Elton Martins Levita

Alta Disponibilidade como Alternativa ao Uso de Servidores BDC em Ambientes Samba

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Ricardo Martins De Abreu Silva

Lavras
Minas Gerais - Brasil
2005

Elton Martins Levita

Alta Disponibilidade como Alternativa ao Uso de Servidores BDC em Ambientes Samba

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *17 de Abril de 2005*

Prof. Joaquim Quinteiro Uchôa

Prof. Samuel Pereira Dias

Prof. Ricardo Martins De Abreu Silva
(Orientador)

Lavras
Minas Gerais - Brasil

Este trabalho é dedicado à toda comunidade Software Livre

Agradecimentos

À toda equipe do curso ARL, pela iniciativa de disseminar o *Software* Livre e compartilhar o conhecimento. Ao Professor Joaquim Uchôa por escrever a classe uflamon e apoiar a nós, alunos, em nossa caminhada. Aos meus amigos e familiares que sempre me incentivaram e me deram força nos momentos em que foi preciso. À toda comunidade *Software* Livre no mundo, pelo fantástico trabalho que vêm desenvolvendo e pelo espírito de colaboração mútua.

A todos vocês, muito obrigado.

Resumo

Atualmente a procura por servidores Samba como controlador de domínio (PDC - *Primary Domain Controller*) e servidor de arquivos nas empresas, principalmente pequenas e médias, é elevada.

Um possível tempo de inatividade desses servidores para essas empresas proporcionaria prejuízo e queda na produtividade.

Para garantir a disponibilidade do controlador de domínio, os servidores Samba implementam uma estrutura de servidor backup (BDC - *Backup Domain Controller*) que assume o controle do domínio automaticamente, na ocasião de falha do servidor principal. Porém, o servidor BDC garante disponibilidade do serviço controlador de domínio, mas para garantir a disponibilidade do servidor de arquivos há ainda a necessidade de fazer o sincronismo dos arquivos disponibilizados e das configurações dos diretórios compartilhados, além da base de usuários e senhas desses servidores.

Esse sincronismo pode ser feito através do *software rsync*[1], configurado para transferir os dados entre os servidores a cada intervalo de tempo especificado. O problema é que, caso ocorra uma falha, os arquivos que estarão no servidor que assumirá a função do principal, não possuirão as alterações feitas entre a última transferência e o momento da falha.

A proposta deste trabalho é estudar e implementar uma solução de alta disponibilidade e baixo custo, baseada em *software* livre que possa ser usada como alternativa a redundância oferecida pelo BDC.

[1] http://www.samba.org/rsync

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Esse trabalho tem por objetivo implementar uma solução de alta disponibilidade para servidores Samba, voltada para pequenas e médias empresas. Essa solução deve ser baseada em *software* livre e buscar atender os seguintes quesitos:

baixo custo;

independência de *hardware* específico;

permitir interrupção para manutenções planejadas;

tolerância a falhas do sistema;

tolerância a falhas de comunicação.

Esse trabalho foi motivado pelo aumento da utilização do Samba como controlador de domínio e servidor de arquivos nas pequenas e médias empresas, onde há a necessidade de garantir a disponibilidade dos serviços; e devido ao fato dos servidores BDC oferecerem contingência apenas para serviço controlador de domínio, não garantindo disponibilidade do servidor de arquivos.

O Capítulo 2 apresenta uma breve abordagem sobre alta disponibilidade; o Capítulo 3 aborda a implementação de um pequeno *cluster* de alta disponibilidade; o Capítulo 4, por sua vez, mostra os testes da solução; e o Capítulo 5 traz as considerações finais.

Capítulo 2

Alta disponibilidade

Cada vez mais é necessário garantir a disponibilidade de um serviço, mas praticamente todos os sistemas de informação atuais possuem componentes que estão sujeitos a falhas, podendo causar a paralisação desse sistema. O tempo dessa paralisação pode variar, dependendo da estrutura e da política adotada para o restabelecimento do serviço.

Para garantir a ausência de interrupções do serviço é necessário, muitas vezes, dispor de *hardware* redundante que entre em funcionamento automaticamente caso ocorra a falha de um dos componentes em utilização. Quanto mais redundância existir, menores serão os pontos vulneráveis desse sistema, e menor será a probabilidade de interrupções no serviço.

Até há poucos anos, sistemas desse tipo eram muito dispendiosos, o que intensificou uma procura em soluções alternativas. Surgem então sistemas de alta disponibilidade construídos com *hardware* acessível, altamente escaláveis e de custo mínimo, também conhecidos como *cluster*s de alta disponibilidade.

Um *cluster* de alta disponibilidade, pode ser definido como um sistema montado com mais de um computador, cujo objetivo é prover *hardware* redundante e eliminar pontos únicos de vulnerabilidade, fazendo com que, na ocasião de uma falha, um outro *hardware* assuma automaticamente as funções ou serviços que foram afetados no computador principal. A idéia é que, não haja um único ponto nesta arquitetura que, ao falhar, implique a indisponibilidade de outro ponto qualquer. Cada computador que integra um *cluster* é também conhecido como nó desse *cluster*.

Os *cluster*s de alta disponibilidade são projetados para que, o tempo de suas paralisações seja reduzido ao mínimo, e geralmente abrigam aplicações críticas, cuja paralisação implicaria em algum tipo de prejuízo ao negócio da empresa em questão.

