

RAMON BASSAN

0402110

**AVALIAÇÃO DE CLUSTER DE ALTA DISPONIBILIDADE
BASEADO EM SOFTWARE LIVRE**

Jaguariúna

2008

RAMON BASSAN

0402110

**AVALIAÇÃO DE CLUSTER DE ALTA DISPONIBILIDADE
BASEADO EM SOFTWARE LIVRE**

Monografia apresentada à disciplina Trabalho de Conclusão de Curso, do Curso de Ciência da Computação da Faculdade de Jaguariúna, sob a orientação do Prof. Carlos Alessandro Bassi Viviani, como exigência parcial para conclusão do curso de graduação.

Jaguariúna

2008

BASSAN, Ramon. **Avaliação de Cluster de Alta Disponibilidade Baseado em Software Livre**. Monografia defendida e aprovada na FAJ em <data> pela banca examinadora constituída pelos professores:

Prof. Carlos Alessandro Bassi Viviani
FAJ – orientador

Prof. Dr. André Mendeleck

Prof. Maurício Tadeu Teixeira

À meus pais, amigos e professores

Pela confiança que depositaram em mim e também pelo entusiasmo com que me motivaram a concluir este trabalho.

AGRADECIMENTOS

Agradeço a minha família e aos meus amigos pelo apoio e compreensão e principalmente a Deus.

Ao meu orientador pelo apoio, motivação e orientação.

À rede de supermercados BON-NETTO pela oportunidade dada no desenvolvimento e implantação da rede de Alta Disponibilidade

Software é como sexo: é melhor quando é de graça.

(Linus Torvalds)

BASSAN, Ramon. **Avaliação de Cluster de Alta Disponibilidade Baseado em Software Livre**. 2008. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação da Faculdade de Jaguariúna, Jaguariúna.

RESUMO

Este trabalho tem por objetivo descrever a construção de um cluster de alta disponibilidade usufruindo apenas de componentes de software livre. A solução apresentada reside na implementação da arquitetura Heartbeat e DRBD num cluster com dois nodos, testando a sua viabilidade e adaptabilidade a serviços comuns, como um servidor de arquivos. Os cenários implementados e os testes efetuados pretendem demonstrar o potencial dessa arquitetura em ambientes Linux, tirando partido do baixo custo associado, da facilidade de configuração e da atualização permanente do software. A solução de alta disponibilidade, presente neste trabalho, é baseada em três blocos básicos, que são: replicação de disco, monitoração dos nodos e sistema de arquivos robusto. Estes três blocos podem ser utilizados em conjunto ou individualmente, possibilitando a criação de soluções com failover (falha) e failback (recuperação de falhas), automáticos ou manuais, e mesmo suportando paradas planejadas. Esta solução foi idealizada para um cluster de dois nodos.

Palavras-chave: CLUSTER, DISPONIBILIDADE, LIVRE

SUMÁRIO

LISTA DE SIGLAS	8
LISTA DE FIGURAS	9
LISTA DE TABELAS.....	10
1. INTRODUÇÃO.....	11
2. ESTUDO TEÓRICO SOBRE CLUSTER.....	12
2.1 CARACTERÍSTICAS BÁSICAS DE UM CLUSTER	13
2.2 COMPOSIÇÃO DE UM CLUSTER	13
2.3 UMA POSSÍVEL ARQUITETURA DE CLUSTER	14
2.4 ALTA DISPONIBILIDADE	15
2.4.1 <i>Disponibilidade básica</i>	16
2.4.2 <i>Alta disponibilidade</i>	16
2.4.3 <i>Disponibilidade contínua</i>	16
2.5 CÁLCULO DA DISPONIBILIDADE	17
2.6 CONCEITOS	19
2.6.1 <i>Falha</i>	19
2.6.2 <i>Erro</i>	19
2.6.3 <i>Defeito</i>	19
2.6.4 <i>Failover</i>	20
2.6.5 <i>Failback</i>	20
2.6.6 <i>Missão</i>	20
2.7 O HEARTBEAT	21
2.8 O DRBD	23
2.9 SISTEMA DE ARQUIVOS.....	24
3. CASE DE UM PROJETO DE CLUSTER DE ALTA DISPONIBILIDADE EM UM SUPERMERCADO	25
3.1 CRONOGRAMA DO PROJETO	25
3.2 ETAPAS DO PROJETO	25
4. CONCLUSÃO.....	35
5. REFERÊNCIAS BIBLIOGRÁFICAS.....	36
6. ANEXOS.....	37

Lista de Siglas

CPU	- Unidade Central de Processamento
DRBD	- Distributed Replicated Block Device
EXT3	- Third Extended File System
IP	- Internet Protocol
MTTF	- Mean Time to Failure
MTTR	- Mean Time to Recovery
PDV	- Ponto De Venda

Lista de Figuras

Figura 1: Cluster de 4 nodos (FILHO, 2004)	12
Figura 2: Arquitetura de um cluster (ZEM...,2008)	15
Figura 3: Cluster de Alta Disponibilidade (FERREIRA, 2008)	21
Figura 4: Esquema básico do funcionamento do Heartbeat (HEARTBEAT, 2008).....	23
Figura 5: Esquema de funcionamento do DRBD (DRBD, 2008)	23
Figura 6: Exemplo do uso do DRBD (ZENOSS, 2008).....	24
Figura 7: Topologia da rede de alta disponibilidade	26
Figura 8: Instalação do pacote Heartbeat no Slackware	27
Figura 9: Instalação do pacote DRBD no Slackware.....	28
Figura 10: DRBD sincronizado entre os nodos	29
Figura 11: Monitoração de serviços mostra Heartbeat ativo.....	30
Figura 12: Interface virtual criada pelo Heartbeat.....	30
Figura 13: A interface virtual não existe nodo secundário	31
Figura 14: Nodo primário em atividade	31
Figura 15: Conteúdo do nodo primário.....	32
Figura 16: Nodo secundário em atividade.....	33
Figura 17: Conteúdo no nodo secundário	33
Figura 18: Estações na rede	34

Lista de Tabelas

Tabela 1: Níveis de alta disponibilidade (WIKIPEDIA, 2008).....	18
Tabela 2: Protocolos de replicação de dados do DRBD (LEVITA, 2005)	24
Tabela 3: Cronograma do projeto	25

1. INTRODUÇÃO

Atualmente as empresas precisam prover serviços ininterruptos. Torna-se vital para qualquer empresa que deseja manter-se competitiva no mercado possuir uma estratégia envolvendo alta disponibilidade para os seus servidores de aplicação e, estar preparada para uma falha de hardware inesperada.

Há um tempo atrás, quando falava-se de um ambiente de alta disponibilidade já se pensava nos custos envolvidos para a construção desse ambiente. Podemos afirmar que os valores já foram extremamente altos, mas agora acompanham a tendência dos outros ambientes de tecnologia.

Com a utilização do Linux como sistema operacional para controle do servidor, a implantação do cluster de alta disponibilidade tornou-se mais acessível.

Sistemas de alta disponibilidade provêm um aumento da disponibilidade de serviços através de técnicas computacionais, tanto através de recursos de software quanto de hardware.

O Cluster de alta disponibilidade entrega contínua disponibilidade dos serviços eliminando pontos de falha e garantindo que o serviço seja transferido para outro nodo do cluster ou em até um grupo de nodos (Domínio de Failover) em caso de detecção de falha. Genericamente serviços de alta disponibilidade efetuam leitura e gravação (através de um sistema de arquivos montado centralizado) como em aplicações de banco de dados por exemplo.

As configurações realizadas conseguem garantir a integridade das operações dos serviços entre os nodos do cluster, seja ativo-ativo ou em ativo-passivo.

Em resumo a falha nos serviços em um cluster de alta disponibilidade não é detectada pelos clientes.

Várias empresas desenvolvem soluções integradas de alta disponibilidade proprietárias. Além do custo elevado, estas soluções assentam em arquiteturas proprietárias de um fabricante. Por outro lado, sistemas de código abertos têm conseguido uma aceitação crescente nos últimos anos em várias áreas. Também na implantação de arquiteturas de alta disponibilidade é possível encontrar soluções de atualização quase permanentes e a custos reduzidos.

Garantir o máximo de disponibilidade dos serviços sem que houvesse um custo muito elevado, utilizando-se de robustez e confiabilidade no sistema, da configuração e manutenção não muito complexa, foram fatores que pesaram na hora da escolha do sistema operacional Linux e as ferramentas Heartbeat e DRBD para a criação do cluster.

2. ESTUDO TEÓRICO SOBRE CLUSTER

Para que se possa compreender o conceito de alta disponibilidade é necessário entender como funcionam os clusters.

O cluster é um conjunto de máquinas independentes chamadas de nodos (ou nós), que cooperam entre si para atingir uma determinada tarefa. Para garantir que as tarefas sejam executadas, esse conjunto de máquinas deve comunicar-se umas com as outras a fim de coordenar e organizar todas as ações a serem executadas, além de compartilhar algum hardware a fim de poder tolerar certas situações de falhas.

Para o usuário externo, o cluster é visto como um único sistema lógico.

A Figura 1 exibe um cluster de 4 nodos, onde são compartilhados dispositivos de armazenamento e dispositivos de comunicação em rede.

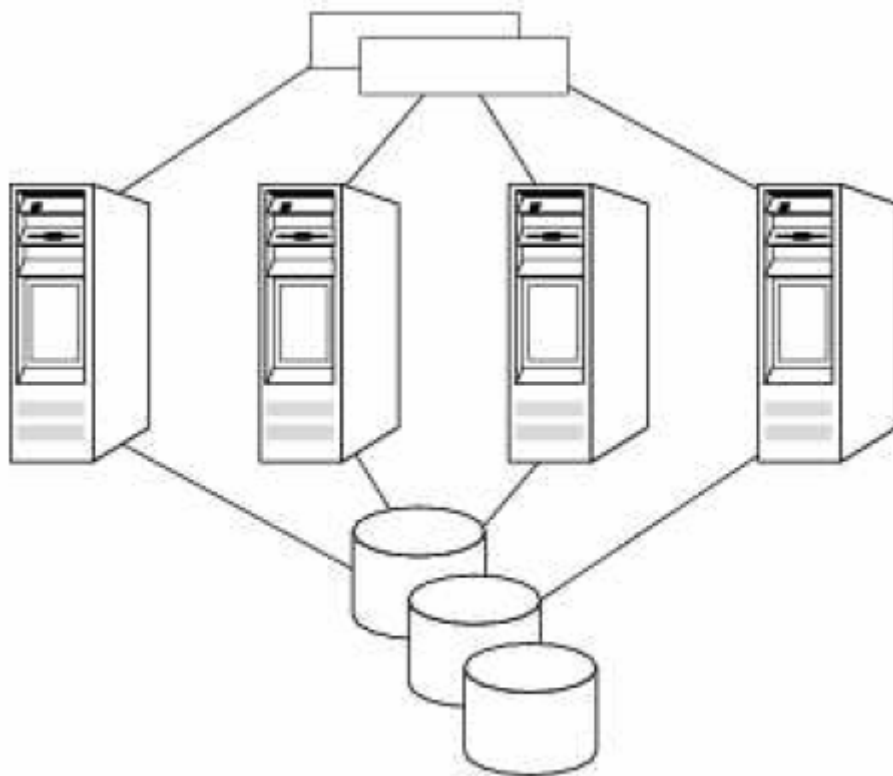


Figura 1: Cluster de 4 nodos (FILHO, 2004)

Há dois tipos de formação de clusters:

- Alta Disponibilidade (HA – High Availability): capacidade de um sistema manter-se no ar após a ocorrência de falhas. Dependente do cluster.
- Alta Performance (HPC – High Performance Computing): um tipo de sistema para processamento paralelo ou distribuído que consiste em uma coleção de computadores interconectados.

2.1 Características básicas de um cluster

- **Comodidade:** Os nodos de um cluster devem ser máquinas normais interconectadas por uma rede genérica. O sistema operacional também deve ser padrão, sendo que o software de gerenciamento deve estar acima dele como uma aplicação qualquer (FILHO, 2004).
- **Escalabilidade:** Deve ser possível adicionar aplicações, nodos, periféricos e interconexões de rede sem interromper a disponibilidade dos serviços do cluster (FILHO, 2004).
- **Transparência:** Apesar de ser constituído por um grupo de nodos independentes fracamente agrupados, um cluster aparece como um único sistema a clientes externos. Aplicações clientes interagem com o cluster como se ele fosse um único servidor com alta performance e/ou disponibilidade (FILHO, 2004).
- **Confiabilidade:** O cluster deve ter capacidade de detectar falhas internas ao grupo, assim como de tomar atitudes para que as falhas não comprometam os serviços oferecidos (FILHO, 2004).
- **Gerenciamento e manutenção:** Uma das principais dificuldades em se trabalhar com clusters é o seu gerenciamento. A configuração e manutenção de clusters são muitas vezes tarefas complexas e propensas a erros. Um fácil mecanismo de gerenciamento do ambiente deve existir a fim de que o cluster não seja um grande sistema complexo com um árduo trabalho de administração (FILHO, 2004).

2.2 Composição de um cluster

O nodo é a unidade básica do cluster; um conjunto de nodos forma um cluster. Em um cluster, um nodo comunica-se com os outros através de mensagens sobre conexões de rede, e falhas de nodos podem ser detectadas através da ausência destas mensagens no canal de comunicação (FILHO, 2004).

Um nodo é um sistema computacional unicamente identificado, conectado a um ou mais computadores em uma rede. Assim, um nodo tem quatro componentes principais:

- **CPU:** O componente de processamento principal de um computador.
- **Memória:** É o componente usado para executar a bufferização das informações. Memória volátil.
- **Repositório de Armazenamento:** Um dispositivo que armazena informações. Geralmente um repositório persistente que deve ser acessado por transações de leitura/escrita para alterar seu conteúdo.

- Interconexão: Este é o canal de comunicação entre os nodos.

2.3 Uma possível arquitetura de cluster

Um exemplo de como os computadores podem ser interligados para constituir um cluster de computadores pode ser visto na Figura 2. A representação apresentada na referida figura demonstra uma configuração ideal, replicando-se os componentes responsáveis pelo balanceamento de carga e do provimento do sistema de arquivos, além de vários subsistemas de comunicação visando uma melhoria na comunicação entre os componentes, porém, este arranjo nem sempre é materializado; na verdade não existe a obrigatoriedade de que um cluster de computadores tenha ou necessite de toda essa infraestrutura para funcionar (ZEM...,2008).

Na Figura 2 podem ser observados os seguintes componentes, aqui chamados de nodos:

- NU0-NU7: são os nodos dos usuários, ou seja, são os equipamentos que os usuários utilizam para interagir com o cluster de computadores (ZEM...,2008).
- BC0-BC1: são os nodos balanceadores de carga e têm como funções o recebimento das requisições enviadas pelos usuários e a incumbência de escolher para qual dos NCs a mesma deverá ser encaminhada. No caso, o nodo BC0 é o nodo balanceador de carga principal e o BC1 é sua réplica (ZEM...,2008).
- NC0-NC7: são os nodos de computação e sua função é a de realizar o processamento daquilo que foi solicitado pelos usuários e atribuído pelo balanceador de carga (ZEM...,2008).
- NA0-NA1: são os nodos de armazenamento e sua função é a de prover um sistema de arquivos disponível, estável e confiável. De maneira análoga ao que acontece com o nodo balanceador de carga, o NA0 é o nodo de armazenamento principal e o NA1 é sua réplica (ZEM...,2008).

E os seguintes subsistemas de comunicação:

- Sub-Sistema de Comunicação A: é utilizado pelos nodos dos usuários para enviar as requisições para o cluster de computadores e eventualmente receber o resultado do processamento realizado (ZEM...,2008).
- Sub-Sistema de Comunicação B: é utilizado pelo nodo balanceador de carga para enviar as requisições dos usuários para os nodos de computação escolhidos para processamento e vice-versa, podem também ser usado para uma eventual comunicação entre os próprios nodos de computação (ZEM...,2008).
- Sub-Sistema de Comunicação C: é utilizado para que os nodos de computação possam realizar o acesso ao sistema de arquivos do cluster, armazenando ou

recuperando informações do sistema de arquivos provido pelos nodos de armazenamento (ZEM...,2008).

- Sub-Sistema de Comunicação D: é utilizado pelos nodos balanceadores de carga para a realização de um auto monitoramento sobre seu funcionamento. Este auto monitoramento pode ser realizado através da técnica conhecida por Heartbeat (ZEM...,2008).
- Sub-Sistema de Comunicação E: é utilizado pelos nodos de armazenamento para realização de um auto monitoramento (similar ao realizado pelos nodos balanceadores de carga) e também para a sincronização dos sistemas de arquivos. Esta sincronização pode ser feita utilizando-se o DRBD (ZEM...,2008).

