SISTEMA DE TOLERÂNCIA A FALHAS ATRAVÉS DE REDUNDÂNCIA DE SOFTWARE E HARDWARE

Aline Souza Rocha[1]
Eliane Raimann[2]
Heverton Barros de Macêdo[3]

[1]IFG/Campus Jataí/Engenharia Elétrica – PIBIC/CNPq, aline_souzaroch@yahoo.com.br
[2]IFG/ Campus Jataí / Departamento I, elianeraimann@gmail.com
[3]IFG/ Campus Jataí /Departamento I, hevertonbarros@gmail.com

Resumo

O termo "tolerância a falhas", que antes era empregado exclusivamente a sistemas de funcionamento crítico, está sendo amplamente utilizado nos mais diversos sistemas computacionais, devido à maior garantia de disponibilidade que este recurso fornece às redes e aos sistemas distribuídos. A necessidade de manter a rede de computadores dos laboratórios de informática do Instituto Federal de Goiás, Campus Jataí, em constante funcionamento, fez com que o tema "tolerância a falhas" seja objeto de investigação na construção de um sistema capaz de contornar falhas que eventualmente possam ocorrer. Estas falhas podem ser ocasionadas, dentre várias causas, por um ataque de vírus, travamento do sistema operacional devido a um software mal projetado, rompimento ou interferência de um cabo da rede ou uma fonte de alimentação danificada. Para alcançar este objetivo, foi realizado um estudo sobre os sistemas mais utilizados no mercado, com o propósito de oferecer alta disponibilidade, observando suas principais características e funcionamento. Também foram realizadas investigações detalhadas a respeito da configuração do sistema operacional Linux e, em especial, da distribuição *Slackware*, atualmente utilizada na máquina servidora dos laboratórios da instituição. No decorrer da pesquisa foi desenvolvido um software, baseado em *Shell Script*, capaz de detectar a indisponibilidade da máquina servidora. Em seu funcionamento é realizada a replicação de dados em duas máquinas atuando como servidoras, em que detectada uma falha na máquina mestre, a máquina escrava assume o papel de mestre, de modo transparente para o usuário, mantendo assim, qualquer serviço que estiver em funcionamento no momento da falha. Tal sistema foi implantado e testado nos laboratórios da instituição tornando os serviços altamente disponíveis para a comunidade acadêmica, obedecendo, ao mesmo tempo, as particularidades e as necessidades destes laboratórios.

Palavras-chave: Tolerância a falhas. Alta disponibilidade. Confiabilidade. Redundância.

INTRODUÇÃO

A utilização de recursos computacionais nos mais variados setores, provendo diversos tipos de serviços, se tornou rotineira, fazendo com que a população seja cada vez mais dependente de suas facilidades. Atividades que antes exigiam esforço mental e por vezes físico podem atualmente ser desempenhadas computacionalmente, fato que permitiu a economia de tempo e dinheiro para a realização de outras tarefas. No entanto, é comum que tais sistemas possam apresentar falhas que comprometam a atividade a ser realizada. Por este motivo a aplicação de sistemas tolerantes a falhas tornou-se imprescindível, tanto para um sistema de

controle aeroespacial quanto para a mais simples rede de computadores instalada em uma pequena empresa.

Tolerância a falhas é um termo utilizado para alcançar "dependabilidade" (*dependability*), que por sua vez engloba confiabilidade, disponibilidade, segurança de funcionamento (*safety*), segurança (*security*), mantenabilidade, testabilidade e comprometimento do desempenho (*performability*), embora não haja um consenso na utilização do termo pela comunidade acadêmica (WEBER, 2002, pag. 7-10). Na área da computação é comum encontrarmos "alta disponibilidade" no lugar de "tolerância a falhas". Estes termos designam sistemas que possuem dispositivos que atuam como "reserva" caso o principal falhe. Os dispositivos podem ser físicos (hardware) ou lógicos (software), e, quando há a aplicação de ambos, o sistema pode se tornar mais seguro e confiável. A tolerância a falhas, como um termo geral, já teve seu uso restrito aos sistemas de missão crítica, mas atualmente é comum encontrarmos tolerância a falhas em diversos tipos de sistemas.