A tolerância a falhas consiste, basicamente, em ter *hardware* redundante que entra em funcionamento automaticamente após a detecção de falha do *hardware* principal. Independentemente da solução adotada, existe sempre um tempo médio de recuperação, que é o espaço de tempo médio entre a ocorrência da falha e a total recuperação do sistema ao seu estado operacional.

A Tabela 2.1 mostra um dos conceitos geralmente usado na avaliação de soluções de alta disponibilidade — níveis de disponibilidade segundo tempos de indisponibilidade (*downtime*), também conhecido como escala de noves.

Tabela 2.1: Níveis de disponibilidade segundo tempos de indisponibilidade

Disponibilidade (%)	Downtime/ano
97%	10 dias, 22 horas e 48 minutos
98%	7 dias, 7 horas e 12 minutos
99%	3 dias, 15 horas e 36 minutos
99,9%	8 horas e 46 minutos
99,99%	52 minutos
99,999%	5 minutos

Fonte: http://compnetworking.about.com/library/weekly/aa092400a.htm

Geralmente, quanto maior a disponibilidade, maior a redundância e custo das soluções. O nível de disponibilidade a ser adotado, irá depender do tipo de serviço que se pretende disponibilizar.

Capítulo 3

Implementando alta disponibilidade no servidor Samba

O primeiro passo a ser tomado para implementar um sistema de alta disponibilidade é identificar os pontos únicos de vulnerabilidade. Segundo (MILZ, 1998), praticamente todos os componentes de um servidor estão sujeitos a apresentar falhas: a CPU (*Central Processing Unit* — Unidade Central de Processamento), a memória principal, a fonte de alimentação, a placa-mãe, a interface de rede, etc.

Uma placa de rede, por exemplo, pode ter sua redundância implementada com a instalação de uma segunda placa de rede no mesmo servidor, sendo esta ativada na ocasião de falha da primeira. A CPU, a fonte de alimentação e outros componentes de *hardware* também podem ser redundantes em um servidor, mas isso exige *hardware* específico e torna o servidor significativamente mais caro. Entretanto, para garantir a redundância desses componentes, pode ser usado um segundo servidor, configurado junto ao primeiro em um *cluster* de alta disponibilidade que, com a ajuda de componentes de *software*, permitirá a um nó assumir a função do outro que tenha se tornado indisponível.

Um ponto importante a ser observado para que haja alta disponibilidade nesse ambiente é que, para o nó que está em espera poder assumir a função do servidor principal, ele deve possuir todos os dados que estão disponíveis neste, e esses dados devem contemplar todas as alterações que por ventura tenham sido realizadas no nó principal. Essa situação pode ser resolvida com o uso de uma unidade de armazenamento externa, compartilhada entre os dois servidores, como mostra (BLACKMON; NGUYEN, 2001). Assim o servidor de contingência, acessaria as mesmas informações acessadas e alteradas pelo servidor principal. Essa unidade

de armazenamento deveria possuir um sistema de redundância dos seus discos internos e também de sua conexão com os servidores, caso contrário, ela também seria um ponto único de vulnerabilidade do sistema. Porém, o custo de uma unidade de armazenamento como esta é alto, o que tornaria a solução consideravelmente mais cara, e geralmente implica no uso de *hardware* proprietário, fazendo com que a manutenção só possa ser realizada por seu respectivo fornecedor. Para resolver esse problema também será usado um componente de *software* para garantir que todos os dados gravados no disco do servidor principal, sejam também gravados no disco do servidor em espera.

3.1 *Hardware* Necessário

Neste trabalho, para implementar a alta disponibilidade no servidor Samba, são usados dois computadores. Cada um destes deve estar equipado com 2 interfaces de rede e uma interface serial. Para a interligação dos computadores entre si, são necessários um cabo serial *null modem* e um cabo Ethernet *crossover* CAT-5.

Um cabo *null modem* permite que dois dispositivos seriais (RS-232) se comuniquem sem a necessidade de modems ou outros dispositivos entre eles. Sua pinagem é mostrada na Tabela 3.1.

Assim como o cabo *null modem*, um cabo Ethernet *crossover* é usado para conectar duas interfaces de rede Ethernet sem que sejam necessários *switches* ou outros dispositivos entre elas. Sua pinagem é mostrada na Tabela 3.2.

Tabela 3.1: Pinagem do cabo *null modem*

Conector 1 (DB-9 fêmea)	Conector 2 (DB-9 fêmea)
FG (Frame Ground) -	- FG (Frame Ground)
TD (Transmit Data) 3	2 RD (Receive Data)
RD (Receive Data) 2	3 TD (Transmit Data)
RTS (Request To Send) 7	8 CTS (Clear To Send)
CTS (Clear To Send) 8	7 RTS (Request To Send)
SG (Signal Ground) 5	5 SG (Signal Ground)
DSR (Data Set Ready) 6	4 DTR (Data Terminal Ready)
CD (Carrier Detect) 1	4 DTR (Data Terminal Ready)
DTR (Data Terminal Ready) 4	1 CD (Carrier Detect)
DTR (Data Terminal Ready) 4	6 DSR (Data Set Ready)

Fonte: http://www.nullmodem.com

Tabela 3.2: Pinagem do cabo Ethernet *crossover*

Conector 1 (RJ45)	Conector 2 (RJ45)
TD+ 1	3 RD+
TD- 2	6 RD-
RD+ 3	1 TD+
4	4
5	5
RD- 6	2 TD-
7	7
8	8

Fonte: http://www.nullmodem.com

3.2 Conectando os nós do *cluster*

Visando obter alta disponibilidade, a ligação dos nós para formação do *cluster* deve ser redundante, evitando assim, um ponto único de vulnerabilidade. Para isso serão usadas duas conexões: uma através da interface serial, e outra através da interface Ethernet. Deve-se conectar o cabo *null modem* nas portas seriais dos dois computadores, e conectar o cabo Ethernet *crossover* em uma das interfaces de rede de cada computador, deixando a outra interface de rede para conexão com a rede local que irá acessar serviços nestes servidores.