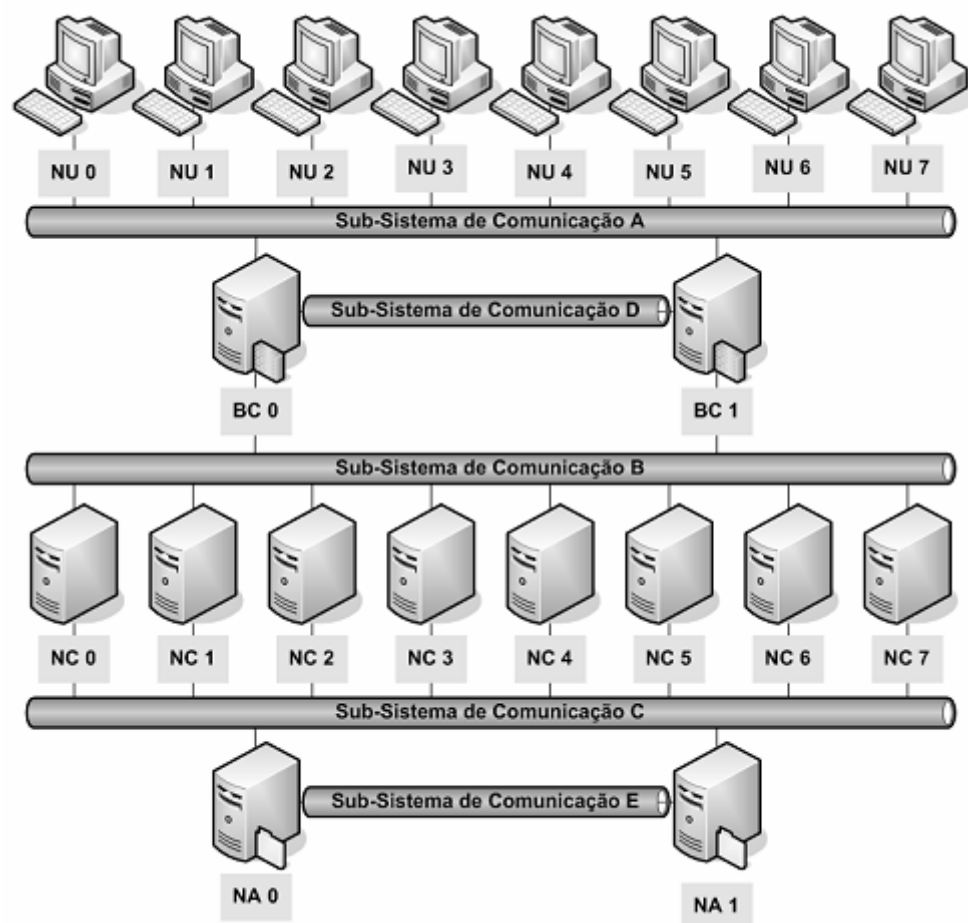


Figura 2: Arquitetura de um cluster (ZEM...,2008)

2.4 Alta disponibilidade

Para que se entenda a alta disponibilidade faz-se necessário, antes de tudo, perceber que a alta disponibilidade não é apenas um produto ou uma aplicação que se instale, e sim uma característica de um sistema computacional. Existem mecanismos e técnicas, blocos básicos, que podem ser utilizados para aumentar a disponibilidade de um sistema. A simples utilização destes blocos, entretanto, não garante este aumento se não for

acompanhado de um completo estudo e projeto de configuração.

A disponibilidade de um sistema computacional, indicada por $A(t)$, é a probabilidade de que este sistema esteja funcionando e pronto para uso em um dado instante de tempo t . Esta disponibilidade pode ser enquadrada em três classes, de acordo com a faixa de valores desta probabilidade. As três classes são: Disponibilidade Básica, Alta Disponibilidade e Disponibilidade Contínua (SZTOLTZ, 2003).

2.4.1 Disponibilidade básica

A Disponibilidade básica é aquela encontrada em máquinas comuns, sem nenhum mecanismo especial, em software ou hardware, que vise de alguma forma mascarar as eventuais falhas destas máquinas. Costuma-se dizer que máquinas nesta classe apresentam uma disponibilidade de 99% a 99,9%. Isto equivale a dizer que em um ano de operação a máquina pode ficar indisponível por um período de 9 horas a quatro dias. Estes dados são empíricos e os tempos não levam em consideração a possibilidade de paradas planejadas (que serão abordadas mais adiante), porém são aceitas como o senso comum na literatura da área (SZTOLTZ, 2003).

2.4.2 Alta disponibilidade

Adicionando-se mecanismos especializados de detecção, recuperação e mascaramento de falhas, pode-se aumentar a disponibilidade do sistema, de forma que este venha a se enquadrar na classe de alta disponibilidade. Nesta classe as máquinas tipicamente apresentam disponibilidade na faixa de 99,99% a 99,999%, podendo ficar indisponíveis por um período de pouco mais de 5 minutos até uma hora em um ano de operação. Aqui se encaixam grande parte das aplicações comerciais de alta disponibilidade, como centrais telefônicas (SZTOLTZ, 2003).

2.4.3 Disponibilidade contínua

Com a adição de novas se obtém uma disponibilidade cada vez mais próxima de 100%, diminuindo o tempo de inoperância do sistema de forma que este venha a ser desprezível ou mesmo inexistente. Isso é a disponibilidade contínua, o que significa que todas as paradas planejadas e não planejadas são mascaradas, e o sistema está sempre disponível.

Como já pode ser percebido de sua definição, o principal objetivo da alta disponibilidade é buscar uma forma de manter os serviços prestados por um sistema a outros elementos, mesmo que o sistema em si venha a se modificar internamente por causa de uma falha. Esse é o conceito de mascaramento de falhas, através de redundância ou replicação. Um determinado serviço, que se quer altamente disponível, é colocado por trás

de uma camada de abstração, que permita mudanças em seus mecanismos internos mantendo intacta a interação com elementos externos.

Este é o coração da alta disponibilidade, uma sub-área da tolerância a falhas, que visa manter a disponibilidade dos serviços prestados por um sistema computacional, através da redundância de hardware e reconfiguração de software. Vários computadores juntos agindo como um só, cada um monitorando os outros e assumindo seus serviços caso perceba que algum deles falhou.

Outra possibilidade importante da alta disponibilidade é fazer isto com computadores básicos, como os que se pode comprar até num supermercado. A complexidade pode estar apenas no software. Mais fácil de desenvolver que o hardware, o software de alta disponibilidade é quem se preocupa em monitorar outras máquinas de uma rede, saber que serviços estão sendo prestado, quem os está prestando, e o que fazer quando uma falha é percebida (SZTOLTZ, 2003).

2.5 Cálculo da disponibilidade

Se um componente falha, ele é reparado ou substituído por um novo componente. Se este novo componente falha, é substituído por outro e assim por diante. O componente reparado é tido como no mesmo estado que um componente novo. Durante sua vida útil, um componente pode ser considerado como estando em um destes estados: funcionando ou em reparo. O estado funcionando indica que o componente está operacional e o estado em reparo significa que ele falhou e ainda não foi substituído por um novo componente.

Em caso de defeitos, o sistema vai de funcionando para em reparo, e quando a substituição é feita ele volta para o estado funcionando. Sendo assim, pode-se dizer que o sistema apresenta ao longo de sua vida um tempo médio até apresentar falha (MTTF) e um tempo médio de reparo (MTTR). Seu tempo de vida é uma sucessão de MTTFs e MTTRs, à medida que vai falhando e sendo reparado. O tempo de vida útil do sistema é a soma dos MTTFs nos ciclos MTTF+MTTR já vividos.

Simplificando, diz-se que a disponibilidade de um sistema é a relação entre o tempo de vida útil deste sistema e seu tempo total de vida. Isto pode ser representado pela fórmula abaixo:

$$\text{Disponibilidade} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$$

Exemplo:

- A rede não deve falhar mais do que 1 vez em 166 dias (4000 horas) e o tempo de reparo não deve exceder 1 hora
- Disponibilidade = $4.000/4.001 = 99,98\%$

Ao avaliar uma solução de alta disponibilidade, é importante levar em consideração se na medição do MTTF são observadas como falhas as possíveis paradas planejadas (SZTOLTZ, 2003).

Disponibilidade pode ser expressa como um percentual por ano, mês, semana, dia ou hora que a rede está disponível em relação ao tempo total.

Por exemplo:

- Operação 24/7 = 24 horas, 7 dias da semana = 100% da disponibilidade
- 165 horas em 168 horas semanais = 98,21% de disponibilidade

Aplicações podem requerer diferentes tipos de disponibilidade.

A Tabela 1 ilustra um dos termos de comparação geralmente utilizado na avaliação de soluções de alta disponibilidade. Níveis de disponibilidade segundo tempos de indisponibilidade (downtime). Excluídos desta tabela, os tempos de downtime estimados (geralmente para manutenção ou reconfiguração dos sistemas) são alheios às soluções e muito variáveis (WIKIPEDIA, 2008).

Disponibilidade (%)	Downtime/ano	Downtime/mês
95%	18 dias 6:00:00	1 dia 12:00:00
96%	14 dias 14:24:00	1 dia 4:48:00
97%	10 dias 22:48:00	0 dias 21:36:00
98%	7 dias 7:12:00	0 dias 14:24:00
99%	3 dias 15:36:00	0 dias 7:12:00
99,9%	0 dias 8:45:35.99	0 dias 0:43:11.99
99,99%	0 dias 0:52:33.60	0 dias 0:04:19.20
99,999%	0 dias 0:05:15.36	0 dias 0:00:25.92

Tabela 1: Níveis de alta disponibilidade (WIKIPEDIA, 2008)

Quanto maior a disponibilidade, maior a redundância e custo das soluções. Tudo depende do tipo de serviço que se pretende disponibilizar. Por exemplo, um operador de telecomunicações quererá certamente o mais elevado a fim de poder garantir um elevado nível de disponibilidade, sob pena de perder os seus clientes caso o sistema sofra falhas constantemente. No entanto, uma empresa com horário de trabalho normal poderá considerar que 90% de disponibilidade serão suficientes. É de salientar que o nível de disponibilidade mensal não é o mesmo que o anual. Efetivamente, para se obter um nível de disponibilidade mensal de 97%, é necessário que o nível anual seja aproximadamente de 99,75% (WIKIPEDIA, 2008).

2.6 Conceitos

Para se entender corretamente do que se está falando quando se discute uma solução de alta disponibilidade, deve-se conhecer os conceitos envolvidos. Não são muitos, porém estes termos são muitas vezes utilizados de forma errônea em literatura não especializada. Antes de qualquer coisa, deve-se entender o que é falha, erro e defeito. Estas palavras, que parecem tão próximas, na verdade designam a ocorrência de algo anormal em três universos diferentes de um sistema computacional (SZTOLTZ, 2003).

2.6.1 Falha

Uma falha acontece no universo físico, ou seja, no nível mais baixo do hardware. Uma flutuação da fonte de alimentação, por exemplo, é uma falha. Uma interferência eletromagnética também. Estes são dois eventos indesejados, que acontecem no universo físico e afetam o funcionamento de um computador ou de partes dele (SZTOLTZ, 2003).

2.6.2 Erro

A ocorrência de uma falha pode acarretar um erro, que é a representação da falha no universo informacional. Um computador trabalha com bits, cada um podendo conter 0 ou 1. Uma falha pode fazer com que um (ou mais de um) bit troque de valor inesperadamente, o que certamente afetará o funcionamento normal do computador. Uma falha, portanto, pode gerar um erro em alguma informação (SZTOLTZ, 2003).

2.6.3 Defeito

Se o erro não for percebido e tratado, poderá gerar o que se conhece por defeito. O sistema simplesmente trava, mostra uma mensagem de erro, ou ainda perde os dados do usuário sem maiores avisos. Isto é percebido no universo do usuário.

Generalizando, uma falha no universo físico pode causar um erro no universo informacional, que por sua vez pode causar um defeito percebido no universo do usuário. A tolerância a falhas visa exatamente acabar com as falhas, ou tratá-las enquanto ainda são erros. Já a alta disponibilidade permite que máquinas travem ou errem, contanto que exista outra máquina para assumir seu lugar.

Para que uma máquina assuma o lugar de outra, é necessário que descubra de alguma forma que a outra falhou. Isso é feito através de testes periódicos, cujo período deve ser configurável, nos quais a máquina secundária testa não apenas se a outra está ativa, mas também fornecendo respostas adequadas a requisições de serviço. Um mecanismo de detecção equivocado pode causar instabilidade no sistema. Por serem periódicos, nota-se

que existe um intervalo de tempo durante o qual o sistema pode estar indisponível sem que a outra máquina o perceba (SZTOLTZ, 2003).

2.6.4 Failover

O processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha, é chamado failover. O failover pode ser automático ou manual, sendo o automático o que normalmente se espera de uma solução de alta disponibilidade, porém não é o recomendado. Ainda assim, algumas aplicações não críticas podem suportar um tempo maior até a recuperação do serviço, e, portanto podem utilizar failover manual. Além do tempo entre a falha e a sua detecção, existe também o tempo entre a detecção e o restabelecimento do serviço. Grandes bancos de dados, por exemplo, podem exigir um considerável período de tempo até que atualizem os índices de suas tabelas e, durante este tempo, o serviço ainda estará indisponível.

Para se executar o failover de um serviço, é necessário que as duas máquinas envolvidas possuam recursos equivalentes. Um recurso pode ser uma placa de rede, um disco rígido, os dados neste disco, e todo e qualquer elemento necessário à prestação de um determinado serviço. É vital que uma solução de alta disponibilidade mantenha recursos redundantes com o mesmo estado, de forma que o serviço possa ser retomado sem perdas.

Dependendo da natureza do serviço, executar um failover significa interromper as transações em andamento, perdendo-as, sendo necessário reiniciá-las após o failover. Em outros casos, significa apenas um retardo até que o serviço esteja novamente disponível. Nota-se que o failover pode ou não ser um processo transparente, dependendo da aplicação envolvida (SZTOLTZ, 2003).

2.6.5 Failback

Ao ser percebida a falha de um servidor, além do failover é obviamente necessário que se faça manutenção no servidor falho. Ao ser recuperado de uma falha, este servidor será recolocado em serviço, e então se tem a opção de realizar o processo inverso do failover, que se chama failback, ou seja, é o processo de retorno de um determinado serviço de outra máquina para sua máquina de origem. Também pode ser automático, manual ou até mesmo indesejado. Em alguns casos, em função da possível nova interrupção na prestação de serviços, o failback pode não ser atraente (SZTOLTZ, 2003).

2.6.6 Missão

Missão de um sistema é o período de tempo no qual ele deve desempenhar suas funções sem interrupção. Por exemplo, uma farmácia, que funcione das 8h às 20h, não

pode ter seu sistema fora do ar durante este período de tempo. Se este sistema vier a apresentar defeitos fora deste período, ainda que indesejados, estes defeitos não atrapalham em nada o andamento correto do sistema quando ele é necessário. Uma farmácia 24h obviamente tem uma missão contínua, de forma que qualquer tipo de parada deve ser mascarada.

A alta disponibilidade visa eliminar as paradas não planejadas. Porém, no caso da primeira farmácia, as paradas planejadas não devem acontecer dentro do período de missão. Paradas não planejadas decorrem de defeitos, já paradas planejadas são aquelas que se devem a atualizações, manutenção preventiva e atividades correlatas. Desta forma, toda parada dentro do período de missão pode ser considerada uma falha no cálculo da disponibilidade.

Uma aplicação de Alta Disponibilidade pode ser projetada inclusive para suportar paradas planejadas, o que pode ser importante, por exemplo, para permitir a atualização de programas por problemas de segurança, sem que o serviço deixe de ser prestado (SZTOLTZ, 2003).

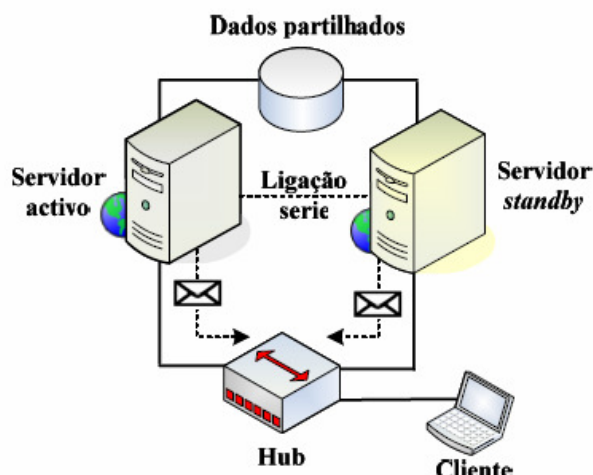


Figura 3: Cluster de Alta Disponibilidade (FERREIRA, 2008)

2.7 O Heartbeat

O programa Heartbeat é usado para construir cluster da altíssima disponibilidade. Heartbeat significa batimento cardíaco. Este termo é usado para definir o pulso enviado entre duas máquinas, indicando que estão “vivas”, ou seja, estão funcionando e disponíveis para executarem tarefas. Se o Heartbeat falhar, a máquina secundária assumirá que a primária falhou e tomará para si os serviços que estavam sendo executados na máquina primária (TRIGO, 2007).

A comunicação entre os servidores pode ser feita em broadcast, multicast ou unicast. Esta escolha depende da aplicação que será dada ao Heartbeat. No entanto, se o servidor

primário estiver ligado apenas ao servidor de backup, é aconselhável utilizar o unicast, por enviar um pacote único e, assim, evitar congestionamento da rede (TRIGO, 2007).

A configuração do Heartbeat consiste em três arquivos:

- `ha.cf`: configuração dos computadores envolvidos e o meio de comunicação utilizado entre os nodos.
- `haresources`: configura o endereço IP do cluster e o grupo de serviços que serão executados preferencialmente em cada um dos nodos.
- `authkeys`: armazena a autenticação necessária entre os dois nodos.

O Heartbeat deverá ser instalado em ambos os servidores e todos os arquivos devem estar iguais nos dois nodos.

É possível configurar o tempo entre os Heartbeats, o tempo de alerta no caso do Heartbeat não chegar a tempo e o tempo em que se considera o nodo como morto. O Heartbeat utiliza a porta 694 para a comunicação bcast ou ucast. Existe uma opção chamada `auto_failback` que nos permite escolher entre on e off, isto é, quando pusermos o `auto_failback` a on, quando o nodo primário voltar de uma falha qualquer, ele volta a tomar controle do cluster, enquanto que em off o nodo primário é prevenido para readquirir o controle do cluster (HEARTBEAT, 2008).

O Heartbeat funciona através de um IP virtual que identifica o cluster, este IP está associado a um domínio. Quando o nodo primário falhar, o nodo secundário assume o IP virtual, ou seja, o IP do cluster. Isso vai permitir continuar com o funcionamento dos serviços que estavam a ser usados pelo nodo primário com o mesmo domínio (HEARTBEAT, 2008).

O funcionamento do Heartbeat é apresentado na Figura 4. Uma vez a conexão estabelecida, começa o processo dos Heartbeats. Quando há um Heartbeat perdido, vem juntamente com ele uma tentativa de reconexão, dessa tentativa duas hipóteses é tomada em conta. A primeira hipótese é o caso da tentativa permitida e então pode acontecer que a conexão seja restabelecida ou perdida. A segunda hipótese é o caso da tentativa inválida e daí, só há uma saída que é a conexão perdida. A chamada conexão terminada acontece quando se decide desconectar ou quando a conexão é perdida (HEARTBEAT, 2008).