Dependendo dos recursos a serem utilizados, um projeto de tolerância a falhas pode representar um alto custo na aquisição de produtos (hardware e software) e implantação do sistema, no entanto, é possível construir soluções simples capazes de alcançar resultados satisfatórios dependendo do grau de disponibilidade e confiabilidade requerido.

No mercado estão disponíveis sistemas proprietários, que fornecem ao cliente a garantia de um sistema robusto e confiável, como o *Sun Enterprise Cluster* e o *Microsoft Cluster Server* (MSCS). Em contrapartida, a comunidade *open-source* oferece sistemas com propósitos similares, sem custo algum pela sua aquisição. É o caso de sistemas como o *Heartbeat*, *Common Address Redundancy Protocol* (CARP), *Linux Virtual Server* (LVS) e o *High Availability Open Source Aplication Resources* (HA-OSCAR). Embora existam várias implementações para sistemas tolerantes a falhas disponíveis no mercado, será dada maior ênfase ao *Heartbeat*, por ser de fácil acesso e possuir características mais próximas ao sistema proposto neste trabalho, além de ser bastante utilizado em *clusters*[1].

Neste contexto, foram estudados métodos para propor a construção de um sistema de alta disponibilidade direcionado aos laboratórios do Instituto Federal de Goiás, Campus Jataí. Atendendo às particularidades dos laboratórios, foi elaborado um sistema com o propósito de suprir as necessidades de operação contínua do servidor, além de possibilitar a recuperação de dados através de redundância. A linguagem usada para programação foi o *Shell Script*, considerada por muitos projetistas como uma importante ferramenta para a elaboração de tarefas complexas, sendo mais eficiente que várias ferramentas análogas a ela. A distribuição do *Linux* em que este sistema foi inserido e testado é o *Slackware*, considerada rápida, estável e que ainda permite a personalização de seus recursos de acordo com a necessidade da rede.

Basicamente, o sistema elaborado foi instalado em duas máquinas, que foram denominadas "máquina mestre", o servidor principal, e "máquina escrava", o servidor secundário. Vale destacar que o conceito de mestre e escravo pode ser invertido entre as máquinas, ou seja, uma máquina que atualmente é escrava passa a ser a máquina mestre, caso seja identificada alguma indisponibilidade na máquina que atuava anteriormente como mestre.

O sistema consiste em dois scripts distintos, mas que possuem dependência entre si. Este conjunto de scripts é capaz de realizar uma replicação de todos os dados gravados na

[1] Conjunto de computadores conectados em sistema distribuído e que funcionam como uma única máquina de grande porte.

máquina mestre para a máquina escrava, e esta replicação é atualizada conforme a necessidade do administrador da rede, através de um arquivo de configuração. O envio de pacotes da máquina escrava para a máquina mestre, com o intuito de verificar sua disponibilidade, também é programado de acordo com o intervalo de tempo pré-configurado pelo administrador da rede, e a resposta a estes pacotes em sentido contrário é esperada. Caso a máquina mestre seja impossibilitada de enviar a resposta, a máquina escrava interpreta esta ausência como uma falha por parte da mestre e, automaticamente, todas as solicitações de serviços passarão a ser respondidos pela máquina escrava, garantindo, desta maneira, a alta disponibilidade desejada. É importante ressaltar que este processo é feito de forma transparente para o usuário, ou seja, para o usuário não há a percepção da troca de servidores. Esta é uma característica que proporciona acessibilidade contínua aos serviços oferecidos pelo servidor.

Com embasamento em resultados de testes realizados durante o funcionamento do sistema tolerante a falhas, observa-se êxito na obtenção de um sistema que disponha à comunidade acadêmica acessibilidade contínua e robustez sem retirar as particularidades do sistema de rede do campus de Jataí.