Para que as interfaces Ethernet se comuniquem, deve-se configurar um endereço IP (*Internet Protocol*)[1] em cada uma dessas interfaces. Esses endereços devem pertencer a mesma subrede, como por exemplo 10.0.0.1/24 para o servidor1 e 10.0.0.2/24 para o servidor2.

A outra interface de rede de cada servidor será conectada rede local, por onde os clientes irão acessar os serviços disponibilizados. O endereço IP a ser configurado nesta interface deve estar na mesma subrede dos computadores clientes. Neste trabalho, será considerada a rede 192.168.0.0/24. Assim, foi atribuído o endereço 192.168.0.251 para o servidor1 e o endereço 192.168.0.252 para o servidor2. As estações clientes, porém, deverão ver esses dois servidores como um servidor único. Para isso elas irão acessar o servidor através de um outro endereço, o endereço do cluster, que será atribuído pelo software de gerenciamento do cluster que será abordado posteriormente. Neste trabalho, o endereço atribuído para o cluster foi 192.168.0.253.

A Figura 3.1 ilustra as conexões dos servidores do *cluster* entre si, e do *cluster* com a rede local.

[1] Protocolo utilizado para comunicação de dados na Internet

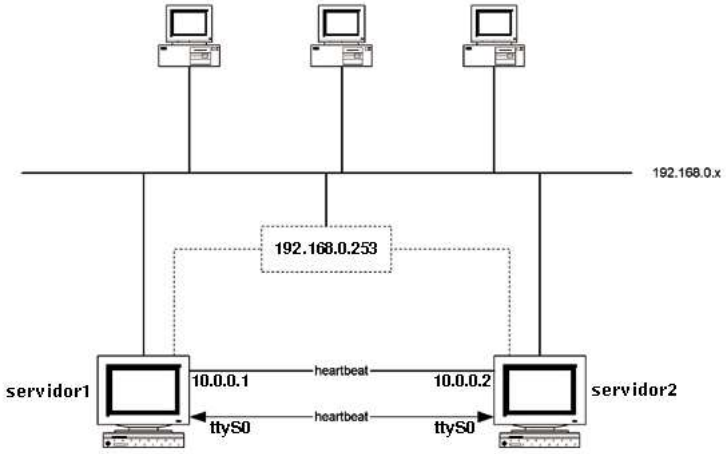

Figura 3.1: Cluster de alta disponibilidade com dois nós

3.3 *Software* Necessário

3.3.1 Samba

O *Samba*[2] é um *software* usado para integrar redes Linux e Windows. Ele é uma implementação do protocolo *CIFS*[3] que permite que sistemas Linux possam interagir com sistemas Windows em uma rede.

Através do Samba, é possível acessar, de uma estação de trabalho com o sistema Linux, serviços de arquivo, impressão, e de autenticação em domínio, oferecidos por servidores Windows. Ele também permite que um servidor Linux ofereça esses serviços de forma transparente para estações de trabalho rodando o sistema Windows. A instalação e configuração do servidor Samba, foge ao escopo deste trabalho. Para maiores informações sobre a instalação e configuração de servidores Samba, consultar (TS; ECKSTEIN; COLLIER-BROWN, 2003) e (VERNOOIJ; TERPSTRA; CARTER, 2003).

[2] http://www.samba.org

[3] *Common Internet File System* — é a evolução do protocolo *SMB (Server Message Block)*, usado em redes Windows

3.3.2 Heartbeat

O *Heartbeat*[4] é o *software* responsável pelo estabelecimento do *cluster* de alta disponibilidade. Através dele, um computador fica monitorando constantemente a disponibilidade do outro computador que compõe o *cluster*. Essa tarefa é feita, enviando-se sinais ao outro nó do *cluster* em um intervalo especificado, como por exemplo, a cada segundo. Caso haja uma falha no servidor principal, este deixará de enviar os sinais. Essa ausência de sinais, será interpretada pelo outro nó como indisponibilidade do servidor principal e ocorrerá então um *failover*[5].

3.3.3 DRBD

O *DRBD (Distributed Replicated Block Device)*[6] é o *software* que realiza a replicação de dados entre os nós do *cluster*, fazendo o espelhamento (*mirroring*) de um dispositivo de blocos através de uma conexão de rede dedicada. Ele pode ser entendido como um *RAID 1*[7] via rede, onde recebe os dados a serem gravados, escreve no disco local e os envia para o outro computador, onde os dados também são gravados no disco.

O DRBD cria um dispositivo de armazenamento que, na verdade, estará distribuído nos dois servidores que compõe o *cluster*. O dispositivo DRBD trabalha em uma camada superior aos dispositivos de bloco usados para armazenamento (disco local) dos computadores do *cluster*. Sempre que um nó gravar no dispositivo DRBD, as alterações serão gravadas no disco local, e serão replicadas para o outro nó em tempo real.