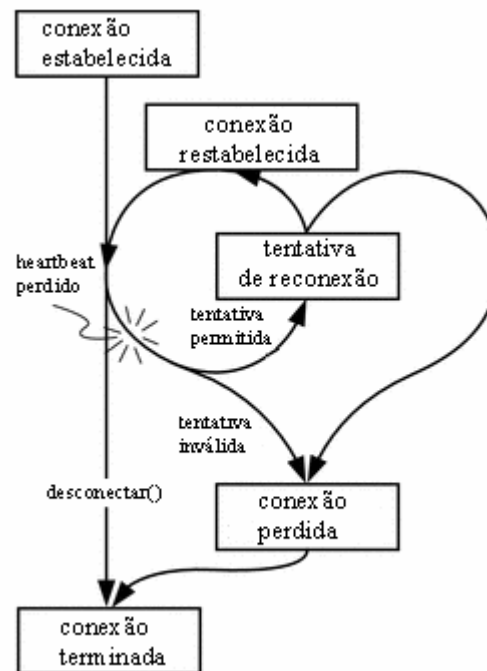


Figura 4: Esquema básico do funcionamento do Heartbeat (HEARTBEAT, 2008)

2.8 O DRBD

A replicação dos discos é feita pelo DRBD, que é um driver de bloco para o kernel que cria um dispositivo de bloco virtual, consistindo tanto de um disco real local quanto de uma conexão de rede, que terá na outra ponta outro driver DRBD atuando como secundário. Tudo aquilo que é escrito no dispositivo virtual é escrito no disco local e também enviado para o outro driver, que fará a mesma operação em seu disco local. Com isto se obtém dois nodos com discos exatamente iguais, até o instante da falha. As aplicações que trabalham com dados dinâmicos ou atualizados com muita frequência se beneficiam deste driver (SZTOLTZ, 2003).

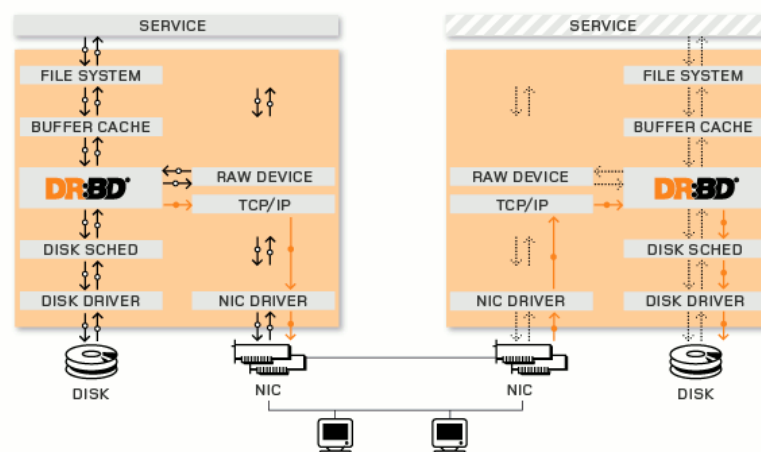


Figura 5: Esquema de funcionamento do DRBD (DRBD, 2008)

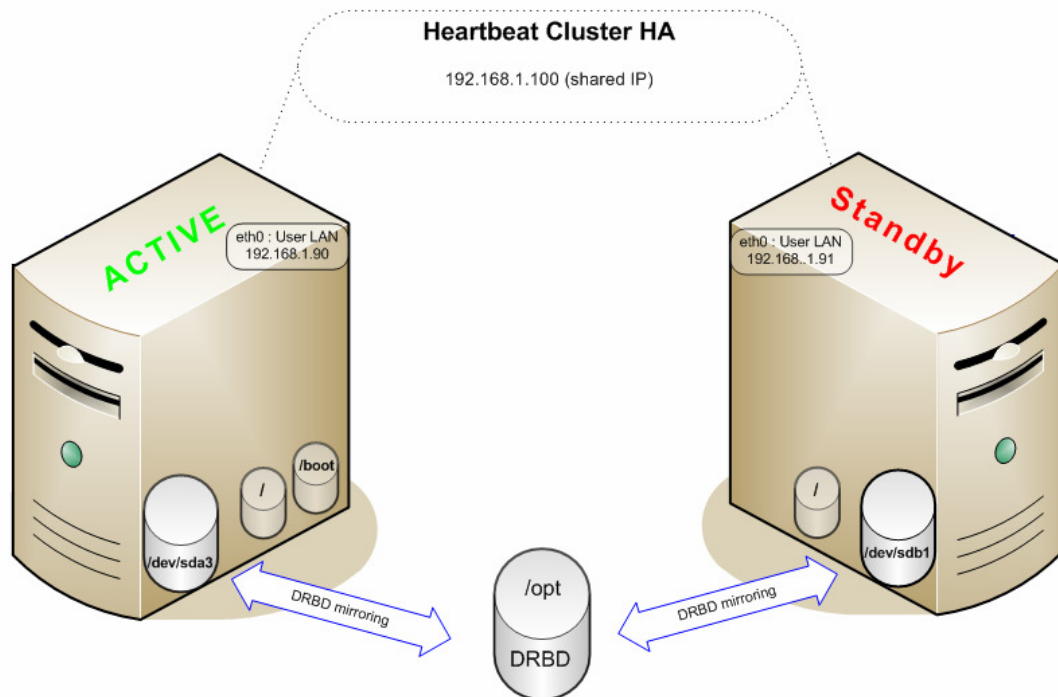


Figura 6: Exemplo do uso do DRBD (ZENOSS, 2008)

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

Tabela 2: Protocolos de replicação de dados do DRBD (LEVITA, 2005)

2.9 Sistema de arquivos

Dados replicados ou não, é importante que o sistema de arquivos esteja consistente. Nem todos os sistemas de arquivos garantem isso, portanto para esta solução se escolheu trabalhar com o sistema de arquivos ext3¹. Este sistema de arquivos trabalha com journal, o que significa que todas as alterações de dados são antes registradas no disco para que, caso o sistema venha a falhar durante este processo, a transação possa ser recuperada quando o sistema voltar. Isto confere agilidade ao processo de recuperação de falhas, bem como aumenta muito a confiabilidade das informações armazenadas (SZTOLTZ, 2003).

¹ Sistema de arquivos mais utilizado no mundo Linux. Usado por padrão na maioria das distribuições Linux.

3. CASE DE UM PROJETO DE CLUSTER DE ALTA DISPONIBILIDADE EM UM SUPERMERCADO

3.1 Cronograma do projeto

Segue abaixo o cronograma de implantação da solução com suas etapas e o tempo gasto em cada uma delas.

	Ago.	Set.	Out.	Nov.
1	X	X		
2		X	X	
3			X	
4			X	X
5				X
6				X

Tabela 3: Cronograma do projeto

Etapa 1 – Aquisição dos equipamentos

Etapa 2 – Definição da topologia a ser utilizada

Etapa 3 – Definição do sistema operacional e dos softwares

Etapa 4 – Instalação e configuração do sistema

Etapa 5 – Testes iniciais

Etapa 6 – Implantação definitiva

3.2 Etapas do projeto

Etapa 1 – Aquisição dos equipamentos

A aquisição de equipamentos mais robustos para exercer a função de servidores foi tomada após várias reuniões com o setor de TI, ao qual decidiu adquirir os seguintes equipamentos:

Servidor Principal

- Dell Poweredge 1900

Servidor Escravo

- HP Proliant ML 110

Ambos os servidores possuem 2 interfaces ethernet, sendo que as interfaces escolhidas para conexão Heartbeat foram as interfaces de velocidade Gigabit², para manter a sincronização a uma velocidade boa.

Etapa 2 – Definição da topologia a ser utilizada

A função dos servidores é apenas prover autenticação dos usuários e disponibilizar o sistema principal através de um servidor de arquivos.

A Figura 6 mostra a topologia utilizada, a base replicada (/home) e a separação da rede Heartbeat (10.0.0.0/8) da rede do supermercado (192.168.0.0/24).

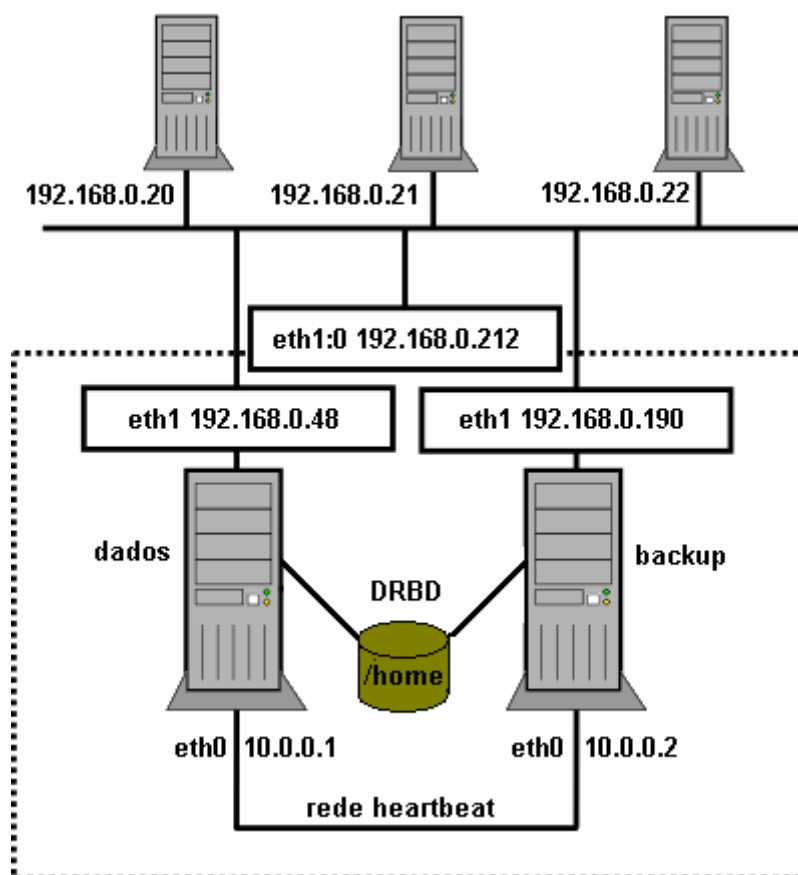


Figura 7: Topologia da rede de alta disponibilidade

Etapa 3 – Definição do sistema operacional e dos softwares de cluster

O sistema operacional escolhido para ser instalado nos dois servidores foi o Linux (Slackware³ 12.1), o qual foi modificado em ambos os servidores para trabalhar de forma homogênea.

² Gigabit: termo que descreve transmissão de quadros em uma rede definido no padrão IEEE 802.3-2005.

³ Slackware: distribuição Linux.

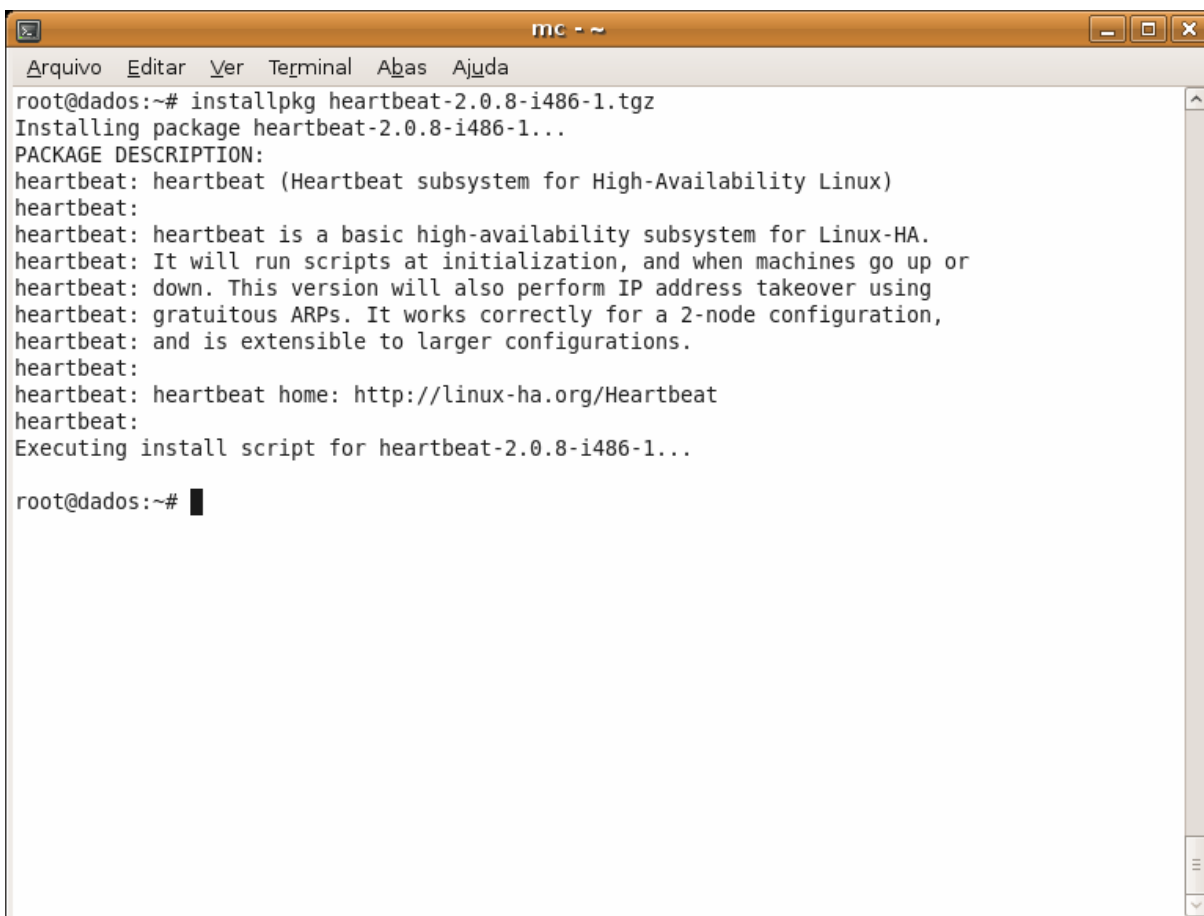
O Heartbeat, versão 2.0.8, foi escolhido para atuar como monitor dos servidores (nodos).

O DRBD, versão 2.8.6, foi escolhido para realizar a replicação dos dados.

Tanto o Heartbeat como o DRBD são softwares livres, porém, não fazem parte da distribuição utilizada, o Slackware, versão 12.1, entretanto, scripts⁴ para compilação e criação dos pacotes foram utilizados.

Etapa 4 – Instalação e configuração do sistema

A Figura 8 mostra o processo de instalação do pacote Heartbeat no sistema. É criado um subdiretório chamado `ha.d` no diretório `/etc` contendo os arquivos necessários para o funcionamento do Heartbeat. Vale ressaltar que os arquivos `ha.cf`⁵ e `haresources`⁶ devem ser idênticos nos dois nodos. O script para iniciar e parar o serviço está localizado em `/etc/rc.d/rc.heartbeat`



```
mc - ~
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
root@dados:~# installpkg heartbeat-2.0.8-i486-1.tgz
Installing package heartbeat-2.0.8-i486-1...
PACKAGE DESCRIPTION:
heartbeat: heartbeat (Heartbeat subsystem for High-Availability Linux)
heartbeat:
heartbeat: heartbeat is a basic high-availability subsystem for Linux-HA.
heartbeat: It will run scripts at initialization, and when machines go up or
heartbeat: down. This version will also perform IP address takeover using
heartbeat: gratuitous ARPs. It works correctly for a 2-node configuration,
heartbeat: and is extensible to larger configurations.
heartbeat:
heartbeat: heartbeat home: http://linux-ha.org/Heartbeat
heartbeat:
Executing install script for heartbeat-2.0.8-i486-1...
root@dados:~#
```

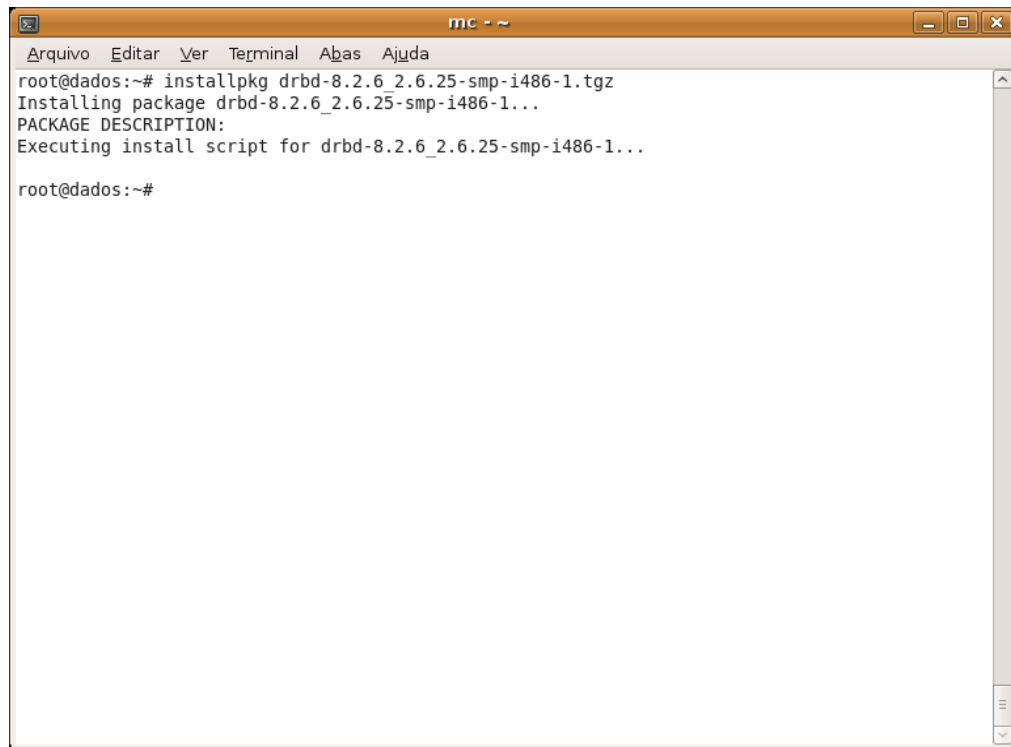
Figura 8: Instalação do pacote Heartbeat no Slackware

⁴ Scripts em anexo

⁵ `ha.cf` em anexo

⁶ `haresources` em anexo

Em seguida foi instalado o pacote DRBD no sistema, como mostra logo abaixo, a Figura 9.



```

mc - ~
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
root@dados:~# installpkg drbd-8.2.6_2.6.25-smp-i486-1.tgz
Installing package drbd-8.2.6_2.6.25-smp-i486-1...
PACKAGE DESCRIPTION:
Executing install script for drbd-8.2.6_2.6.25-smp-i486-1...

root@dados:~#

```

Figura 9: Instalação do pacote DRBD no Slackware

Após a instalação do pacote, foi copiado o arquivo `drbd.conf`:

```
cp /usr/doc/drbd-8.2.6/drbd.conf /etc
```

No arquivo `hosts`, localizado no diretório `/etc`, foi incluído as linhas abaixo:

```
10.0.0.1    dados
10.0.0.2    backup
```

Onde:

10.0.0.1 corresponde ao nodo primário e 10.0.0.2 corresponde ao nodo secundário. A Figura 7 mostra com mais detalhes como ficou todo o esquema.