ALTA DISPONIBILIDADE

Segundo Weber (2002) o termo “tolerância a falhas”, usado para designar toda área de pesquisa que se ocupa com o comportamento de sistemas computacionais sujeitos à ocorrência de falhas, não obteve a popularização desejada por seu autor, Avizienis, que a criou em 1967. Os mais variados campos de aplicação destes sistemas cunharam denominações distintas para este termo. Para os desenvolvedores dos sistemas de controle industriais, o termo “sistemas redundantes” é mais bem aplicado. Para a comunidade de desenvolvedores de sistemas computacionais, o termo “alta disponibilidade” é o que melhor se aplica ao serviço que este tipo de sistema oferece aos servidores de redes, e designa sua principal qualidade.

Para se compreender exatamente qual o contexto em que se está inserida a tolerância a falhas, é necessário conhecer alguns termos que fazem parte deste universo computacional e que, na maioria das vezes, são tratados erroneamente por alguns autores. De acordo com Sztoltz, Teixeira e Ribeiro (2003), falha, erro e defeito possuem significados distintos:

> *Recapitulando, 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. (SZTOLTZ; TEIXEIRA; RIBEIRO, 2003).*

Inúmeras técnicas são empregadas por projetistas e administradores de rede para alcançar alta disponibilidade. O objetivo é obter maiores recursos tecnológicos e novas funcionalidades aos seus sistemas. A principal técnica utilizada é a redundância de hardware e de software.

Para Ferreira e Santos (2005), o termo redundância significa a existência de vários métodos para efetuar uma determinada função ou de algum equipamento alternativo que mesmo na presença de falha pode assegurar automaticamente o acesso aos serviços prestados. Na maioria dos casos, a redundância é empregada simultaneamente em hardware e software, garantido uma melhor funcionalidade do sistema, principalmente em casos de missão crítica.

Sob o ponto de vista de Morimoto (2009), redundância implica em possuir componentes "reserva" em caso de falha do principal. A redundância pode ser aplicada em fontes, *arrays* de discos e mesmo em servidores completos (*clusters*), sincronizados em tempo real, em que há o monitoramento de um deles e a substituição do principal pelo redundante em caso de falha.

A tolerância a falhas, ou alta disponibilidade, já teve seu uso restrito justamente aos sistemas de missão crítica, como em aviões, sondas espaciais, bancos e sistemas de controle industriais. Com a maior dependência dos computadores para a realização de tarefas repetitivas, cansativas ou mesmo monótonas, houve uma crescente popularização destes sistemas para a maioria da prestação de serviços, segundo Weber (2002). Esta dependência proporcionou aos projetistas e administradores de rede a chance de desenvolver softwares que proporcionassem uma maior garantia de acessibilidade e segurança aos seus clientes.

Já a popularização dos sistemas computacionais permitiu um desenvolvimento positivo da tolerância a falhas. Atualmente existem no mercado inúmeros sistemas tolerantes a falhas, tanto prioritários quanto os *open-source* (livres). Nas soluções *open-source* se destaca o projeto *Linux HA*, que desenvolve o *Heartbeat*, bastante utilizado por possuir um código aberto e de livre aquisição.

Segundo Ferreira e Santos (2005), o *Heartbeat* funciona através de um método denominado *failover*, no qual ocorre uma comutação dos pedidos enviados do servidor principal para o servidor secundário, caso ocorra alguma falha ou a indisponibilidade temporária por parte do servidor principal. As autoras destacam ainda que esta seja uma importante solução para sistemas de missão crítica e que necessitam de fornecer uma acessibilidade contínua.