Cada dispositivo DRBD tem um estado, que pode ser primário ou secundário. O dispositivo primário fica no nó principal, onde a aplicação está executando e tem acesso ao dispositivo DRBD (`/dev/drbdX`). Todo acesso ao disco passa a ser feito através deste dispositivo. Quando é realizada uma operação de escrita, os dados são gravados no disco local e enviados para o nó com o dispositivo secundário. O secundário simplesmente escreve o dado no dispositivo da camada inferior, que neste caso, é o disco local do nó remoto. As operações de leituras são sempre realizadas no nó principal.

Para usar o DRBD, é recomendável que o sistema de arquivos a ser utilizado

[4] http://www.linux-ha.org/HeartbeatProgram

[5] Evento onde um nó do *cluster* assume os serviços de um outro nó que encontra-se indisponível.

[6] http://www.drbd.org

[7] Técnica onde dois discos são configurados como um único disco para sistema operacional, e os dados são gravados em ambos. Caso um disco falhe, os dados continuam disponíveis através do outro disco.

tenha suporte a *journaling*[8]. Caso contrário, quando o nó primário estiver indisponível e houver o *failover*, o *fsck*[9] deverá ser executado.

Se o nó que falhou retornar, ele se torna o secundário e tem que sincronizar seus dados com o primário. Isto é feito em *background*, sem interrupção do serviço.

Um cuidado que deve ser tomado é que, após configurado um dispositivo DRBD, não se deve tentar montar diretamente nenhum dos discos aos quais ele faz referência. O acesso aos dados deve ser feito através do dispositivo DRBD, caso contrário, corre-se o risco de haver corrupção dos dados e tornar todo o conteúdo do disco inacessível.

3.4 Configurando o Heartbeat

A configuração do Heartbeat é feita através de três arquivos: `ha.cf`, `haresources` e `authkeys`. Esses arquivos geralmente residem no diretório `/etc/ha.d`, mas dependendo da distribuição usada e das opções de instalação esse caminho pode variar.

Abaixo, são explicados os arquivos e suas opções. Uma abordagem mais aprofundada desses arquivos de configuração pode ser obtida em (ROBERTSON,).

3.4.1 O arquivo `ha.cf`

O arquivo `ha.cf` configura o funcionamento do Heartbeat. Suas opções estão descritas abaixo.

`debugfile`

Especifica o arquivo onde serão gravados os arquivos com informações para depuração de erros. O valor padrão é `/var/log/ha-debug`.

`logfile`

Indica o arquivo onde serão gravados os logs do Heartbeat. O valor padrão é `/var/log/ha-log`.

`logfacility`

[8] Também conhecido como *logging*, é uma técnica onde são gravadas as alterações realizadas aos dados em um arquivo, para que um estado anterior possa ser reconstruído se necessário.

[9] Ferramenta para verificar e corrigir erros no sistema de arquivos do Unix e de sistemas nele baseados.

Opção usada para que as mensagens de debug e de log sejam registradas através do uso do syslog. O valor padrão é `local0`. Para maiores informações sobre o syslog, consulte sua *man page*[10].

Se a opção logfacility não estiver definida, as mensagens de debug e de log serão gravadas nos arquivos definidos em `debugfile` e em `logfile`. Caso estes também não estejam definidos serão usados seus valores padrão.

`keepalive`

Configura o intervalo, em segundos, entre os sinais Heartbeat enviados. O valor padrão é `2`.

`deadtime`

Define o tempo, em segundos, que deve-se aguardar sem receber sinais Heartbeat, para considerar que um nó está indisponível. Se este valor for muito baixo, ocorrerá o problema de *split-brain*[11]. O valor padrão é `30`.

`warntime`

Define o tempo, em segundos, que deve-se esperar antes de gravar um alerta de sinal Heartbeat atrasado nos logs. O valor padrão é `10`.

`initdead`

Em alguns sistemas, a rede demora um pouco a entrar em funcionamento quando o computador é reiniciado. Para isso existe esse parâmetro, que atua como um `deadtime` quando o sistema está sendo iniciado. Seu valor deve ser pelo menos o dobro do valor definido em `deadtime`. O valor padrão é `120`.

`udpport`

Configura a porta UDP (*User Datagram Protocol*)[12] a ser usada quando a comunicação for configurada como `bcast` ou `ucast`. O valor padrão é `694`.

`baud`

Define a baud rate para as portas seriais. O valor padrão é `19200`.

[10]Em ambientes Linux, as *man pages* são manuais *on-line* de um *software*, que podem ser acessados através do comando `man`

[11]Também chamado de *cluster partition*, é o evento onde os dois nós do *cluster* identificam a indisponibilidade do outro nó e tentam assumir os serviços ao mesmo tempo.

[12]Um dos principais protocolos do conjunto de protocolos da Internet (TCP/IP)

`serial`

Indica o nome da porta serial a ser usada para envio de sinais Heartbeat. Para sistemas Linux, o valor padrão é `/dev/ttyS0`.

`bcast`

Indica em qual interface os sinais Heartbeat serão enviados em *broadcast*[13]. O valor padrão para sistemas Linux é `eth0`.

`mcast`

Configura a envio de sinais Heartbeat em multicast. Sua sintaxe é

```
mcast [dev] [mcast group] [port] [ttl] [loop]
```

onde

`[dev]` é o dispositivo usado para envia e receber sinais Heartbeat.

`[mcast group]` é o grupo multicast a ser associado (endereço multicast classe D — 224.0.0.0 – 239.255.255.255).