O arquivo `drbd.conf` deve ser idêntico em ambos os nodos.

Na instalação do sistema operacional são definidas as partições e seus respectivos sistemas de arquivos. A partição `/sda1` foi formatada e preparada para usar o sistema de arquivo `ext3` e que será montada no sistema como `/home`. Para que pudesse funcionar a replicação dessa partição, a mesma foi recriada após a instalação do sistema operacional e inibido a linha de montagem automática do arquivo `fstab`, localizado no diretório `/etc`, colocando-se apenas o símbolo `#` (cerquilha) no início da mesma. Essa linha faz com que a partição `/dev/sda1` seja montada automaticamente na inicialização do sistema. A linha comentada é mostrada logo abaixo:

```
# /dev/sda1 /home ext3 defaults 1 2
```

No lugar, foi acrescentada a linha abaixo:

```
/dev/drbd0 /home ext3 noauto,acl 0 0
```

O comando abaixo foi utilizado para recriar a partição `/dev/sda1`. Isso se faz necessário para que o DRBD consiga criar o seu dispositivo, no caso aqui o `drbd0`.

```
dd if=/dev/zero of=/dev/sda1 bs=1M count=128
```

Foi utilizado o script `/etc/rc.d/rc.drbd7` para iniciar o serviço. O script já carrega o módulo e cria o dispositivo. Os comandos abaixo foram realizados no servidor primário:

Definido como unidade primária:

```
drbdadm primary all
```

Formatado o dispositivo utilizando o sistema de arquivos ext3:

```
mkfs.ext3 /dev/drbd0
```

Montado o dispositivo em `/home`

```
mount /dev/drbd0 /home
```

Os mesmos procedimentos foram realizados no servidor secundário, alterando-se apenas o comando para definir a unidade como secundária.

```
drbdadm secondary all
```

Os procedimentos acima foram realizados apenas uma vez. O script `/etc/rc.d/rc.drbd` foi colocado para ser iniciado automaticamente e que terá como função criar os dispositivos (`drbd0`, `drbd1...`) e ver se há sincronia entre os nodos. O Heartbeat por sua vez, na inicialização do sistema, verifica a disponibilidade entre os nodos e automaticamente aciona o script `drbd8`, que definirá quem será o primário ou secundário.

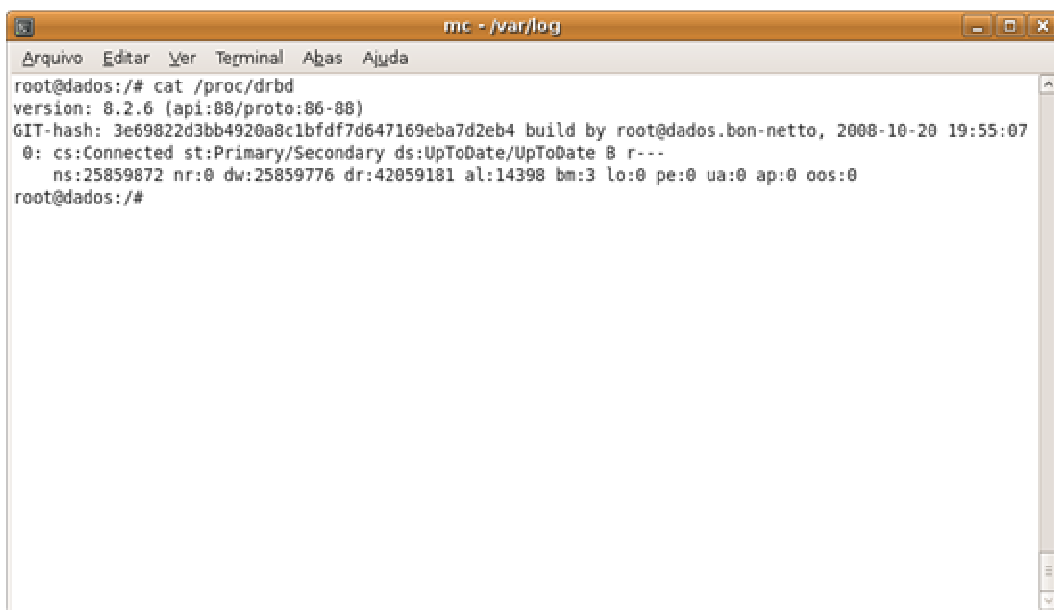
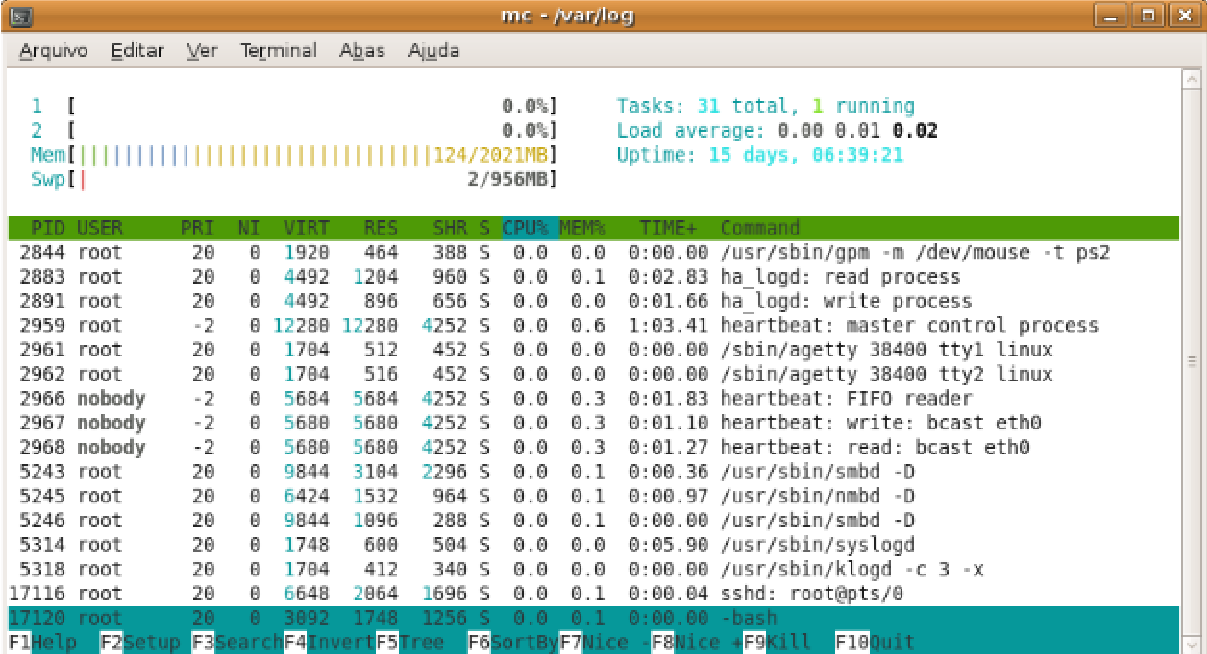


Figura 10: DRBD sincronizado entre os nodos

⁷ rc.drbd em anexo

⁸ drbd em anexo

Após iniciado o sistema, foi verificado se tudo procedeu corretamente. A uso do comando de monitoração de serviço mostra, na Figura 11, que o serviço Heartbeat está funcionando. O PID 2966, 2967 e 2968 corresponde ao Heartbeat.



```

mc - /var/log
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

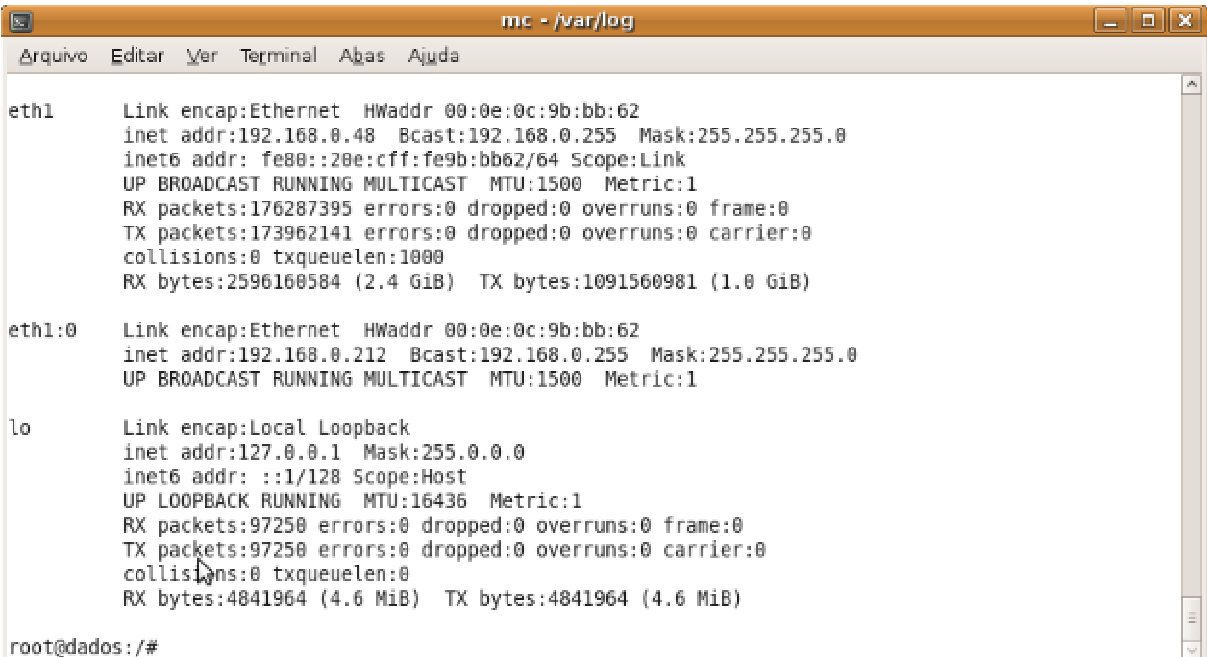
1 [          0.0%]   Tasks: 31 total, 1 running
2 [          0.0%]   Load average: 0.00 0.01 0.02
Mem[|||||124/2021MB] Uptime: 15 days, 06:39:21
Swp[|2/956MB]

PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
2844 root        20   0   1920    464   388  S   0.0  0.0   0:00.00 /usr/sbin/gpm -m /dev/mouse -t ps2
2883 root        20   0   4492   1204   960  S   0.0  0.1   0:02.83 ha_logd: read process
2891 root        20   0   4492    896   656  S   0.0  0.0   0:01.66 ha_logd: write process
2959 root        -2   0  12280  12280  4252  S   0.0  0.6   1:03.41 heartbeat: master control process
2961 root        20   0   1704    512   452  S   0.0  0.0   0:00.00 /sbin/agetty 38400 tty1 linux
2962 root        20   0   1704    516   452  S   0.0  0.0   0:00.00 /sbin/agetty 38400 tty2 linux
2966 nobody      -2   0   5684   5684  4252  S   0.0  0.3   0:01.83 heartbeat: FIFO reader
2967 nobody      -2   0   5680   5680  4252  S   0.0  0.3   0:01.10 heartbeat: write: bcast eth0
2968 nobody      -2   0   5680   5680  4252  S   0.0  0.3   0:01.27 heartbeat: read: bcast eth0
5243 root        20   0   9844   3104  2296  S   0.0  0.1   0:00.36 /usr/sbin/smbd -D
5245 root        20   0   6424   1532   964  S   0.0  0.1   0:00.97 /usr/sbin/nmbd -D
5246 root        20   0   9844   1096   288  S   0.0  0.1   0:00.00 /usr/sbin/smbd -D
5314 root        20   0   1748    600   504  S   0.0  0.0   0:05.90 /usr/sbin/syslogd
5318 root        20   0   1704    412   340  S   0.0  0.0   0:00.00 /usr/sbin/klogd -c 3 -x
17116 root        20   0   6648   2064  1696  S   0.0  0.1   0:00.04 sshd: root@pts/0
17120 root        20   0   3092   1748  1256  S   0.0  0.1   0:00.00 -bash
F1Help  F2Setup  F3Search F4Invert F5Free   F6SortBy F7Nice  F8Nice  F9Kill  F10Quit

```

Figura 11: Monitoração de serviços mostra Heartbeat ativo

A Figura 12 mostra a interface virtual criada pelo Heartbeat, a eth1:0, com o IP do servidor. O nodo secundário assumirá a mesma interface caso o nodo primário falhar.



```

mc - /var/log
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

eth1    Link encap:Ethernet  HWaddr 00:0e:0c:9b:bb:62
        inet addr:192.168.0.48 Bcast:192.168.0.255 Mask:255.255.255.0
        inet6 addr: fe80::20e:cff:fe9b:bb62/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:176287395 errors:0 dropped:0 overruns:0 frame:0
        TX packets:173962141 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:2596160584 (2.4 GiB)  TX bytes:1091560981 (1.0 GiB)

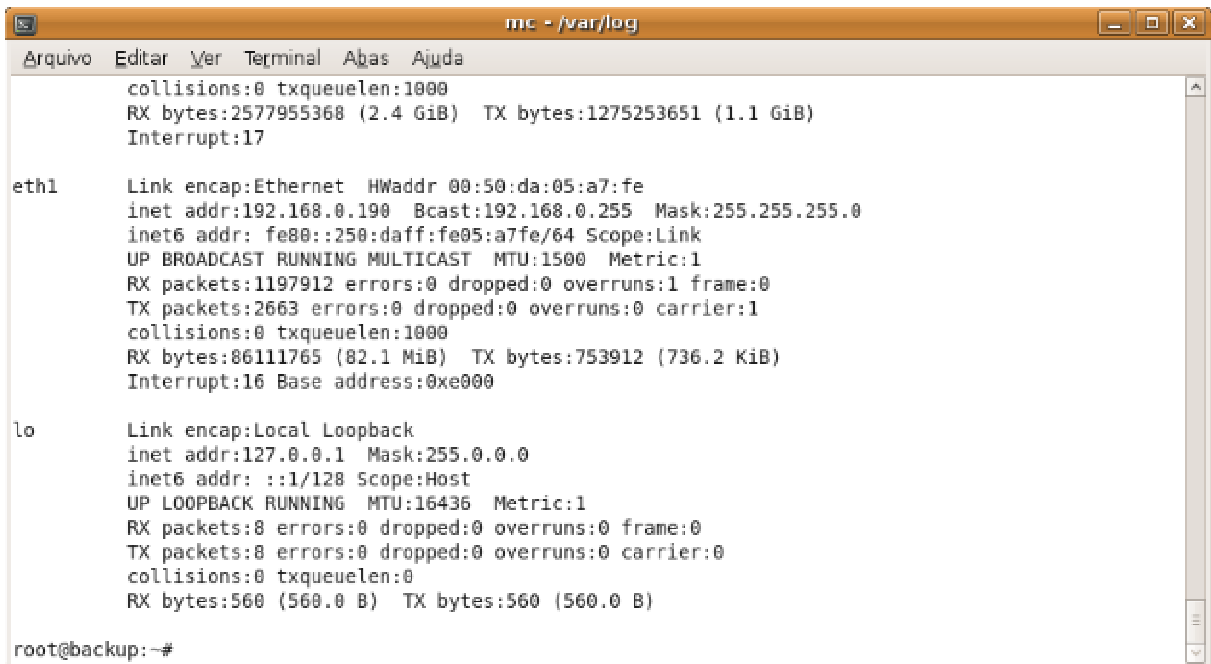
eth1:0  Link encap:Ethernet  HWaddr 00:0e:0c:9b:bb:62
        inet addr:192.168.0.212 Bcast:192.168.0.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:97250 errors:0 dropped:0 overruns:0 frame:0
        TX packets:97250 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:4841964 (4.6 MiB)  TX bytes:4841964 (4.6 MiB)

root@dados: /#

```

Figura 12: Interface virtual criada pelo Heartbeat



```

mc - /var/log
Arquivo  Editar  Ver  Terminal  Abas  Ajuda

collisions:0 txqueuelen:1000
RX bytes:2577955368 (2.4 GiB) TX bytes:1275253651 (1.1 GiB)
Interrupt:17

eth1    Link encap:Ethernet  HWaddr 00:50:da:05:a7:fe
        inet addr:192.168.0.190 Bcast:192.168.0.255 Mask:255.255.255.0
        inet6 addr: fe80::250:daff:fe05:a7fe/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1197912 errors:0 dropped:0 overruns:1 frame:0
        TX packets:2663 errors:0 dropped:0 overruns:0 carrier:1
        collisions:0 txqueuelen:1000
        RX bytes:86111765 (82.1 MiB) TX bytes:753912 (736.2 KiB)
        Interrupt:16 Base address:0xe000

lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:8 errors:0 dropped:0 overruns:0 frame:0
        TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:560 (560.0 B) TX bytes:560 (560.0 B)

root@backup:~#

```

Figura 13: A interface virtual não existe nodo secundário

Etapa 5 – Testes iniciais

A Figura 14 exibe o resultado da pesquisa pelo servidor através de uma estação na rede e mostra claramente que o nodo primário está ativo, através da descrição (MASTER).