De acordo com Morimoto (2009), cada servidor possui pelo menos duas placas de rede, para que eles sejam simultaneamente conectados à rede e entre si através de um cabo *cross-over* (cabo cruzado) ou de um *switch* dedicado. A conexão interna é realizada pelo *Heartbeat* para realizar processos de monitoramento e sincronismo e realizar a substituição de servidores quando houver a necessidade. É utilizado em conjunto com o *Heartbeat*, o DRBD (*Distributed Replicated Block Device*), um módulo capaz de manter os HDs dos dois servidores sincronizados em tempo real.

A alta disponibilidade pode ser "medida" pelo número de "noves". Um nove indica disponibilidade de 90%, enquanto que dois noves indicam disponibilidade de 99% e assim em diante. Segundo Morimoto (2009), uma disponibilidade de 99% é comum a um servidor sem dispositivos redundantes, significando que há uma tolerância de até 7 horas e 18 minutos por em que este servidor pode permanecer fora do ar, incluindo todas as atividades de manutenção planejada, quedas de energia, entre outros fatores. O alcance de módulos redundantes permite atingir marcas de disponibilidade bem maiores. O alcance de três noves é bastante importante, e, segundo o autor, já é possível, neste caso, incluir dispositivos como as fontes redundantes ou mesmo os *clusters* de alta disponibilidade. Isto significa que ao servidor não é permitido que permaneça mais de 4 minutos e meio por mês fora do ar.

No entanto, é possível possuir um sistema tolerante a falhas com alta disponibilidade com pouco de investimento financeiro ou mesmo nenhum. O uso do software livre permite a implementação de sistemas para qualquer que seja a finalidade. Outra questão é o uso de máquinas para a redundância do servidor. Dependendo da aplicação não há a necessidade de se adquirir equipamentos mais eficientes para a montagem de um *cluster*. O reaproveitamento de máquinas que estão em desuso, ou a compra de computadores que possuam o mínimo de processamento e memória permitem ao menos uma disponibilidade de 99,9%.

A ideia do sistema desenvolvido neste trabalho é aplicar os conceitos de tolerância a falha fazendo uso de uma máquina a mais, de preferência que esteja subutilizada, através do desenvolvimento de um software responsável por identificar a indisponibilidade da máquina servidora. O desenvolvimento do software foi realizado utilizando a linguagem *Shell Script*.

SHELL SCRIPT

"Shell é o nome genérico de uma classe de programas que funciona como interpretador de comandos e linguagem de programação script (interpretada) no Unix." (FERREIRA, 2008, p. 112). Ainda segundo o autor, o *Shell*, além de uma interface entre o *kernel*[2] e o usuário, trata-se de uma linguagem de programação poderosa e interpretada e que possui estruturas de controle de alto nível, capaz de resolver problemas sem ter que recorrer às linguagens de programação compiladas, como a linguagem C.

Presente em qualquer distribuição *Linux*, o *Shell* está disponível para interpretação e gerenciamento de comandos. De acordo com Negus (2008), apesar de o *Shell* ser menos intuitivo que as interfaces gráficas de usuário, as denominadas GUIs, muitos especialistas o consideram bem mais eficiente que elas.

> *Os programas que consistem em comandos que são armazenados e executados a partir de um arquivo são conhecidos como: shell scripts. Muitos administradores de sistema Linux usam o shell scripts para automatizar tarefas, como, por exemplo, recuperar dados, monitorar log de dados ou verificar a saúde do sistema. (NEGUS, 2008, p. 32)*

No contexto do presente trabalho, será dada ênfase a dois comandos importantes do *Shell*, que possibilitaram a formação dos scripts para monitoramento da máquina principal e replicação de dados: *ping* e *rsync*.

Segundo Schroder (2009), uma das primeiras tentativas para solucionar um problema presente em uma rede de computadores é testar a conectividade entre as estações cliente e o servidor, ou mesmo entre servidores. O uso do comando *ping* permite que este teste seja efetuado com êxito.

