenário baseiam-se nos mesmos princípios do cenário anterior. Através da falha não planeada dos servidores principais, foi testado o *takeover* dos recursos por parte do servidor secundário, resultando num impacto mínimo no acesso do cliente aos recursos.
Para uma visão mais realista do que acontece na maioria das empresas em termos de armazenamento de dados, foi também implementado um cenário que engloba um *file server* utilizando o protocolo NFS (*Network File System*)

[12], um servidor Web e um servidor em *standby*. Desta forma, os servidores acedem alternadamente aos dados partilhados, neste caso páginas Web, aos quais o cliente acede de forma transparente.

5. Conclusão

A maioria das soluções de alta disponibilidade existentes nas empresas assentam em arquitecturas proprietárias, com custos elevados de aquisição, instalação e manutenção. Este trabalho pretendeu demonstrar que é possível implementar uma solução de alta disponibilidade de código aberto e de baixo custo.

Os *clusters* implementados e testados, embora de uma forma simplista, permitem ilustrar o impacto desta arquitectura em empresas que pretendam implementar uma solução de alta disponibilidade. Embora os testes tenham incidido sobre os serviços Web e Proxy, outros poderão ser incluídos no *cluster*, tendo em conta as necessidades da organização. Há, no entanto, duas condições de aceitação importantes: as aplicações que os suportam têm necessariamente de ter automatismos (*scripts*) bem definidos de arranque e paragem; essas mesmas aplicações deverão garantir a coerência e integridade dos dados após uma falha inesperada do servidor (e.g. as base de dados).

Remetendo as soluções apresentadas a um nível empresarial ou académico, conclui-se que a disponibilidade dos serviços ao utilizador é garantida de uma forma simples e eficaz.

Este projecto permite continuidade no que diz respeito ao aumento da redundância aplicada aos cenários e à implementação de soluções para balanceamento de carga. A gestão e monitorização dos *clusters* recorrendo a utilitários com uma interface gráfica *user friendly* podem constituir um tópico de melhoramento ao trabalho. O *software Heartbeat* encontra-se em actualização permanente, prevendo-se versões recentes com novas funcionalidades, nomeadamente a possibilidade de definir *clusters* com mais de dois servidores, recorrendo a um algoritmo "N+1". Este melhoramento permitirá integrar as funções de *backup* e balanceamento de carga utilizando o mesmo *software* de *clustering*.

Agradecimentos

Agradecemos a colaboração de todos os docentes que durante o curso nos prepararam para enfrentar este desafio. Os nossos agradecimentos são também dirigidos a todos os colegas que, directa ou indirectamente, colaboraram para a realização e sucesso deste projecto.

Referências

[1] RAID Levels, www.acnc.com/raid.html

[2] Evan Marcus, Hal Stern, "Blueprints for High Availability", Wiley, 2ª Edição, 2003

[3] High Availability on the RISC System/6000 Family, IBM Corporation, International Technical Support Organization, Austin Center, Texas, 1ª Edição, Outubro 1995 (www.redbooks.ibm.com/redbooks/pdfs/sg244551.pdf)

[4] HP.com – High Availability – HP Serviceguard for Linux – Overview & Features, http://h18022.www1.hp.com/solutions/enterprise/highavailability/linux/serviceguard/index.html

[5] Filipa Ferreira, Nélia Santos, "*Clusters* de alta disponibilidade – abordagem OpenSource", Relatório desenvolvido na disciplina de Projecto I, Julho 2005

[6] Introduction to HAGEO, www.matilda.com/hacmp/hageo_introduction.html

[7] The GNU Operating System, www.gnu.org

[8] Linux-HA Project Web Site, www.linux-ha.org

[9] GettingStarted: Linux HA, www.linux-ha.org/GettingStarted

[10] VMware – Virtual Infrastructure Software, www.vmware.com

[11] R. Carr, The split brain syndrome, Private Communication, 1990

[12] The Network File System, www.tldp.org/LDP/nag/node140.html