`[port]` é a porta UDP a ser usada.

`[ttl]` é o valor TTL (*Time To Live*)[14] para os sinais Heartbeat enviados (0 – 255). Ele define a quantidade de saltos que um pacote multicast será capaz de se propagar. Deve ser maior que 0.

`[loop]` define a interface loopback para envio dos pacotes Heartbeat (0 ou 1). Se estiver ativo, o pacote será recebido na mesma interface em que foi enviado. Geralmente, deve ser configurado para 0.

Por exemplo: `mcast eth0 225.0.0.1 694 1 0`

`ucast`

Configura o envio de sinais Heartbeat em *unicast*[15]. Sua sintaxe é:

```
ucast [dev] [peer-ip-addr]
```

onde

`[dev]` é o dispositivo usado para envia e receber sinais Heartbeat.

`[peer-ip-addr]` é o endereço IP do nó ao qual serão enviados pacotes Heartbeat.

Por exemplo: `ucast eth0 192.168.1.2`

[13] Envio de pacotes que serão recebidos por todos os dispositivos em uma rede.
[14] Limite de tempo que um pacote pode existir antes de ser descartado
[15] Envio de pacotes a um único computador destino em uma rede de computadores

`auto_failback`

Determina se deverá ocorrer um *failback*[16], ou se deixará o outro nó servindo até que ele falhe, ou até que o administrador intervenha. Os valores possíveis são: `on` — ativa o *failback* automático, `off` — desativa o *failback* automático, e `legacy` — habilita o *failback* automático em sistemas onde todos os nós ainda não suportam a opção `auto_failback`. O valor padrão é `legacy`.

`stonith`

Esta diretiva assume que existe um dispositivo STONITH(*Shoot The Other Node In The Head*)[17] no *cluster*. Os parâmetros desse dispositivo são lidos do arquivo de configuração. A sintaxe dessa linha é:

`stonith <stonith_type> <configfile>`

Por exemplo: `stonith baytech /etc/ha.d/conf/stonith.baytech`

`stonith_host`

Pode-se configurar vários dispositivos STONITH com essa diretiva. Sua sintaxe é:

`stonith_host <hostfrom> <stonith_type> <params...>`

onde

`<host_from>` é a máquina a qual o dispositivo STONITH está ligado. Se for usado `*` significa que o dispositivo é acessível por qualquer host.

`<stonith_type>` é o tipo do dispositivo STONITH

`<params...>` parâmetros específicos do driver. Para ver o formato para um dispositivo, deve-se rodar:

`stonith -l -t <stonith_type>`

`watchdog`

O watchdog é um módulo que monitora a própria máquina. Essa monitoração é feita através dos próprios sinais Heartbeat que ela envia. Se ela não conseguir detectar seus próprios sinais Heartbeat por um minuto, a máquina reinicia.

[16] Quando um recurso que retornou do estado de falha automaticamente assume sua posição de nó primário.

[17] Técnica que faz com que um nó que esteja com problema e que possa estar consumindo indiscriminadamente recursos compartilhados do *cluster* seja desativado.

Ao usar o watchdog, deve-se carregar o módulo com o parâmetro `nowayout=0` ou compilá-lo sem que a variável CONFIG_WATCHDOG_NOWAYOUT esteja definida. Caso contrário, mesmo uma paralisação programada do Heartbeat irá fazer a máquina reiniciar.

Por exemplo: `watchdog /dev/watchdog`

`node`

Essa diretiva indica que máquinas estão no *cluster*. Deve ser passado o nome das máquina conforme a saída do comando `uname -n`.

Exemplos:

```
node servidor1
node servidor2
```

`ping`

Este parâmetro é opcional e especifica os nós de ping que são usados para teste de conectividade de rede por outros módulos, como o Ipfail. O endereço IP informado será tratado como um pseudo-membro do *cluster*.

Por exemplo: `ping 10.10.10.254`

`ping_group`

Semelhante ao anterior, porém esse parâmetro reúne vários IP's em um pseudo-membro do *cluster* com o nome especificado. Assim se um dos endereços passado estiver ativo, o pseudo-membro estará ativo.

Por exemplo: `ping_group grupo1 10.10.10.254 10.10.10.253`

Se um dos hosts 10.10.10.254 ou 10.10.10.253 estiverem ativos, grupo1 será considerado ativo.

`respawn`

Indica os processos que serão monitorados e controlados pelo Heartbeat. Esses processos serão reiniciados a menos que terminem com código de retorno igual a 100. Sua sintaxe é:

```
respawn userid cmd
```

Por exemplo:

```
respawn hacluster /usr/lib/heartbeat/ipfail
```

Informa ao Heartbeat para iniciar o comando `ipfail` com as credenciais do usuário `hacluster` e monitorá-lo, reiniciando-o caso ele pare.

`hopfudge`

Opcional. Nas topologias em anel, configura o número de saltos permitidos além do número de nós no *cluster*.

`deadping`

Tempo, em segundos, que deve-se esperar resposta dos nós de ping antes de considerá-los indisponíveis.

`realtime`

Ativa ou desativa a execução em tempo real (alta prioridade). O valor padrão é `on`.

`debug`

Informa o nível depuração. O valor padrão é `0`.

`msgfmt`

Define o formato da mensagem usada pelo Heartbeat. Pode ser `classic` ou `netstring`. Se for configurado `classic`, a mensagem é enviada em formato de cadeia de caracteres e valores binários são convertidos em caractere com uso de uma biblioteca base64[18]. Se for configurado `netstring`, mensagens binárias serão enviadas diretamente. Este último é mais eficiente pois evita conversão entre binários e cadeias de caractere. O valor padrão é `classic`.