A sintaxe para este comando é *ping [opções] [destino]*. Para testar a conectividade, no campo *[destino]* é necessário inserir o endereço IP de uma máquina remota. O comando deverá retornar, no caso de confirmação da conectividade, algo que se assemelhe à seguinte resposta: *64 bytes from 192.168.0.x: icmp_seq=1 ttl=64 time=5.49 ms*. Algumas estatísticas são demonstradas no final da resposta ao comando, informando, por exemplo, o número de pacotes enviados à máquina de destino e o número de pacotes recebidos por ela.

O comando *rsync* demonstra bastante utilidade para a realização de backups. Para Morimoto (2009), realizar backups em intervalos de tempo regulares é imprescindível na administração de qualquer servidor, por mais confiável que seja a máquina. O autor destaca que o comando *rsync* se torna uma ferramenta eficiente para este tipo de serviço, pois permite sincronizar o conteúdo de dois diretórios, transferindo apenas arquivos que sofreram modificações.

> *O comando rsync possui uma grande vantagem para os administradores, pois ele copia apenas o que mudou na árvore de diretórios. Por exemplo, se um arquivo foi*

[2] Núcleo ou cerne de um computador. Ponte entre os aplicativos e o processamento real de dados em nível de hardware.

> *modificado, ele irá transferir apenas os blocos novos ou alterados, assim, o tempo envolvendo ações com o rsync é sempre pequeno. (BONAN, 2010, p. 278).*

Outra vantagem deste comando é o tempo, citado pelo autor Bonan (2010), que é sempre pequeno. Já Morimoto (2009), sugere que podem ser realizados backups com *rsync* em intervalos de tempo menores, e incrementá-los com backups mais completos em intervalos de tempo maiores. A sintaxe para este comando é *rsync [opções] [origem] [destino]*.

SISTEMA PROPOSTO

No contexto da alta disponibilidade, e com base em conceitos e comandos relacionados ao terminal do *Linux*, foi implementado um sistema que possui requisitos para a tolerância a falhas e alta disponibilidade. O sistema foi instalado nos laboratórios de informática do Instituto Federal de Goiás, Campus Jataí, e, em comparação com o projeto *Linux-HA*, *Heartbeat*, o sistema foi elaborado respeitando certas particularidades e necessidades do laboratório citado.

O sistema consiste em dois scripts distintos que, entretanto, possuem extrema dependência entre si. Um arquivo de configuração também foi elaborado, a fim de minimizar esforços em caso de necessidade de mudança em alguma configuração do sistema elaborado.

O sistema foi projetado de forma que os mesmos scripts são usados nas duas máquinas (mestre e escrava), juntamente com um arquivo de configuração que possuem dados particulares para cada máquina. Os dados do arquivo de configuração possuem as seguintes informações:

a) Tempo em que a máquina escrava verifica se a máquina mestre está operando;
b) IP da máquina mestre;
c) IP da própria máquina;
d) Placa de rede a ser usada no monitoramento (eth0, eth1 ou wlan0);
e) Tempo necessário para efetuar o backup dos dados;
f) Diretórios a serem utilizados no backup dos dados.

O funcionamento dos scripts desenvolvidos está representado pelos diagramas das Figuras 1 e 2, que serão examinados a seguir.

Script Principal

Na figura 1, está representada a esquematização do script denominado “principal”. De acordo com o diagrama a primeira operação a ser realizada é a configuração de um endereço IP alternativo para esta máquina, sendo que este endereço havia sido previamente reservado a estas atividades pelo administrador da rede e informado no arquivo de configuração do sistema. O IP alternativo é configurado no momento em que este script entra em execução independente da máquina em questão ser ou não a máquina mestre. Ou seja, no início da execução do script a máquina em questão é considerada uma máquina escrava e irá verificar se já existe uma máquina mestre (IP principal) operando.

Logo após a configuração do endereço IP alternativo, é realizada uma verificação da atividade da placa de rede através do comando *mii-tool*. Se a resposta a este comando for *“no link”*, significa que algum cabo de rede está desconectado ou que a placa de rede sofreu algum tipo de falha. Então o sistema aguarda alguns segundos para realizar novamente a verificação de conectividade.