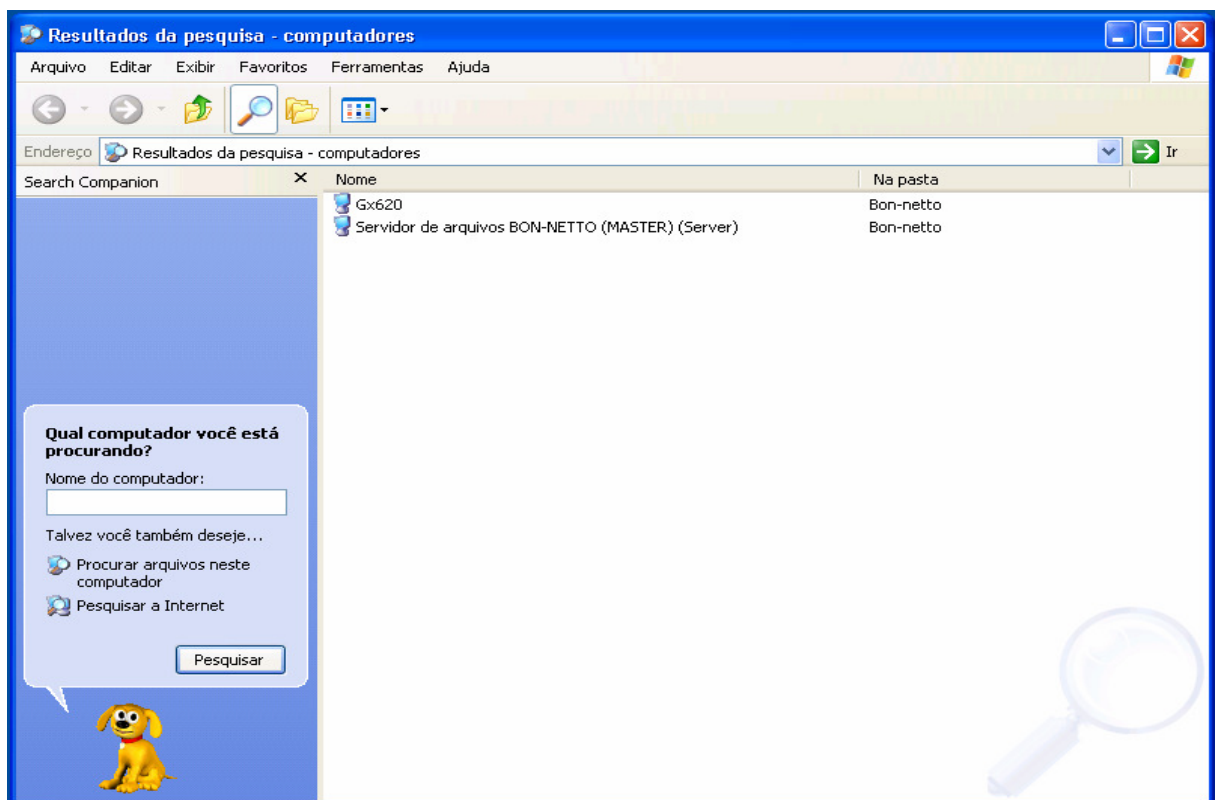


Figura 14: Nodo primário em atividade

A Figura 15 exibe o conteúdo da unidade compartilhada do nodo primário.

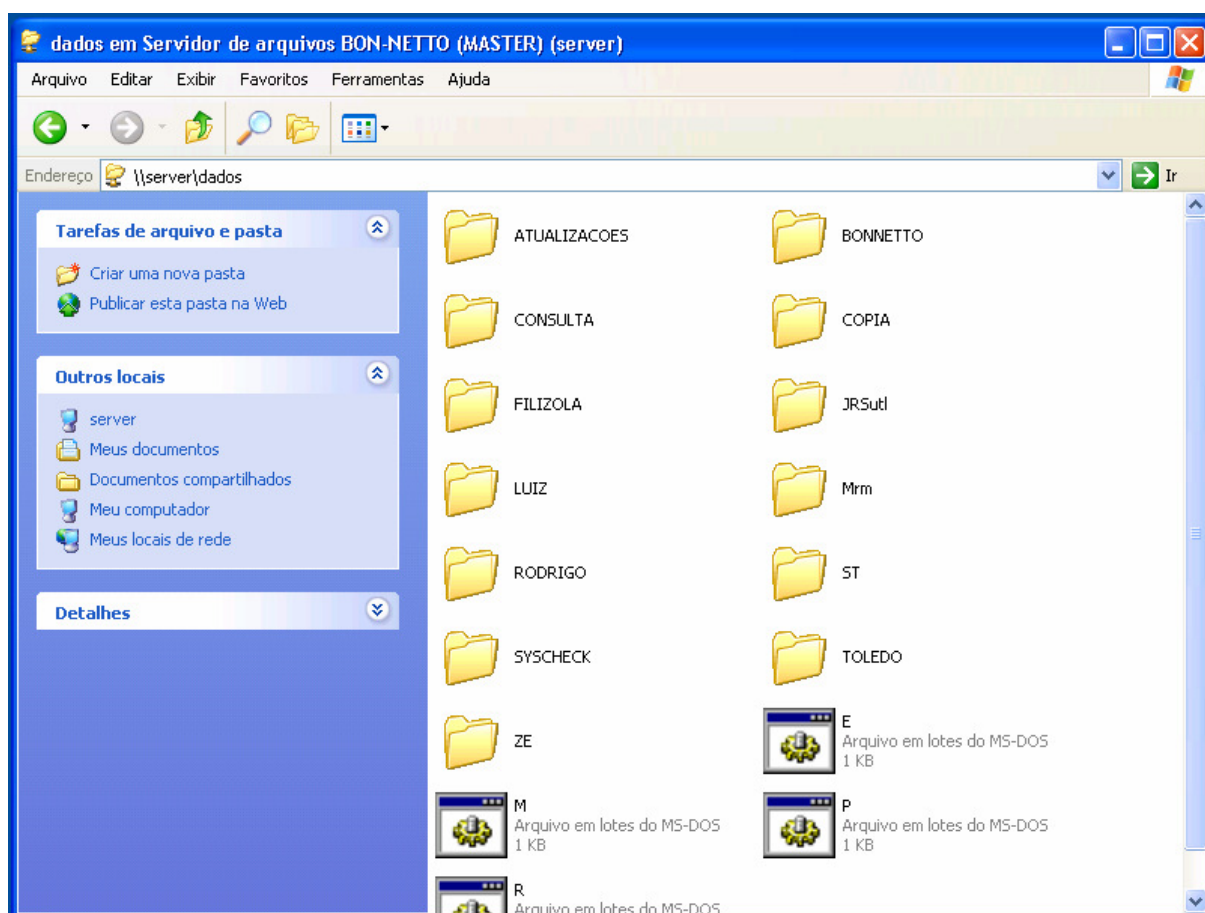


Figura 15: Conteúdo do nodo primário

Afim de verificar se o sistema estava realmente funcionando, foi provocado uma falha e verificado se o failover ocorreria com sucesso. Após aproximadamente 15 segundos o servidor secundário assumiu a posição do servidor primário. Para confirmar se o serviço estava executando no nodo secundário, a mesma pesquisa pelo servidor foi realizada novamente. A Figura 16 mostra no resultado da pesquisa que não mais aparece a descrição MASTER e sim a descrição ESCRAVO, que faz referência ao nodo secundário.

O SAMBA é o software que faz com que estações Windows visualizem na rede computadores Linux, e geralmente é utilizado como servidor de arquivos/impressão. O arquivo de configuração do SAMBA que está no nodo primário muda em relação ao arquivo que está no nodo secundário somente o parâmetro que informa a descrição do servidor. No nodo primário está descrito MASTER e no nodo secundário está descrito ESCRAVO.

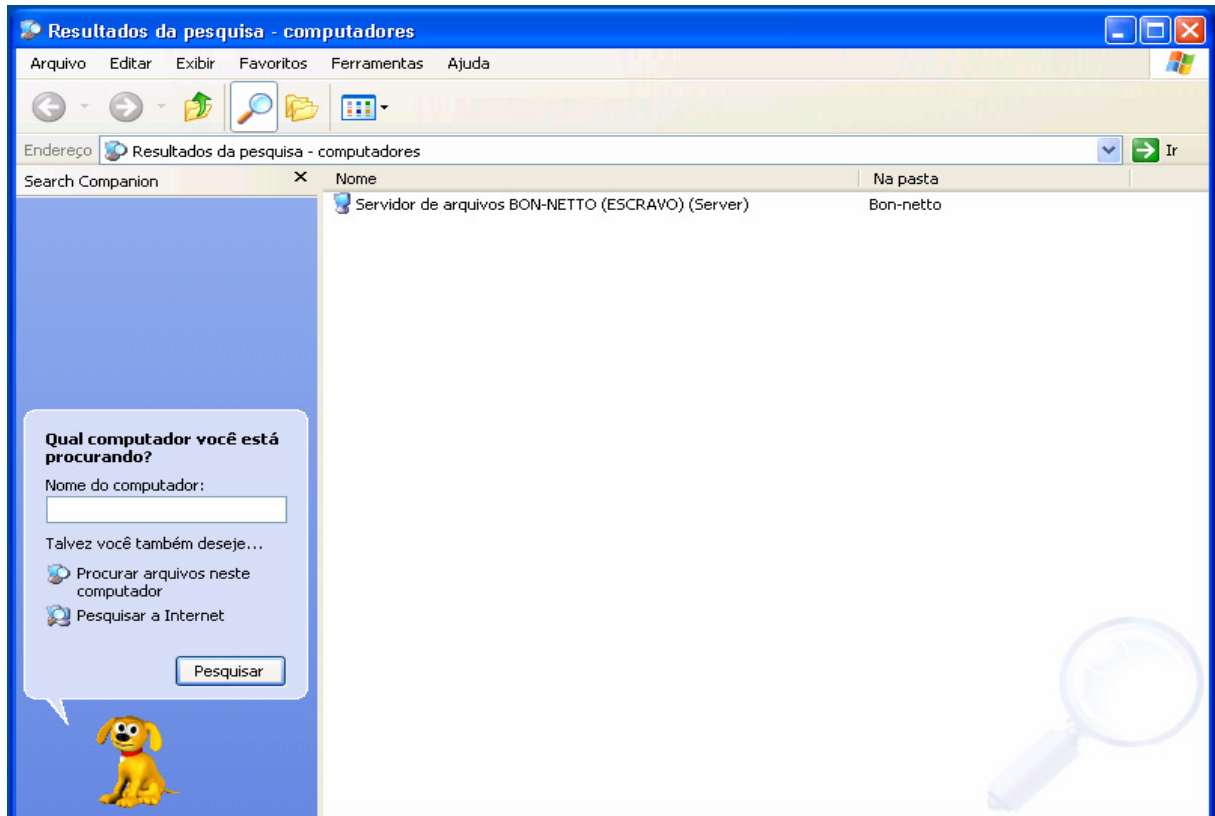


Figura 16: Nodo secundário em atividade

A Figura 17 mostra conteúdo idêntico ao que estava no nodo primário, porém esses dados estão no nodo secundário.

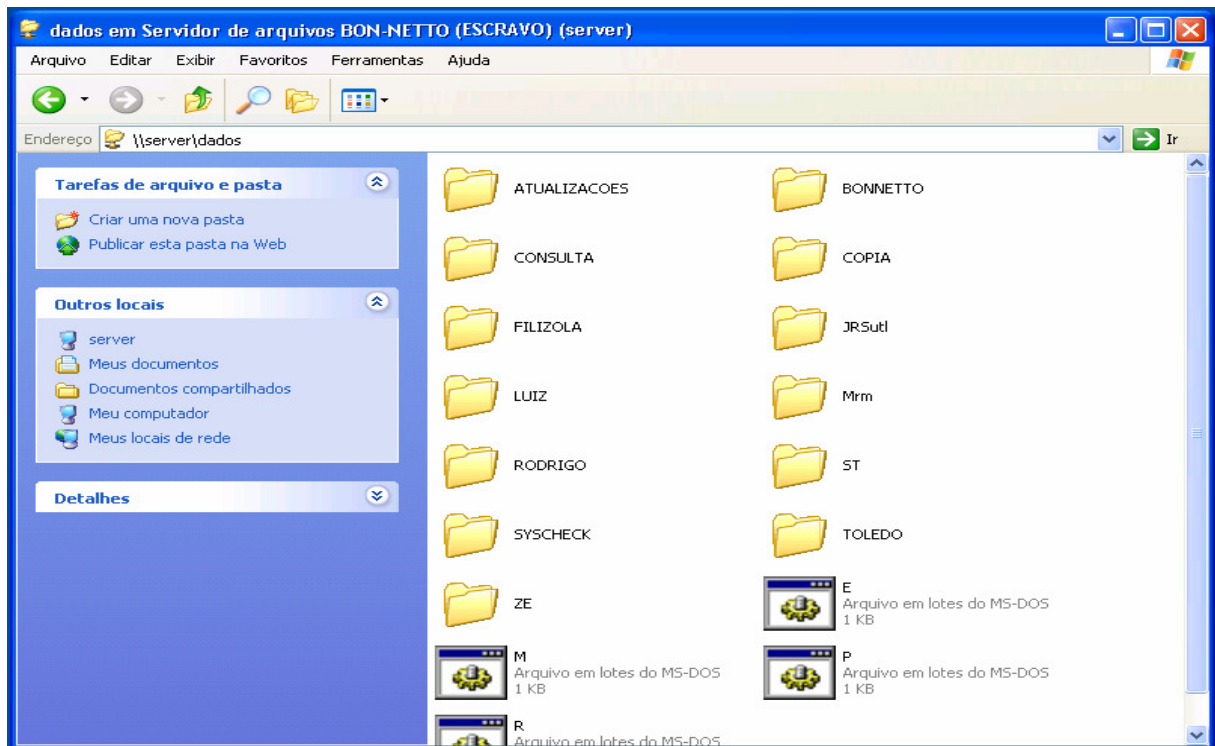
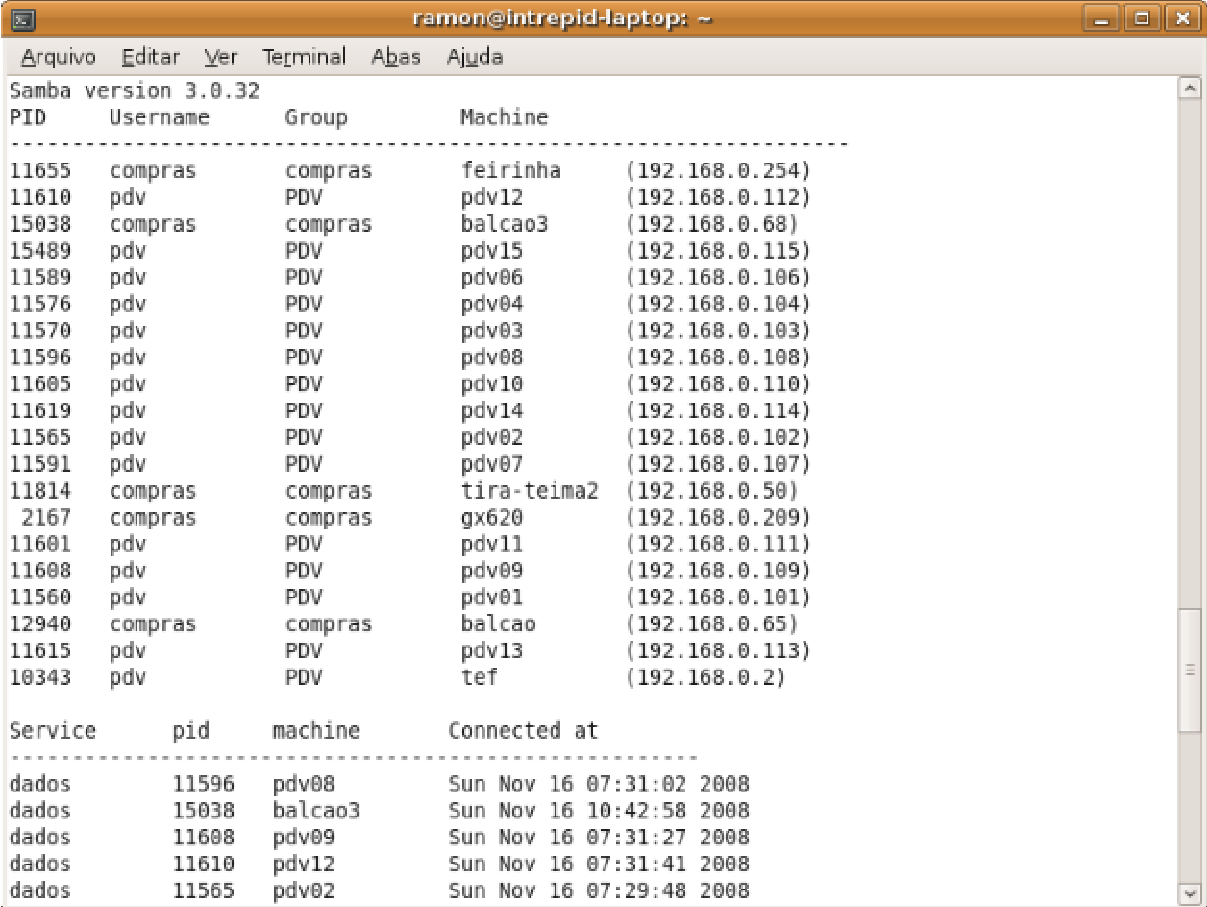


Figura 17: Conteúdo no nodo secundário

Etapa 6 – Implantação definitiva



Samba version 3.0.32

PID	Username	Group	Machine	
11655	compras	compras	feirinha	(192.168.0.254)
11610	pdv	PDV	pdv12	(192.168.0.112)
15038	compras	compras	balcao3	(192.168.0.68)
15489	pdv	PDV	pdv15	(192.168.0.115)
11589	pdv	PDV	pdv06	(192.168.0.106)
11576	pdv	PDV	pdv04	(192.168.0.104)
11570	pdv	PDV	pdv03	(192.168.0.103)
11596	pdv	PDV	pdv08	(192.168.0.108)
11605	pdv	PDV	pdv10	(192.168.0.110)
11619	pdv	PDV	pdv14	(192.168.0.114)
11565	pdv	PDV	pdv02	(192.168.0.102)
11591	pdv	PDV	pdv07	(192.168.0.107)
11814	compras	compras	tira-teima2	(192.168.0.50)
2167	compras	compras	gx620	(192.168.0.209)
11601	pdv	PDV	pdv11	(192.168.0.111)
11608	pdv	PDV	pdv09	(192.168.0.109)
11560	pdv	PDV	pdv01	(192.168.0.101)
12940	compras	compras	balcao	(192.168.0.65)
11615	pdv	PDV	pdv13	(192.168.0.113)
10343	pdv	PDV	tef	(192.168.0.2)

Service	pid	machine	Connected at
dados	11596	pdv08	Sun Nov 16 07:31:02 2008
dados	15038	balcao3	Sun Nov 16 10:42:58 2008
dados	11608	pdv09	Sun Nov 16 07:31:27 2008
dados	11610	pdv12	Sun Nov 16 07:31:41 2008
dados	11565	pdv02	Sun Nov 16 07:29:48 2008

Figura 18: Estações na rede

A Figura 18 exibe várias estações, como PDVs e computadores usados no supermercado usufruindo da nova estrutura, estrutura esta baseada em software livre para prover alta disponibilidade.