A Figura 3.2 mostra o arquivo `ha.cf` usado neste trabalho.

3.4.2 O arquivo `haresources`

O arquivo `haresources` define a lista de recursos que serão movidos de uma máquina para outra quando um nó ficar indisponível no *cluster*. Os recursos podem ser o endereço IP do *cluster*, ou os scripts localizados em `/etc/rc.d/init.d` e `/etc/ha.d/resource.d` para iniciar e parar serviços. O arquivo `haresources` deve ser idêntico em ambos os nós no *cluster*.

Cada linha no arquivo é considerada um *grupo de recursos*. Um grupo de recursos é uma lista de serviços que serão movidos de um nó para outro, na ocasião

[18] Sistema de numeração que representa a maior base de 2 que pode ser representada por caracteres ASCII (American Standard Code for Information Interchange — conjunto de caracteres baseado no alfabeto romano, usado para representar texto em sistemas de computação). Usa os caracteres A-Z, a-z e 0-9 nessa ordem para os 62 primeiros dígitos e os símbolos representados pelos 2 últimos dígitos podem variar entre diferentes sistemas.

```
logfacility local0
keepalive 1
deadtime 30
initdead 120
udpport 694
baud 19200
serial /dev/ttyS0
bcast eth0
auto_failback on
watchdog /dev/watchdog
node servidor1
node servidor2
ping 192.168.0.254
respawn hacluster /usr/lib/heartbeat/ipfail
deadping 30
```

Figura 3.2: Arquivo de configuração `/etc/ha.d/ha.cf`

de uma falha. Os recursos em um grupo são iniciados, na ordem em que estão listados, da esquerda para a direita, e terminados da direita para a esquerda. Assim, se um serviço depende de outro para funcionar, este último deve aparecer primeiro no grupo de recursos, ou seja, deve estar a esquerda do que depende dele na linha de configuração.

O nó listado no início da definição de grupo de recursos é preferido para rodar o serviço. Não é necessariamente o nome da máquina onde está o arquivo. Se estiver sendo usada a opção `auto_failback on` (ou `legacy`), esses serviços serão iniciados no nó preferido sempre que este estiver disponível. O nome do nó informado deve coincidir com a saída do comando `uname -n`.

O formato de uma linha no arquivo haresources é mostrado abaixo:

```
node-name resource1 resource2 ...  resourceN
```

Por exemplo:

```
servidor1 192.168.0.253 drbddisk smb
```

Informa que o serviço deverá ser iniciado preferencialmente no nó servidor1 com o endereço IP compartilhado 192.168.0.253 e os serviços a serem iniciados são o DRBD e o Samba, nessa ordem.

O endereço IP usado nesse arquivo, é o endereço do *cluster*, ou seja, o ende-

```
servidor1 IPaddr::192.168.0.253/24/eth1 drbddisk smb
```

Figura 3.3: Arquivo de configuração `/etc/ha.d/haresources`

reço através do qual os clientes irão acessar o *cluster* de alta disponibilidade. Esse endereço será direcionado a interface de rede que possuir rota configurada para o endereço dado. A máscara de subrede para este endereço IP será a mesma configurada na rota referida anteriormente, e o endereço de *broadcast* será o endereço mais alto da subrede.

Quando for necessário passar parâmetros a um recurso, isso pode ser feito usando-se o delimitador `::` entre o recurso e o parâmetro.

No caso do endereço IP, o nome do recurso chama-se `IPaddr`, e pode ser omitido. Mas se for necessário especificar uma máscara de rede, endereço de *broadcast* ou interface diferentes, pode-se passar parâmetros ao *script* `IPaddr` usando o delimitador.

Por exemplo, para especificar máscara de subrede, interface e endereço de *broadcast* a linha a ser usada seria:

```
servidor1 IPaddr::192.168.0.253/24/eth1/192.168.0.220
drbddisk smb
```

A Figura 3.3 mostra o arquivo `haresources` usado neste trabalho.

3.4.3 O arquivo `authkeys`

O arquivo `authkeys` é usado para configurar a autenticação do Heartbeat. Existem três métodos de autenticação disponíveis: `crc`, `md5` e `sha1`.

O método `sha1` é o mais seguro e robusto, porém apresenta maior consumo de CPU que os demais. Ele é o mais indicado quando for usada uma rede insegura e não se está preocupado com o uso de CPU. O método `md5` é o segundo no ponto de vista da segurança, mas pode ser útil em redes onde seja requerida alguma segurança e queira-se minimizar o uso de CPU. O método `crc` é o mais inseguro e o mais econômico do ponto de vista de recursos. É indicado em redes fisicamente seguras, como por exemplo uma conexão direta através de um cabo Ethernet *crossover*.

O formato do arquivo é o mostrado a seguir:

```
auth <id>
<id> <authmethod> [<authkey>]
```

onde

```
auth 1
1 crc
```

Figura 3.4: Arquivo de configuração `/etc/ha.d/authkeys`

`<id>` é o identificador do método a ser usado (1 — `crc`, 2 — `sha1`, 3 — `md5`).

`<authmethod>` é o método usado (`crc`, `sha1`, `md5`).

`<authkey>` é a chave para autenticação.