Entretanto, se a resposta ao comando for *“link ok”*, o sistema fará a execução do script de escuta sobre o IP principal, endereço da máquina mestre.

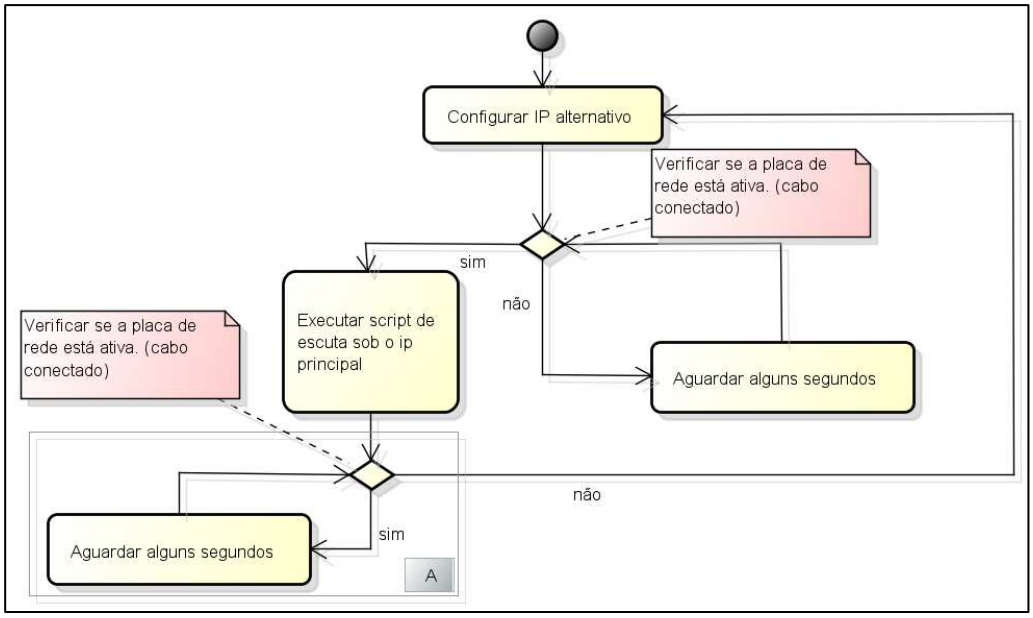

Figura 1: Esquematização do script principal (Fonte: Autoria própria).

Se a máquina mestre (IP principal) não está operando é esperado que a máquina escrava assuma o papel da máquina mestre. Em seguida, a máquina mestre ficará aguardando até que ocorra algum problema em sua conexão de rede. Possibilitando assim, o reinício do script ao seu estado inicial para a configuração de um endereço IP alternativo.

De fato, é esperado que a máquina principal fique em operação na maior parte do tempo, fazendo com que o script principal fique em *loop* na região marcada com o caractere A na Figura 1.

Script de Escuta

Após a primeira verificação da placa de rede no script principal, o script de escuta é executado, e seu diagrama está representado na Figura 2. A primeira operação realizada no momento da execução deste script é a verificação do funcionamento da máquina mestre (IP principal).