4. CONCLUSÃO

O conceito de alta disponibilidade não deve, e nem pode, ser uma novidade para os administradores de sistemas, principalmente em ambientes computacionais onde a disponibilidade é uma característica crítica. Esse conceito, que não é recente, está sendo amplamente disseminado, e ganha importância diretamente proporcional a influência que os sistemas de computação exercem nas empresas ou entidades que sustentam. Quando relacionamos alta disponibilidade com os sistemas Linux, está na sombra desse conceito dois maduros softwares livres que permitem preencher alguns requisitos de sua implementação: o Heartbeat e o DRBD. Ambos são parte integrante do conhecido projeto Linux-HA.

Não foi especificado neste trabalho o uso de monitoração de serviços para alta disponibilidade, cuja finalidade, por exemplo, é ativar o nodo secundário caso o serviço SAMBA no nodo primário venha a falhar.

A utilização desse recurso se faz com o uso do software livre MON. Este recurso será implantado posteriormente.

Este trabalho teve por objetivo mostrar, através de um CASE, que é possível, através do uso de alguns componentes de software livre, implementar uma solução de alta disponibilidade e de baixo custo.

5. REFERÊNCIAS BIBLIOGRÁFICAS

1. DRBD. **Welcome to DRBD**. Disponível em: <<http://www.drbd.org>>. Acessado em: 17 de novembro de 2008.
2. FERREIRA, Filipa et. al. **Clusters de alta disponibilidade – uma abordagem Open Source**. Disponível em <<http://mosel.estg.ipleiria.pt/files/Artigo.pdf>>. Acessado em: 15 de março de 2008.
3. FILHO, Nélio Alves Pereira. **Serviços de Pertinência para Cluster de Alta Disponibilidade**. 2004. 270. Dissertação (Mestre em Ciência da Computação). Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo.
4. HEARTBEAT. **Introdução**. Disponível em: <<http://replicacao.no.sapo.pt/heartbeat.htm#Alta>>. Acessado em: 17 de novembro de 2008.
5. LEVITA, Elton Martins. **Alta Disponibilidade como Alternativa ao Uso de Servidores BDC em Ambientes Samba**. 2005. 43. Monografia (Pós-Graduação em Administração em Redes Linux) – Departamento de Ciência da Computação. Universidade Federal de Lavras, Minas Gerais.
6. SZTOLTZ, Lisiane et. al. **Guia do Servidor Conectiva Linux**. Curitiba: Editora Conectiva, 2003.
7. TRIGO, Clodonil Honório. **OpenLDAP: Uma abordagem integrada**. São Paulo: Novatec Editora, 2007. 239p.
8. ZEM, José Luís. **Uso de cluster de computadores no ambiente corporativo**. Disponível em: <<http://www.unibratex.com.br/anaisdecongresso/diretorio/unimep+jlz%20revisado.doc>>. Acessado em: 29 de maio de 2008.
9. ZENOSS. **Create a HighAvailable zenoss**. Disponível em <<http://www.zenoss.com/Members/netdata/create-a-highavailable-zenoss>>. Acessado em 17 de novembro de 2008.
10. WIKIPEDIA. **Sistema de alta disponibilidade**. Disponível em <<http://pt.wikipedia.org/wiki/Disponibilidade>> . Acessado em: 17 de novembro de 2008.

6. ANEXOS

ANEXO 1

Script heartbeat.Slackbuild para compilação do Heartbeat

```
#!/bin/sh
# $Id: heartbeat.SlackBuild,v 1.1 2007/01/15 19:17:29 root Exp root $
# Copyright (c) 2007 Eric Hamelers <alien@slackware.com>
# -----
----
#
# Slackware SlackBuild script
# =====
# By:      Eric Hamelers <alien@slackware.com>
# For:      heartbeat
# Descr:    Heartbeat subsystem for High-Availability Linux
# URL:      http://linux-ha.org/Heartbeat
# Needs:    libnet
# Changelog:
# 2.0.8-1:  12/Jan/2007 by Eric Hamelers <alien@slackware.com>
#           * Initial build.
#
# Run 'sh heartbeat.SlackBuild --cleanup' to build a Slackware package.
# The package (.tgz) plus descriptive .txt file are created in /tmp .
# Install using 'installpkg'.
#
# -----
----

# --- INIT ---
# Set initial variables:

PRGNAM=heartbeat
VERSION=${VERSION:-2.0.8}
ARCH=${ARCH:-i486}
BUILD=${BUILD:-1}

# the user:group "hacluster:haclient" uses the id's "90:90" in the RPMs
that
# Heartbeat creates. Try to keep to the same idnumbers here:
HUID=90
HGID=90

# Not used, heartbeat's build script does it well enough:
#DOCS=""

# Where do we look for sources?
CWD=`pwd`
SRCDIR=`dirname $0`
[ "${SRCDIR:0:1}" == "." ] && SRCDIR=${CWD}/${SRCDIR}

# Place to build (TMP) package (PKG) and output (OUTPUT) the program:
TMP=${TMP:-/tmp/build}
PKG=${TMP}/package-${PRGNAM}
OUTPUT=${OUTPUT:-/tmp}

# Input URL: http://linux-ha.org/download/heartbeat-2.0.8.tar.gz
SOURCE="${SRCDIR}/${PRGNAM}-${VERSION}.tar.gz"
SRCURL="http://linux-ha.org/download/${PRGNAM}-${VERSION}.tar.gz"
```

```

##
## --- with a little luck, you won't have to edit below this point --- ##
##

# Exit the script on errors:
set -e
trap 'echo "$0 FAILED at line $LINENO!" | tee $OUTPUT/error-${PRGNAM}.log'
ERR
# Catch uninitialized variables:
set -u
P1=${1:-1}

# Slackware 11 and up need other option (gcc > 3.3.x)
if [ `gcc -dumpversion | tr -d '.' | cut -c 1-2` -gt 33 ]; then
    MOPT=tune
else
    MOPT=cpu
fi

case "$ARCH" in
    i386)
        SLKCFLAGS="-O2 -march=i386 -m${MOPT}=i686"
        SLKLDFLAGS=""; LIBDIRSUFFIX=""
        ;;
    i486)
        SLKCFLAGS="-O2 -march=i486 -m${MOPT}=i686"
        SLKLDFLAGS=""; LIBDIRSUFFIX=""
        ;;
    s390)
        SLKCFLAGS="-O2"
        SLKLDFLAGS=""; LIBDIRSUFFIX=""
        ;;
    powerpc)
        SLKCFLAGS="-O2"
        SLKLDFLAGS=""; LIBDIRSUFFIX=""
        ;;
    x86_64)
        SLKCFLAGS="-O2 -fPIC"
        SLKLDFLAGS="-L/usr/lib64"; LIBDIRSUFFIX="64"
        ;;
    athlon-xp)
        SLKCFLAGS="-march=athlon-xp -O3 -pipe -fomit-frame-pointer"
        SLKLDFLAGS=""; LIBDIRSUFFIX=""
        ;;
esac

if [ ! -d $TMP/tmp-$PRGNAM ]; then
    mkdir -p $TMP/tmp-$PRGNAM # location to build the source
elif [ "$P1" != "--oldbuild" ]; then
    # If the "--oldbuild" parameter is present, we keep
    # the old build files and continue;
    # By default we remove the remnants of previous build and continue:
    rm -rf $TMP/tmp-$PRGNAM/*
fi

if [ ! -d $PKG ]; then
    mkdir -p $PKG # place for the package to be built
else
    rm -rf $PKG/* # We always erase old package's contents:
fi

if [ ! -d $OUTPUT ]; then
    mkdir -p $OUTPUT # place for the package to be saved
fi

# --- SOURCE FILE AVAILABILITY ---

```

```

if ! [ -f ${SOURCE} ]; then
  if ! [ "x${SRCURL}" == "x" ]; then
    # Check if the $SRCDIR is writable at all - if not, download to $OUTPUT
    [ -w "$SRCDIR" ] || SOURCE="$OUTPUT/`basename $SOURCE`"
    echo "Source '`basename ${SOURCE}`' not available yet..."
    echo "Will download file to `dirname $SOURCE`"
    wget -nv -O "${SOURCE}" "${SRCURL}" || true
    if [ $? -ne 0 ]; then
      echo "Downloading '`basename ${SOURCE}`' failed... aborting the
build."
      mv -f "${SOURCE}" "${SOURCE}.FAIL"
      exit 1
    fi
  else
    echo "File '`basename ${SOURCE}`' not available... aborting the build."
    exit 1
  fi
fi

if [ "$P1" == "--download" ]; then
  echo "Download complete."
  exit 0
fi

# --- PACKAGE BUILDING ---

echo "++"
echo "|| $PRGNAM-$VERSION"
echo "++"

cd $PKG

# Explode the package framework:
if [ -f $SRCDIR/_$PRGNAM.tar.gz ]; then
  explodpkg $SRCDIR/_$PRGNAM.tar.gz
fi

cd $TMP/tmp-$PRGNAM

# --- TARBALL EXTRACTION, PATCH, MODIFY ---

echo "Extracting the source archive(s) for $PRGNAM..."
if `file ${SOURCE} | grep -q ": bzip2"`; then
  tar -xjvf ${SOURCE}
elif `file ${SOURCE} | grep -qi ": zip"`; then
  unzip ${SOURCE}
elif `file ${SOURCE} | grep -qi ": 7-zip"`; then
  7za -x ${SOURCE}
else
  tar -xzvf ${SOURCE}
fi
if [ -d ${PRGNAM}-${VERSION} ]; then
  cd ${PRGNAM}-${VERSION}
else
  cd ${PRGNAM}* # a little less specific
fi

chown -R root.root *
chmod -R u+w,go+r-w,a-s .

```



```

# --- BUILDING ---

echo Building ...

LDFLAGS="$SLKLDFLAGS" \
CFLAGS="$SLKCFLAGS" \
./configure --prefix=/usr \
            --libdir=/usr/lib${LIBDIRSUFFIX} \
            --localstatedir=/var \
            --sysconfdir=/etc \
            --with-group-id=${HGID} --with-ccmuser-id=${HUID} \
            --enable-snmp \
            --enable-bundled_ltdl \
            --with-initdir=/etc/rc.d \
            --program-prefix="" \
            --program-suffix="" \
            $ARCH-slackware-linux \
2>&1 | tee $OUTPUT/configure-${PRGNAM}.log
# The management GUI needs PAM and GnuTLS...
#--enable-mgmt \

make 2>&1 | tee $OUTPUT/make-${PRGNAM}.log

#
# Install all the needed stuff to the package dir
#
# Use installwatch if available, to produce a logfile of the installation
# process that is more easily readable:
if `which installwatch > /dev/null 2>&1`; then
    installwatch -o $OUTPUT/install-${PRGNAM}.log make DESTDIR=$PKG install
else
    make DESTDIR=$PKG install 2>&1 |tee $OUTPUT/install-${PRGNAM}.log
fi

# Add a symlink for ldirectord
ln -s /sbin/ldirectord $PKG/etc/ha.d/resource.d/ldirectord

# Create Slackware rc script names:
mv $PKG/etc/rc.d/heartbeat $PKG/etc/rc.d/rc.heartbeat.new
mv $PKG/etc/rc.d/ldirectord $PKG/etc/rc.d/rc.ldirectord.new

# Add this to the doinst.sh
! [ -d $PKG/install ] && mkdir -p $PKG/install
cat <<EOINS >> $PKG/install/doinst.sh
# Handle the incoming configuration files:
config() {
    for infile in \${1}; do
        NEW="\${infile}"
        OLD="\`dirname \${NEW}\`/\`basename \${NEW} .new\`"
        # If there's no config file by that name, mv it over:
        if [ ! -r \${OLD} ]; then
            mv \${NEW} \${OLD}
        elif [ "\`cat \${OLD} | md5sum\`" = "\`cat \${NEW} | md5sum\`" ]; then
            # toss the redundant copy
            rm \${NEW}
        fi
        # Otherwise, we leave the .new copy for the admin to consider...
    done
}

config etc/rc.d/rc.heartbeat.new

```

```

config etc/rc.d/rc.ldirectord.new

# Create heartbeat user/group :
if chroot . getent group haclient >/dev/null
then
    : OK group haclient already present
else
    GROUPOPT="-g $HGID"
    if chroot . groupadd \${GROUPOPT} haclient 2>/dev/null
    then
        : OK we were able to add group haclient
    else
        chroot . groupadd haclient
    fi
fi

if chroot . getent passwd hacluster >/dev/null
then
    : OK user hacluster already present
else
    USEROPT="-g haclient -u $HUID -d /var/lib/heartbeat/cores/hacluster"
    if chroot . useradd \${USEROPT} hacluster 2>/dev/null
    then
        : OK we were able to add user hacluster
    else
        chroot . useradd hacluster
    fi
fi

# Now, set the correct permissions on the installed stuff:
chown root: /var/lib/heartbeat/cores/root
chmod 0700 /var/lib/heartbeat/cores/root

chown nobody: /var/lib/heartbeat/cores/nobody
chmod 0700 /var/lib/heartbeat/cores/nobody

chown hacluster: /var/lib/heartbeat/cores/hacluster
chmod 0700 /var/lib/heartbeat/cores/hacluster

chown hacluster:haclient /usr/bin/cl_status
chmod 2555 /usr/bin/cl_status

chown hacluster:haclient /var/run/heartbeat/ccm
chmod 750 /var/run/heartbeat/ccm

chown hacluster:haclient /var/run/heartbeat/crm
chmod 750 /var/run/heartbeat/crm

chown hacluster:haclient /var/lib/heartbeat/pengine
chmod 750 /var/lib/heartbeat/pengine

EOINS

# --- DOCUMENTATION ---

mv $PKG/usr/share/doc $PKG/usr/
chmod -R a-w $PKG/usr/doc/${PRGNAM}-${VERSION}/*

# Move incorrectly installed man pages, if any
[ -d $PKG/usr/share/man ] && \
    mv $PKG/usr/share/man $PKG/usr/ && rmdir $PKG/usr/share || true

```

```

# Compress the man page(s)
if [ -d $PKG/usr/man ]; then
    find $PKG/usr/man -type f -name "*.?" -exec gzip -9f {} \;
    for i in `find $PKG/usr/man -type l -name "*.?"`; do ln -s $( readlink
$i ).gz $i.gz ; rm $i ; done
fi
rmdir $PKG/usr/share || true

# Strip binaries
( cd $PKG
    find . | xargs file | grep "executable" | grep ELF | cut -f 1 -d : |
xargs strip --strip-unneeded 2> /dev/null
    find . | xargs file | grep "shared object" | grep ELF | cut -f 1 -d : |
xargs strip --strip-unneeded 2> /dev/null
)

# --- OWNERSHIP, RIGHTS ---

chmod -R o-w $PKG

# --- PACKAGE DESCRIPTION ---

mkdir -p $PKG/install
cat $SRCDIR/slack-desc > $PKG/install/slack-desc
if [ -f $SRCDIR/doinst.sh ]; then
    cat $SRCDIR/doinst.sh >> $PKG/install/doinst.sh
fi
if [ -f $SRCDIR/slack-required ]; then
    cat $SRCDIR/slack-required > $PKG/install/slack-required
fi

# --- BUILDING ---

# Build the package:
cd $PKG
makepkg --linkadd y --chown n $OUTPUT/${PRGNAM}-${VERSION}-${ARCH}-
${BUILD}.tgz \
    2>&1 | tee $OUTPUT/makepkg-${PRGNAM}.log
(cd $OUTPUT && md5sum ${PRGNAM}-${VERSION}-${ARCH}-${BUILD}.tgz >
${PRGNAM}-${VERSION}-${ARCH}-${BUILD}.tgz.md5)
cat $PKG/install/slack-desc | grep "^${PRGNAM}" > $OUTPUT/${PRGNAM}-
${VERSION}-${ARCH}-${BUILD}.txt
if [ -f $PKG/install/slack-required ]; then
    cat $PKG/install/slack-required > $OUTPUT/${PRGNAM}-${VERSION}-${ARCH}-
${BUILD}.dep
fi

# --- CLEANUP ---

# Clean up the extra stuff:
if [ "$P1" = "--cleanup" ]; then
    rm -rf $TMP/tmp-${PRGNAM}
    rm -rf $PKG
fi