Para o método `crc`, não é necessário usar chave. Um exemplo é mostrado abaixo:

```
auth 1
1 crc
```

Para `sha1`, tem-se o exemplo:

```
auth 2
2 sha1 little-text
```

Finalmente, para md5, tem-se:

```
auth 3
3 md5 other-text
```

A Figura 3.4 mostra o arquivo `authkeys` usado neste trabalho.

É necessário alterar as permissões do arquivo `authkeys` para permitir leitura apenas ao usuário `root`. Caso contrário o Heartbeat não irá executar.

3.4.4 Configurando o Ipfail

O *plugin Ipfail* realiza a detecção de falhas de conectividade na rede, para então reagir de maneira inteligente. Por exemplo, se a interconexão entre os dois nós do *cluster* estiver funcional, mas a conectividade do servidor principal com a rede por onde os clientes acessam os serviços estiver comprometida, o Ipfail, após identificar que a máquina não está se comunicando com essa rede, pode forçar um *failover* para que os clientes possam acessar os serviços no outro nó. Para fazer isso, o Ipfail usa nós de ping, configurados no arquivo `ha.cf` através das diretivas `ping` e `ping_group`.

É essencial que nós de ping estratégicos sejam escolhidos para o teste de conectividade. Deve-se escolher equipamentos de rede como *switches* e roteadores e evitar usar outros membros do *cluster* ou alguma estação de trabalho da rede como nós de ping. Os nós de ping escolhidos, devem refletir ao máximo a conectividade da rede que se deseja monitorar.

Tabela 3.3: Protocolos de replicação de dados do DRBD

C	A gravação é considerada completa ao gravar o dado no disco local e no disco remoto.
B	A gravação é considerada completa ao gravar o dado no disco local e confirmar sua recepção pelo host remoto.
A	A gravação é considerada completa ao gravar o dado no disco local e envia-lo para host remoto.

O Ipfail só irá operar se o parâmetro `auto_failback` estiver diferente de `legacy` no arquivo `ha.cf`. Deve-se configurar esse parâmetro como `on` ou `off` para que o Ipfail funcione.

3.5 Configurando o DRBD

A configuração do DRBD é feita através do arquivo `drbd.conf`, geralmente localizado no diretório `/etc`.

Esse arquivo é formado por, no máximo uma seção `global`, e uma ou mais seções `resource`.

Na seção global pode-se especificar, através da opção `minor-count`, quantos dispositivos DRBD se deseja configurar. Essa opção é útil caso seja necessário definir mais recursos depois, sem precisar recarregar o módulo, evitando a interrupção do serviço.

Cada seção `resource` se subdivide em configurações agrupadas em partes como `startup`, `disk`, `net` e `syncer`, e em configurações específicas do nó, que são agrupadas em subseções denominadas `on [hostname]`.

Um exemplo do arquivo `drbd.conf` é mostrado na Figura 3.5. O significado de cada item é descrito em seguida. Uma abordagem mais detalhada do arquivo `drbd.conf` pode ser obtida em (ELLENBERG, b) e (ELLENBERG, a).

`minor-count`

Usado quando se deseja definir mais recursos posteriormente, sem precisar reiniciar o módulo. Por padrão, o módulo é carregado com a quantidade exata de recursos definidos no arquivo `drbd.conf`.

`protocol`

É o protocolo de transferência de dados utilizado. Ele pode ser `A`, `B` ou `C` conforme a Tabela 3.3.

`device`

Indica o dispositivo a ser usado. Normalmente `/dev/drbd0`.

```
global {
  minor-count 5;
}
resource r0 {
  protocol C;
  on hostname1 {
    device /dev/drbd0;
    disk /dev/sda3;
    meta-disk internal;
    address 192.168.77.1:7788;
  }
  on hostname2 {
    device /dev/drbd0;
    disk /dev/hdc;
    meta-disk internal;
    address 192.168.77.2:7788;
  }
}
```

Figura 3.5: Exemplo de arquivo de configuração `/etc/drbd.conf`

`disk`

O disco físico (camada inferior) a ser usado.

`address`

Endereço e porta usados para se conectar ao nó.

`meta-disk`

Dispositivo usado para armazenar os meta-dados do DRBD. Pode-se usar o mesmo dispositivo para armazenar meta-dados de diferentes dispositivos DRBD passando-se um índice. Por exemplo pode-se usar `meta-disk /dev/hdb1[0]` e `meta-disk /dev/hdb1[1]` para dois recursos distintos. Para isso, o disco `/dev/hdb1` deve possuir 128 MB para cada recurso. Nessa opção pode-se usar também o valor `internal`, que faz com que os últimos 128 MB do dispositivo físico sejam usados para armazenar os meta-dados. Configurando esse parâmetro com o valor `internal`, não deve ser usado o índice.

A Figura 3.6 mostra o arquivo `drbd.conf` usado neste trabalho.

```
resource r0 {
  protocol C;
  on servidor1 {
    device /dev/drbd0;
    disk /dev/hda2;
    meta-disk internal;
    address 10.0.0.1:7788;
  }
  on hostname2 {
    device /dev/drbd0;
    disk /dev/hda2;
    meta-disk internal;
    address 10.0.0.2:7788;
  }
}
```

Figura 3.6: Arquivo de configuração `/etc/drbd.conf`

Capítulo 4

Testando a solução

Os testes da solução apresentada foram realizados em ambiente doméstico com dois computadores cuja configuração básica é mostrada na Tabela 4.1.