```

ANEXO 2

Script drbd.Slackbuild para compilação do DRBD

```
#!/bin/sh
# $Id: drbd.SlackBuild,v 1.1 2007/01/12 22:09:50 root Exp root $
# Copyright (c) 2007 Eric Hameleers <alien@slackware.com>
# -----
----
#
# Slackware SlackBuild script
# =====
# By:      Eric Hameleers <alien@slackware.com>
# For:      drbd
# Descr:    a network block device for high availability clusters
# URL:      http://www.drbd.org/
# Needs:    Linux kernel 2.6.x
# Changelog:
# 0.7.23-1: 12/Jan/2007 by Eric Hameleers <alien@slackware.com>
#           * Initial build.
#
# Run 'sh drbd.SlackBuild --cleanup' to build a Slackware package.
# The package (.tgz) plus descriptive .txt file are created in /tmp .
# Install using 'installpkg'.
#
# -----
----

# --- INIT ---
# Set initial variables:

PRGNAM=drbd
VERSION=${VERSION:-8.2.6}
ARCH=${ARCH:-i486}
BUILD=${BUILD:-1}

DOCS="ChangeLog COPYING INSTALL README *.txt scripts/drbd.conf"

# Where do we look for sources?
CWD=`pwd`
SRCDIR=`dirname $0`
[ "${SRCDIR:0:1}" == "." ] && SRCDIR=${CWD}/${SRCDIR}

# Place to build (TMP) package (PKG) and output (OUTPUT) the program:
TMP=${TMP:-/tmp/build}
PKG=${TMP}/package-${PRGNAM}
OUTPUT=${OUTPUT:-/tmp}

# Kernel module related parameters
KVER=${KVER:-`uname -r`}
KSRC=${KSRC:-/lib/modules/${KVER}/build}
PATCHLEVEL=`echo $KVER|cut -f 2 -d '.'`
[ $PATCHLEVEL -eq 4 ] && MODCONFFILE=modules.conf ||
MODCONFFILE=modprobe.conf

# Input URL: http://oss.linbit.com/drbd/0.7/drbd-0.7.23.tar.gz
SOURCE="${SRCDIR}/${PRGNAM}-${VERSION}.tar.gz"
SRCURL="http://oss.linbit.com/${PRGNAM}/0.7/${PRGNAM}-${VERSION}.tar.gz"

##
```

```

## --- with a little luck, you won't have to edit below this point --- ##
##

# Exit the script on errors:
set -e
trap 'echo "$0 FAILED at line $LINENO!" | tee $OUTPUT/error-${PRGNAM}.log'
ERR
# Catch uninitialized variables:
set -u
P1=${1:-1}

# Slackware 11 and up need other option (gcc > 3.3.x)
if [ `gcc -dumpversion | tr -d '.' | cut -c 1-2` -gt 33 ]; then
    MOPT=tune
else
    MOPT=cpu
fi

case "$ARCH" in
    i386)      SLKCFLAGS="-O2 -march=i386 -m${MOPT}=i686"
               SLKLDFLAGS=""; LIBDIRSUFFIX=""
               ;;
    i486)      SLKCFLAGS="-O2 -march=i486 -m${MOPT}=i686"
               SLKLDFLAGS=""; LIBDIRSUFFIX=""
               ;;
    s390)      SLKCFLAGS="-O2"
               SLKLDFLAGS=""; LIBDIRSUFFIX=""
               ;;
    powerpc)   SLKCFLAGS="-O2"
               SLKLDFLAGS=""; LIBDIRSUFFIX=""
               ;;
    x86_64)    SLKCFLAGS="-O2 -fPIC"
               SLKLDFLAGS="-L/usr/lib64"; LIBDIRSUFFIX="64"
               ;;
    athlon-xp) SLKCFLAGS="-march=athlon-xp -O3 -pipe -fomit-frame-pointer"
               SLKLDFLAGS=""; LIBDIRSUFFIX=""
               ;;
esac

if [ ! -d $TMP/tmp-$PRGNAM ]; then
    mkdir -p $TMP/tmp-$PRGNAM # location to build the source
elif [ "$P1" != "--oldbuild" ]; then
    # If the "--oldbuild" parameter is present, we keep
    # the old build files and continue;
    # By default we remove the remnants of previous build and continue:
    rm -rf $TMP/tmp-$PRGNAM/*
fi

if [ ! -d $PKG ]; then
    mkdir -p $PKG # place for the package to be built
else
    rm -rf $PKG/* # We always erase old package's contents:
fi

if [ ! -d $OUTPUT ]; then
    mkdir -p $OUTPUT # place for the package to be saved
fi

# --- SOURCE FILE AVAILABILITY ---

if ! [ -f ${SOURCE} ]; then

```

```

if ! [ "x${SRCURL}" == "x" ]; then
    # Check if the $SRCDIR is writable at all - if not, download to $OUTPUT
    [ -w "$SRCDIR" ] || SOURCE="$OUTPUT/`basename $SOURCE`"
    echo "Source '`basename ${SOURCE}`' not available yet..."
    echo "Will download file to `dirname $SOURCE`"
    wget -nv -O "${SOURCE}" "${SRCURL}" || true
    if [ $? -ne 0 ]; then
        echo "Downloading '`basename ${SOURCE}`' failed... aborting the
build."
        mv -f "${SOURCE}" "${SOURCE}.FAIL"
        exit 1
    fi
else
    echo "File '`basename ${SOURCE}`' not available... aborting the build."
    exit 1
fi
fi

if [ "$P1" == "--download" ]; then
    echo "Download complete."
    exit 0
fi

# --- PACKAGE BUILDING ---

echo "++"
echo "|| $PRGNAM-$VERSION"
echo "++"

cd $TMP/tmp-$PRGNAM

# --- TARBALL EXTRACTION, PATCH, MODIFY ---

echo "Extracting the source archive(s) for $PRGNAM..."
if `file ${SOURCE} | grep -q ": bzip2"`; then
    tar -xjvf ${SOURCE}
else
    tar -xzvf ${SOURCE}
fi
cd ${PRGNAM}-${VERSION}

chown -R root.root *
find . -perm 777 -exec chmod 755 {} \;
find . -perm 666 -exec chmod 644 {} \;

# --- BUILDING ---

echo Building ...

export LDFLAGS="$SLKLDLDFLAGS"
export CFLAGS="$SLKCFLAGS"
make all doc PREFIX=$PKG/ MANDIR=/usr/man KDIR=${KSRC} \
    2>&1 | tee $OUTPUT/make-${PRGNAM}.log

make install PREFIX=$PKG/ MANDIR=/usr/man \
    2>&1 |tee $OUTPUT/install-${PRGNAM}.log

# Only install a hint to the example conf
cat <<EOT > $PKG/etc/drbd.conf
#

```

```

# please have a a look at the example configuration file in
# /usr/doc/${PRGNAM}-${VERSION}/drbd.conf
#
EOT

# Fix the configuration files so they won't overwrite existing ones
# on package install:
mv $PKG/etc/drbd.conf{,.new}

# Rename the init script to something Slackware compatible:
mv $PKG/etc/rc.d/drbd $PKG/etc/rc.d/rc.drbd

# Add this to the doinst.sh
! [ -d $PKG/install ] && mkdir -p $PKG/install
cat <<EOINS >> $PKG/install/doinst.sh
# Handle the incoming configuration files:
config() {
    for infile in \${1}; do
        NEW="\${infile}"
        OLD="\`dirname \${NEW}\`/\`basename \${NEW} .new\`"
        # If there's no config file by that name, mv it over:
        if [ ! -r \${OLD} ]; then
            mv \${NEW} \${OLD}
        elif [ "\`cat \${OLD} | md5sum\`" = "\`cat \${NEW} | md5sum\`" ]; then
            # toss the redundant copy
            rm \${NEW}
        fi
        # Otherwise, we leave the .new copy for the admin to consider...
    done
}

config etc/drbd.conf.new

EOINS

# Add this to the doinst.sh
! [ -d $PKG/install ] && mkdir -p $PKG/install
cat <<EEOOTT >> $PKG/install/doinst.sh
    # Only run depmod on matching running kernel
    # Slackware will run depmod anyway on reboot):
    MYMODVER=$KVER
    MYKERNEL=\`uname -r\`
    if [ "\${MYKERNEL}" = "\${MYMODVER}" ]; then
        if [ -x sbin/depmod ]; then
            chroot . /sbin/depmod -a \${MYKERNEL} 1> /dev/null 2> /dev/null
        fi
    fi

    EEOOTT

# --- DOCUMENTATION ---

mkdir -p $PKG/usr/doc/${PRGNAM}-${VERSION}
cp -a $DOCS $PKG/usr/doc/${PRGNAM}-${VERSION} || true
chmod -R a-w $PKG/usr/doc/${PRGNAM}-${VERSION}/*

# Move incorrectly installed man pages, if any
if [ -d $PKG/usr/share/man ]; then
    mv $PKG/usr/share/man $PKG/usr/ && rmdir $PKG/usr/share || true
fi
# Compress the man page(s)

```

```

if [ -d $PKG/usr/man ]; then
    find $PKG/usr/man -type f -name "*.?" -exec gzip -9f {} \;
    for i in `find $PKG/usr/man -type l -name "*.?"`; do ln -s $( readlink
$i ).gz $i.gz ; rm $i ; done
fi

# Strip binaries
( cd $PKG
    find . | xargs file | grep "executable" | grep ELF | cut -f 1 -d : |
xargs strip --strip-unneeded 2> /dev/null
    find . | xargs file | grep "shared object" | grep ELF | cut -f 1 -d : |
xargs strip --strip-unneeded 2> /dev/null
)

# Compress the kernel modules
[ $PATCHLEVEL -eq 4 ] && \
    find $PKG/lib/modules -type f -name "*.?" -exec gzip -9 {} \;

# --- OWNERSHIP, RIGHTS ---

chmod -R o-w $PKG

# --- PACKAGE DESCRIPTION ---

mkdir -p $PKG/install
cat $SRCDIR/slack-desc > $PKG/install/slack-desc
if [ -f $SRCDIR/doinst.sh ]; then
    cat $SRCDIR/doinst.sh >> $PKG/install/doinst.sh
fi

# --- BUILDING ---

# Build the package:
cd $PKG
makepkg --linkadd y --chown n $OUTPUT/${PRGNAM}-${VERSION}_${KVER}-${ARCH}-
${BUILD}.tgz \
    2>&1 | tee $OUTPUT/makepkg-${PRGNAM}.log
(cd $OUTPUT && md5sum ${PRGNAM}-${VERSION}_${KVER}-${ARCH}-${BUILD}.tgz >
${PRGNAM}-${VERSION}_${KVER}-${ARCH}-${BUILD}.tgz.md5)
cat $PKG/install/slack-desc | grep "^${PRGNAM}" > $OUTPUT/${PRGNAM}-
${VERSION}_${KVER}-${ARCH}-${BUILD}.txt

# --- CLEANUP ---

# Clean up the extra stuff:
if [ "$P1" = "--cleanup" ]; then
    rm -rf $TMP/tmp-${PRGNAM}
    rm -rf $PKG
fi

```


ANEXO 3

Script rc.drbd para iniciar/parar o DRBD

```
#!/bin/bash
#
# chkconfig: 345 70 8
# description: Loads and unloads the drbd module
#
# Copyright 2001-2008 LINBIT Information Technologies
# Philipp Reisner, Lars Ellenberg
#
### BEGIN INIT INFO
# Provides: drbd
# Required-Start: $network $syslog sshd
# Required-Stop: $network $syslog sshd
# Default-Start: 3 5
# Default-Stop: 0 1 2 6
# Short-Description: Control drbd resources.
### END INIT INFO

DEFAULTFILE="/etc/drbd"
DRBDADM="/sbin/drbdadm"
PROC_DRBD="/proc/drbd"
MODPROBE="/sbin/modprobe"
RMMOD="/sbin/rmmod"
UDEV_TIMEOUT=10
ADD_MOD_PARAM=""

if [ -f $DEFAULTFILE ]; then
. $DEFAULTFILE
fi

test -f $DRBDADM || exit 5

function assure_module_is_loaded
{
    [ -e "$PROC_DRBD" ] && return

    $MODPROBE -s drbd ` $DRBDADM sh-mod-params ` $ADD_MOD_PARAM || {
        echo "Can not load the drbd module."$'\n'; exit 20
    }
    # tell klogd to reload module symbol information ...
    [ -e /var/run/klogd.pid ] && [ -x /sbin/klogd ] && /sbin/klogd -i
}

function adjust_with_progress
{
    IFS_O=$IFS
    NEWLINE='
'
    IFS=$NEWLINE
    local res

    COMMANDS=`$DRBDADM -d -n res adjust all` || exit 20
    echo -n "[ "

    for CMD in $COMMANDS; do
        case "$CMD" in
            res=*)          eval "$CMD";;
            *\ disk\ *)     echo -n "d($res) " ;;
            *\ syncer\ *)   echo -n "s($res) " ;;
            *\ net\ *)      echo -n "n($res) " ;;
            *)              echo ".. " ;;
        esac
        if ! eval "$CMD"; then

```

```

        echo -e "\n[$res] cmd $CMD failed - continuing!\n "
    fi
done
echo -n "]"

IFS=$IFS_O
}

drbd_pretty_status()
{
    local proc_drbd=$1
    # add resource names
    if ! type column &> /dev/null ||
        ! type paste &> /dev/null ||
        ! type join &> /dev/null ||
        ! type sed &> /dev/null ||
        ! type tr &> /dev/null
    then
        cat "$proc_drbd"
        return
    fi
    sed -e '2q' < "$proc_drbd"
    sed_script=$(
        paste <(drbdadm sh-dev all | cat -n) \
              <(drbdadm sh-resources | tr ' ' '\n') |
        sed -e 's/|/_/g' \
            -e 's#^\( *[0-9]\|+.\)/dev/drbd\([0-9]\|+\)\. \(.*\)$#s|^ *2:|\1&\3|#')

    p() {
        sed -e "1,2d" \
            -e "$sed_script" \
            -e '/^ *[0-9]\|+: cs:Unconfigured/d;' \
            -e 's/^\(.* cs:.*\([^\ ]\)\) \([rs]...\)\$/\1 - \2/g' \
            -e 's/^\(.* \)cs:\([^\ ]*\)st:\([^\ ]*\)ds:\([^\ ]*\)/\1\2\3\4/'

        -e 's/^\(.* \)cs:\([^\ ]*\)\$/\1\2/' \
        -e 's/^ *[0-9]\|+:/ x &??not-found?;/' \
        -e '/^$/d;/ns:.*nr:.*dw:/d;/resync:/d;/act_log:/d;' \
        -e 's/^\(.\[.*\)\(sync.ed:\)/... \2;/^finish:/d;' \
        -e 's/^\(.[0-9 %]*oos:\)/... \1/' \
        < "$proc_drbd" | tr -s '\t ' ' '

    }

    m() {
        join -1 2 -2 1 -o 1.1,2.2,2.3 \
            <(drbdadm sh-dev all | cat -n | sort -k2,2) \
            <(sort < /proc/mounts ) |
            sort -n | tr -s '\t ' ' ' | sed -e 's/^ *// '

    }

    # echo "=== p ==="
    # p
    # echo "=== m ==="
    # m
    # echo "====="
    # join -al <(p|sort) <(m|sort)
    # echo "====="
    (
        echo m:res cs st ds p mounted fstype
        join -al <(p|sort) <(m|sort) | cut -d' ' -f2-6,8- | sort -k1,1n -k2,2
    ) | column -t
}

# Just in case drbdadm want to display any errors in the configuration
# file, or we need to ask the user about registering this installation
# at http://usage.drbd.org, we call drbdadm here without any IO
# redirection.
$DRBDADM sh-nop

case "$1" in

```

```

start)
    echo -n "Starting DRBD resources:      "
    assure_module_is_loaded
    adjust_with_progress

    # make sure udev has time to create the device files
    for RESOURCE in ` $DRBDADM sh-resources `; do
        for DEVICE in ` $DRBDADM sh-dev $RESOURCE `; do
            UDEV_TIMEOUT_LOCAL=$UDEV_TIMEOUT
            while [ ! -e $DEVICE ] && [ $UDEV_TIMEOUT_LOCAL -gt 0 ] ; do
                sleep 1
                UDEV_TIMEOUT_LOCAL=$(( $UDEV_TIMEOUT_LOCAL-1 ))
            done
        done
    done

    [ -d /var/lock/subsys ] && touch /var/lock/subsys/drbd      # for RedHat
    echo "."
    $DRBDADM wait-con-int # User interruptible version of wait-connect all

#    $DRBDADM sh-b-pri all # Become primary if configured
;;
stop)
    echo -n "Stopping all DRBD resources"
    if [ -e $PROC_DRBD ] ; then
        $DRBDADM down all
        $RMMOD drbd
    fi
    [ -f /var/lock/subsys/drbd ] && rm /var/lock/subsys/drbd
    echo "."
;;
status)
    # NEEDS to be heartbeat friendly...
    # so: put some "OK" in the output.
    if [ -e $PROC_DRBD ]; then
        echo "drbd driver loaded OK; device status:"
        drbd_pretty_status $PROC_DRBD 2>/dev/null
        exit 0
    else
        echo >&2 "drbd not loaded"
        exit 3
    fi
;;
reload)
    echo -n "Reloading DRBD configuration"
    $DRBDADM adjust all
    echo "."
;;
restart|force-reload)
    echo -n "Restarting all DRBD resources"
    $DRBDADM down all
    $RMMOD drbd
    assure_module_is_loaded
    $DRBDADM up all
    echo "."
;;
*)
    echo      "Usage:      /etc/init.d/drbd      {start|stop|status|reload|restart|force-
reload}"
    exit 1
;;
esac

exit 0

```

ANEXO 4

Script que o Heartbeat usa para ativar/parar o drbd

```
#!/bin/bash
#
DRBDADM="/sbin/drbdadm"
MODPROBE="/sbin/modprobe"
RMMOD="/sbin/rmmod"
MOUNT="/sbin/mount"
UMOUNT="/sbin/umount"

$MODPROBE -s drbd

case "$1" in
    start)
        echo -n "Starting all DRBD devices: "

        $DRBDADM up all
        $DRBDADM primary all
        $MOUNT /home

        echo "Done."

        ;;
    stop)
        echo -n "Stopping all DRBD devices: "

        $UMOUNT /dev/drbd0
        $DRBDADM secondary all

        echo "Done."

        ;;
    *)
        echo "Usage: $0 {start|stop}"
        exit 1

        ;;
esac

exit 0
```

ANEXO 5

Arquivo drb.conf do DRBD

```
global {
    usage-count no;
}
resource r0 {
    protocol B;
    startup {
        wfc-timeout 0;
        degr-wfc-timeout 120;    # 2 minutes.
    }
    disk {
        on-io-error detach;
    }
    syncer {
        rate 100M;
    }
    net {
        timeout 60;
        connect-int 10;
        ping-int 10;
        after-sb-0pri discard-zero-changes;
        after-sb-1pri discard-secondary;
        after-sb-2pri disconnect;
    }
    on dados {
        device    /dev/drbd0;
        disk      /dev/sda1;
        address    10.0.0.1:7788;
        meta-disk  internal;
    }
    on backup {
        device    /dev/drbd0;
        disk      /dev/sda1;
        address    10.0.0.2:7788;
        meta-disk  internal;
    }
}
```