Tabela 4.1: Características dos nós usados

	Nó principal	Nó secundário
CPU	AMD Athlon 2000	AMD Duron 900
Memória	256 MB DDR-SRDAM 266 MHz	256 MB DDR-SRDAM 266 MHz
Disco	40 GB ATA-133	40 GB ATA-133

Para testar a solução, com ambas as máquinas em funcionamento foi feito um acesso a um dos compartilhamentos disponibilizados no servidor Samba usando o endereço IP do *cluster* (192.168.0.253), e em seguida, foram copiados vários arquivos para este compartilhamento.

Após o término da cópia de arquivos foi feita uma conexão *ssh*[1] para o endereço IP do *cluster* afim de identificar em qual máquina o Samba estava sendo executado, e foi constatado que a máquina era o servidor1.

Foi então parado o serviço Heartbeat no servidor1 afim de provocar uma falha e verificar se o *failover* ocorreria com sucesso. Após aproximadamente 30 segundos o serviço Samba estava disponível novamente. Para confirmar se o serviço estava executando no outro nó, foi feita uma conexão ssh, novamente usando o endereço IP do *cluster*, e identificado que o servidor que estava respondendo era o servidor2, o que significa que o *failover* foi bem sucedido.

Com o intuito de checar se houve a replicação de arquivos para o segundo nó, foi feito um novo acesso ao servidor Samba através do IP do *cluster*, e constatado

[1] *Secure Shell* — permite acessar um shell remotamente, através de uma conexão criptografada.

que os arquivos copiados para o servidor1 estavam disponíveis no servidor2, que neste momento respondia pelo *cluster*.

Em seguida, para testar o *failback*, foi reativado o serviço Heartbeat no servidor1 e, como o *cluster* estava configurado com a opção `auto_failback` ativada, novamente foi feito um *failover* devolvendo o serviço para o servidor1.

Para finalizar os testes e verificar se a solução estava tolerante a falhas de conectividade, foi desconectado o cabo de rede que ligava o servidor1 a rede local, por onde os clientes acessavam os serviços disponibilizados. Após pouco mais de 30 segundos, ocorreu um *failover* provocado pelo componente Ipfail, para que os clientes pudessem acessar o serviço através do outro nó.

Capítulo 5

Considerações finais

Após o trabalho realizado, constatou-se que é possível, através do uso de alguns componentes de *software* livre, implementar uma solução de alta disponibilidade e de baixo custo para servidores Samba. O produto resultante foi capaz de suprir a necessidade de sincronismo de arquivos deixada pelo uso de um servidor BDC, quando um mesmo servidor Samba realiza as funções de controlador de domínio e servidor de arquivos. A solução implementada atendeu os quesitos:

baixo custo

: Foram usados equipamentos simples, de baixo custo, e comumente encontrados no mercado.

independência de *hardware* específico

: O sistema não faz uso de nenhum hardware redundante. A alta disponibilidade foi obtida usando-se dois computadores para que não houvesse ponto único de falha, e um conjunto de software para gerenciar o *cluster* e distribuir as solicitações dos clientes para o outro nó, no caso de uma falha. Também não foi usado nenhum *hardware* proprietário, foram usados computadores padrão PC (*Personal Computer*) ficando a manutenção do sistema independente do fornecedor do produto para aquisição peças e serviços.

manutenção planejada

: O sistema permite paralisação para manutenção cada um de seus componentes, desde que não seja simultaneamente. Caso haja a necessidade de manutenção em ambos os nós, ela pode ser feita de maneira alternada.

tolerância a falhas do sistema

Nos testes, foram simuladas falhas de *hardware*, que fizeram com que o outro nó assumisse os serviços e as solicitações dos clientes fosse atendida.

tolerância a falhas de conectividade

Também foram simulados testes de conectividade, que ocasionaram o *failover* e o *failback* com sucesso.

A solução mostrou-se bastante adequada a realidade das pequenas e médias empresas que utilizam servidores Samba como controlador de domínio e servidor de arquivos.

Referências Bibliográficas

BLACKMON, S.; NGUYEN, J. High-availability file server with heartbeat. *Sys Admin Magazine*, v. 10, n. 9, September 2001.

ELLENBERG, L. *Data Redundancy By Drbd*. [on-line]. Disponível na internet via http://wiki.linux-ha.org/DataRedundancyByDrbd. Arquivo capturado em 2 de abril de 2005.

ELLENBERG, L. *Quick Start Guide For DRBD 0.7*. [on-line]. Disponível na internet via http://wiki.linux-ha.org/DRBD_2fQuickStart07. Arquivo capturado em 2 de abril de 2005.

MILZ, H. *Linux High Availability HOWTO*. 1998. [on-line]. Disponível na internet via http://ha.underlinux.com.br/doc/howto/High-Availability-HOWTO.html. Arquivo capturado em 2 de abril de 2005.

ROBERTSON, A. *Heartbeat Home Page*. [on-line]. Disponível na internet via http://www.linux-ha.org/heartbeat/. Arquivo capturado em 2 de abril de 2005.

TS, J.; ECKSTEIN, R.; COLLIER-BROWN, D. *Using Samba*. 2. ed. [S.l.]: O'Reilly, 2003.

VERNOOIJ, J.; TERPSTRA, J.; CARTER, G. *The Official Samba-3 HOWTO And Reference Guide*. 2003. [on-line] Disponível na internet em http://us1.samba.org/samba/docs/man/Samba-HOWTO-Collection/. Arquivo capturado em 2 de abril de 2005.