ANEXO 6

Arquivo haresources do Heartbeat

```
dados IPaddr::192.168.0.212/24/eth1 drbd samba
```

ANEXO 7

Arquivo `ha.cf` do Heartbeat

```
debugfile /var/log/ha-debug
logfile    /var/log/ha-log
logfacility local0
keepalive 2
deadtime 30
warntime 10
initdead 120
udpport    694
bcast eth0
auto_failback off
node dados
node backup
```

ANEXO 8

Arquivo authkeys do Heartbeat

```
Auth 3
#1 crc
#2 sh1
3 md5 Hello!
```


ANEXO 9

Script rc.heartbeat para iniciar/parar o Heartbeat

```
#!/bin/sh
#
#   $Id: heartbeat.in,v 1.45 2006/08/08 16:03:49 davidlee Exp $
#
# heartbeat      Start high-availability services
#
# Author:        Alan Robertson      <alanr@unix.sh>
# License:       GNU General Public License (GPL)
#
#               This script works correctly under SuSE, Debian,
#               Conectiva, Red Hat and a few others. Please let me know if it
#               doesn't work under your distribution, and we'll fix it.
#               We don't hate anyone, and like for everyone to use
#               our software, no matter what OS or distribution you're using.
#
# chkconfig: 2345 75 05
# description: Startup script high-availability services.
# processname: heartbeat
# pidfile: /var/run/heartbeat.pid
# config: /etc/ha.d/ha.cf
#
### BEGIN INIT INFO
# Description: heartbeat is a basic high-availability subsystem.
#   It will start services at initialization, and when machines go up
#   or down. This version will also perform IP address takeover using
#   gratuitous ARPs. It works correctly for a 2-node configuration,
#   and is extensible to larger configurations.
#
#   It implements the following kinds of heartbeats:
#       - Bidirectional Serial Rings ("raw" serial ports)
#       - UDP/IP broadcast (ethernet, etc)
#       - UDP/IP multicast (ethernet, etc)
#       - Unicast heartbeats
#       - "ping" heartbeats (for routers, switches, etc.)
#       (to be used for breaking ties in 2-node systems
#       and monitoring networking availability)
#
# Short-Description: High-availability services.
# Required-Start: $network $time $syslog
# Required-Stop: $network $time $syslog
# Default-Start: 3 5
# Default-Stop: 0 6
### END INIT INFO

HA_DIR=/etc/ha.d; export HA_DIR
CONFIG=$HA_DIR/ha.cf
. $HA_DIR/shellfuncs

LOCKDIR=/var/lock/subsys
RUNDIR=/var/run

if
[ -r /etc/SuSE-release ]
then
# rc.status is new since SuSE 7.0
```

```

[ -r /etc/rc.status ] && . /etc/rc.status
[ -r /etc/rc.config ] && . /etc/rc.config

# Determine the base and follow a runlevel link name.
base=${0##*/}
link=${base#[SK][0-9][0-9]}

fi
if
[ -z "$rc_done" ]
then
    rc_done="Done."
    rc_failed="Failed."
    rc_skipped="Skipped."
fi

# exec 2>>/var/log/ha-debug

#     This should probably be it's own autoconf parameter
#     because RH has moved it from time to time...
#     and I suspect Conectiva and Mandrake also supply it.

DISTFUNCS=/etc/rc.d/init.d/functions
PROC_HA=$HA_BIN/ha.o
SUBSYS=heartbeat
MODPROBE=/sbin/modprobe
US=`uname -n`

# Set this to a 1 if you want to automatically load kernel modules
USE_MODULES=1

[ -x $HA_BIN/heartbeat ] || exit 0

#
#     Some environments like it if we use their functions...
#
if
[ ! -x $DISTFUNCS ]
then
    # Provide our own versions of these functions
    status() {
        $HA_BIN/heartbeat -s
    }
    echo_failure() {
        EchoEsc " Heartbeat failure [rc=$1]. $rc_failed"
        return $1
    }
    echo_success() {
        : Cool! It started!
        EchoEsc "$rc_done"
    }
else
    . $DISTFUNCS
fi

#
#     See if they've configured things yet...
#
if
[ ! -f $CONFIG ]
then

```

```

    EchoNoNl "Heartbeat not configured: $CONFIG not found."
    echo_failure 1
    exit 0
fi

CrmEnabled() {
    case `ha_parameter crm | tr '[A-Z]' '[a-z]'` in
        y|yes|enable|on|true|1|manual) true;;
        *) false;;
    esac
}

StartLogd() {
    $SHA_BIN/ha_logd -s >/dev/null 2>&1

    if
        [ $? -eq 0 ]
    then
        Echo "logd is already running"
        return 0
    fi

    $SHA_BIN/ha_logd -d >/dev/null 2>&1
    if
        [ $? -ne 0 ]
    then
        Echo "starting logd failed"
    fi
}

StopLogd() {
    $SHA_BIN/ha_logd -s >/dev/null 2>&1

    if
        [ $? -ne 0 ]
    then
        Echo "logd is already stopped"
        return 0
    fi

    $SHA_BIN/ha_logd -k >/dev/null 2>&1
    if
        [ $? -ne 0 ]
    then
        Echo "stopping logd failed"
    fi
}

init_watchdog() {
    if
        [ -f /proc/devices -a -x $MODPROBE ]
    then
        init_watchdog_linux
    fi
}

#
# Install the softdog module if we need to
#
init_watchdog_linux() {

```

```

#
# We need to install it if watchdog is specified in $CONFIG, and
# /dev/watchdog refers to a softdog device, or it /dev/watchdog
# doesn't exist at all.
#
# If we need /dev/watchdog, then we'll make it if necessary.
#
# Whatever the user says we should use for watchdog device, that's
# what we'll check for, use and create if necessary. If they misspell
# it, or don't put it under /dev, so will we.
# Hope they do it right :-)
#
#
insmod=no
# What do they think /dev/watchdog is named?
MISCDEV=`grep ' misc$' /proc/devices | cut -c1-4`
MISCDEV=`Echo $MISCDEV`
WATCHDEV=`ha_parameter watchdog`
WATCHDEV=`Echo $WATCHDEV`
if
[ "X$WATCHDEV" != X ]
then
: Watchdog requested by $CONFIG file
#
# We try and modprobe the module if there's no dev or the dev exists
# and points to the softdog major device.
#
if
[ ! -c "$WATCHDEV" ]
then
insmod=yes
else
case `ls -l "$WATCHDEV" 2>/dev/null` in
*$MISCDEV,*)
insmod=yes;;
*)
: "$WATCHDEV isn't a softdog device (wrong major)" ;;
esac
fi
else
: No watchdog device specified in $CONFIG file.
fi
case $insmod in
yes)
if
grep softdog /proc/modules >/dev/null 2>&1
then
: softdog already loaded
else
$MODPROBE softdog nowayout=0 >/dev/null 2>&1
fi;;
esac
if
[ "X$WATCHDEV" != X -a ! -c "$WATCHDEV" -a $insmod = yes ]
then
minor=`cat /proc/misc | grep watchdog | cut -c1-4`
mknod -m 600 $WATCHDEV c $MISCDEV $minor
fi
} # init_watchdog_linux()

#
# Start the heartbeat daemon...

```

```

#

start_heartbeat() {
    if
        ERROR=`$HA_BIN/heartbeat 2>&1`
    then
        : OK
    else
        return $?
    fi
}

#
#     Start Linux-HA
#

StartHA() {
    EchoNoNl "Starting High-Availability services: "

    if
        CrmEnabled
    then
        : OK
    else
        $HA_BIN/ResourceManager verifyallidle
    fi
    if
        [ $USE_MODULES = 1 ]
    then
        # Create /dev/watchdog and load module if we should
        init_watchdog
    fi
    rm -f $RUNDIR/ppp.d/*
    if
        [ -f $HA_DIR/ipresources -a ! -f $HA_DIR/haresources ]
    then
        mv $HA_DIR/ipresources $HA_DIR/haresources
    fi
    #     Start heartbeat daemon
    if
        start_heartbeat
    then
        echo_success
        return 0
    else
        RC=$?
        echo_failure $RC
        if [ ! -z "$ERROR" ]; then
            Echo
            Echo "$ERROR"
        fi
        return $RC
    fi
}

#
#     Ask heartbeat to stop.  It will give up its resources...
#

StopHA() {
    EchoNoNl "Stopping High-Availability services: "

```

```

if
    $HA_BIN/heartbeat -k &> /dev/null      # Kill it
then
    echo_success
    return 0
else
    RC=$?
    echo_failure $RC
    return $RC
fi
}

StatusHA() {
    $HA_BIN/heartbeat -s
}

StandbyHA() {
    auto_failback=`ha_parameter auto_failback | tr ' [A-Z]' '[a-z]'`
    nice_failback=`ha_parameter nice_failback | tr ' [A-Z]' '[a-z]'`

    case "$auto_failback" in
        *legacy*)      echo "auto_failback is set to legacy.  Cannot enter
standby."
                        exit 1;;
    esac
    case "$nice_failback" in
        *off*)      echo "nice_failback is disabled.  Cannot enter standby."
                        exit 1;;
    esac
    case "${auto_failback}${nice_failback}" in
        "")      echo "auto_failback defaulted to legacy.  Cannot enter
standby."
                        exit 1;;
    esac

    echo "auto_failback: $auto_failback"
    if
        StatusHA >/dev/null 2>&1
    then
        EchoNoNl "Attempting to enter standby mode"
        if
            $HA_BIN/hb_standby
        then
            # It's impossible to tell how long this will take.
            echo_success
        else
            echo_failure $?
        fi
    else
        Echo "Heartbeat is not currently running."
        exit 1
    fi
}

#
#     Ask heartbeat to restart.  It will *keep* its resources
#
ReloadHA() {
    EchoNoNl "Reloading High-Availability services: "

    if
        $HA_BIN/heartbeat -r # Restart, and keep your resources

```

```

    then
        echo_success
        return 0
    else
        RC=$?
        echo_failure $RC
        return $RC
    fi
}

RunStartStop() {
    # Run pre-startup script if it exists
    if
        [ -f $HA_DIR/resource.d/startstop ]
    then
        $HA_DIR/resource.d/startstop "$@"
    fi
}

RC=0
# See how we were called.

case "$1" in
    start)
        StartLogd
        RunStartStop pre-start
        StartHA
        RC=$?
        Echo
        if
            [ $RC -eq 0 ]
        then
            [ ! -d $LOCKDIR ] && mkdir -p $LOCKDIR
            touch $LOCKDIR/$SUBSYS
        fi
        RunStartStop post-start $RC
        ;;

    standby)
        StandbyHA
        RC=$?;;

    status)
        StatusHA
        RC=$?;;

    stop)
        RunStartStop "pre-stop"
        StopHA
        RC=$?
        Echo
        if
            [ $RC -eq 0 ]
        then
            rm -f $LOCKDIR/$SUBSYS
        fi
        RunStartStop post-stop $RC
        StopLogd
        ;;

    restart)
        sleeptime=`ha_parameter deadtime`

```

```

    StopHA
    Echo
    EchoNoNl Waiting to allow resource takeover to complete:
    sleep $sleep_time
    sleep 10 # allow resource takeover to complete (hopefully).
    echo_success
    Echo
    StartHA
    Echo
    ;;

force-reload|reload)
    ReloadHA
    Echo
    RC=$?
    ;;

*)
    Echo "Usage: $0 {start|stop|status|restart|reload|force-reload}"
    exit 1
esac

exit $RC

```


ANEXO 10

Script para ativar o samba através do Heartbeat

```
#!/bin/sh
#
# "$Id: samba.sh,v 1.1.2.1 2001/07/03 01:01:09 jra Exp $"
#
# SAMBA startup (init) script for LSB-compliant systems.
#
# Provides: smbd nmbd
# Required-Start: 3 5
# Required-Stop: 0 2 1 6
# Default-Start: 3 5
# Default-Stop: 0 2 1 6
# Description: Starts and stops the SAMBA smbd and nmbd daemons \
#              used to provide SMB network services.
#

# Source LSB function library.
. /etc/init.d/functions

# Check that smb.conf exists.
if test ! -f /etc/samba/smb.conf; then
    log_failure_msg "The smb.conf file does not exist."
    exit 6
fi

# Make sure that smbd and nmbd exist...
if test ! -f /usr/sbin/nmbd -o ! -f /usr/sbin/smbd; then
    log_failure_msg "The nmbd and/or smbd daemons are not installed."
    exit 5
fi

# See how we were called.
case "$1" in
    start)
        daemon nmbd -D
        daemon smbd -D
        echo "Started SMB services."
        ;;

    stop)
        killproc smbd
        killproc nmbd
        echo "Shutdown SMB services."
        ;;

    reload)
        # smbd and nmbd automatically re-read the smb.conf file...
        echo "Reload not necessary with SAMBA."
        ;;

    status)
        if test -z "`pidofproc smbd`"; then
            echo "smbd is not running."
        else
            echo "smbd is running."
        fi
        if test -z "`pidofproc nmbd`"; then
            echo "nmbd is not running."
        fi
    esac

```

```

        else
            echo "nmbd is running."
        fi
    ;;

restart | force-reload)
    $0 stop
    $0 start
    ;;

*)
    echo "Usage: smb {start|stop|reload|force-
reload|restart|status}"
    exit 1
    ;;
esac

# Return "success"
exit 0

#
# End of "$Id: samba.sh,v 1.1.2.1 2001/07/03 01:01:09 jra Exp $".
#

```

ANEXO 11

Mini How To de instalação e configuração dos softwares Heartbeat e DRBD no Slackware Linux 12.1.

Como não existe pacote pré compilado para a distribuição Slackware 12.1, os arquivos fontes do Heartbeat e DRBD serão baixados e compilados.

COMPILAÇÃO E INSTALAÇÃO DO HEARTBEAT

O Heartbeat necessita de uma biblioteca chamada libnet para poder ser compilado.

Crie um subdiretório para o libnet:

```
mkdir /usr/local/src/libnet
```

Acesse o subdiretório:

```
cd /usr/local/src/libnet
```

Baixe o arquivo fonte e os arquivos necessários para a compilação:

```
wget
```

```
http://www.slackware.com/~alien/slackbuilds/libnet/build/libnet-1.1.2.1.tar.gz
```

```
wget
```

```
http://www.slackware.com/~alien/slackbuilds/libnet/build/libnet.SlackBuild
```

```
wget http://www.slackware.com/~alien/slackbuilds/libnet/build/slack-desc
```

Dê permissão de execução:

```
chmod +x libnet.Slackbuild
```

Inicie a compilação:

```
./libnet.Slackbuild
```

Instale a biblioteca:

```
installpkg /tmp/libnet-1.1.2.1-i486-1.tgz
```

Crie um subdiretório para o Heartbeat:

```
mkdir /usr/local/src/heartbeat
```

Acesse o subdiretório:

```
cd /usr/local/src/heartbeat
```

Baixe o arquivo fonte e os arquivos necessários para a compilação:

```
wget
```

```
http://www.slackware.com/~alien/slackbuilds/heartbeat/build/heartbeat-2.0.8.tar.gz
```

```
wget
http://www.slackware.com/~alien/slackbuilds/heartbeat/build/heartbea
t.SlackBuild
wget
http://www.slackware.com/~alien/slackbuilds/heartbeat/build/slack-
desc
```

Dê permissão de execução:

```
chmod +x heartbeat.SlackBuild
```

Inicie a compilação:

```
./heartbeat.SlackBuild
```

Instale o Heartbeat:

```
installpkg /tmp/heartbeat-2.0.8-i486-1.tgz
```

COMPILAÇÃO E INSTALAÇÃO DO DRBD

O DRBD necessita dos fontes do kernel do Linux para poder ser compilado.

Instale o kernel source:

```
slapt-get --install kernel-source
```

Crie o subdiretório para o DRBD:

```
mkdir /usr/local/src/drbd
```

Acesse o subdiretório:

```
cd /usr/local/src/drbd
```

Baixe o arquivo fonte do drbd:

```
wget http://oss.linbit.com/drbd/8.2/drbd-8.2.6.tar.gz
```

Baixe os arquivos necessários para a compilação:

```
wget
http://www.slackware.com/~alien/slackbuilds/drbd/build/drbd.SlackBui
ld
wget http://www.slackware.com/~alien/slackbuilds/drbd/build/slack-
desc
```

Altere o arquivo drbd.SlackBuild para compilar a versão correspondente:

```
VERSION=${VERSION:-8.2.6}
```

Dê permissão de execução:

```
chmod +x drbd.SlackBuild
```

Inicie a compilação:

```
./drbd.SlackBuild
```

Instale o drbd:

```
installpkg /tmp/drbd-8.2.6_2.6.24.5-smp-i486-1.tgz
```

CONFIGURAÇÃO DO DRBD

Configure o arquivo:

```
/etc/drbd.conf9
```

Dê permissão de execução para o script de ativação do drbd:

```
chmod +x /etc/rc.d/rc.drbd
```

Acrescente as linhas abaixo no arquivo rc.local para que o drbd seja ativado na inicialização do sistema:

```
#DRBD
if [ -x /etc/rc.d/rc.drbd ]; then
    /etc/rc.d/rc.drbd start
fi
```

CONFIGURAÇÃO DO HEARTBEAT

Configure os arquivos:

```
/etc/ha.d/ha.cf10
```

```
/etc/ha.d/haresources11
```

```
/etc/ha.d/authkeys12
```

Crie o arquivo drbd:

```
touch /etc/ha.d/resource.d/drbd13
```

Crie o arquivo samba:

```
touch /etc/ha.d/resource.d/samba14
```

Dê permissão de execução para o script de inicialização do Heartbeat:

```
chmod +x /etc/rc.d/rc.heartbeat
```

Configure o sistema para iniciar o script automaticamente na inicialização do sistema.

Adicione as seguintes linhas no arquivo rc.local:

```
#Heartbeat
if [ -x /etc/rc.d/rc.heartbeat ]; then
    /etc/rc.d/rc.heartbeat start
fi
```

⁹ Anexo 5

¹⁰ Anexo 7

¹¹ Anexo 6

¹² Anexo 8

¹³ Anexo 4

¹⁴ Anexo 10