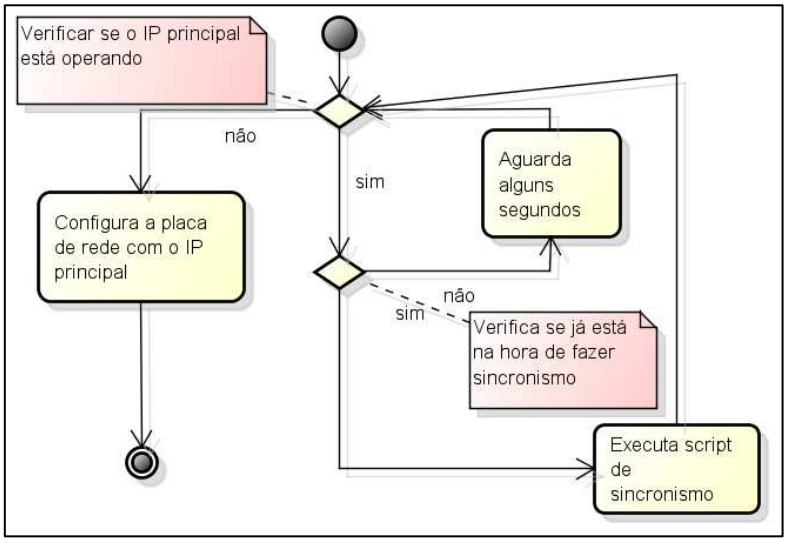

Figura 2: Esquematização do script de escuta (Fonte: Autoria própria).

Para esta verificação é utilizado o comando *ping* descrito na seção sobre *Shell Script*. Um trecho deste script está descrito a seguir:

```
while ping –c1 $ip_pri > /dev/null; do
    echo “responde – IP principal está operando”
    ...
done
```

Neste trecho apresentado, o sistema retorna a mensagem de que o IP principal está operando, enquanto houver uma resposta positiva ao comando *ping*. É importante ressaltar que o comando *ping* deve ser executado uma única vez a cada repetição do laço *while*.

Se o IP principal não for constatado na rede pelo comando *ping*, a máquina escrava irá realizar a configuração de seu endereço de IP como principal, assumindo, assim, o “papel” da máquina mestre. Nesta condição, todos os serviços da máquina mestre passarão a ser prestados pela máquina escrava, de forma transparente para o usuário.

Contudo, se o IP principal estiver operando normalmente, a máquina escrava aguarda alguns segundos e faz uma nova verificação sobre o mesmo endereço IP. Neste laço, que é mantido até que haja alguma falha na máquina mestre, haverá a verificação para o momento do sincronismo entre os dados das máquinas, que deve ser realizado no sentido da máquina mestre para a máquina escrava. Este sincronismo será realizado de acordo com a especificação definida pelo administrador da rede através do arquivo de configuração do sistema, fazendo uso do comando *rsync*, como representado abaixo:

```
rsync - Cravzp --delete root@$ip_pri:${src[i]} ${dst[i]}
```

De acordo com a sintaxe do comando apresentada anteriormente, nota-se que todos os diretórios (previamente configurados no arquivo de configuração) são sincronizados obtendo-se o endereço IP da máquina mestre, e posteriormente são gravados no disco da máquina escrava. Os campos *src* e *dst* são vetores buscados no arquivo de configuração, elaborado para o armazenamento dos principais dados das duas máquinas. A opção *--delete* garante que todos os dados gravados em um backup realizado anteriormente sejam apagados, mantendo-se somente os dados alterados posteriormente ao último backup.

Após a realização do sincronismo de dados, a máquina escrava realizará uma nova verificação da operação do endereço IP principal. Além disso, o script “escuta” se manterá em execução em um laço infinito, ou até que o IP principal não seja mais identificado na rede pelo comando *ping*. Caso este evento ocorra, retorna-se ao script “principal” fazendo com que a máquina atual seja a máquina mestre.

É esperado que a máquina escrava fique em *loop* executando o script de escuta durante todo o tempo e caso detecte falta de operação no IP principal assuma as solicitações realizadas para a máquina principal. Dessa forma, a rede dos laboratórios do IFG se manterá em funcionamento contínuo por uma quantidade de tempo maior.

Um aspecto importante que deve ser observado na realização dos backups são as cotas de usuário. Considerada uma particularidade dos laboratórios do IFG, é mantido um sistema de cotas para armazenamento dos dados dos usuários, incluindo alunos e professores. Estas cotas devem ser desligadas antes de todo backup realizado e posteriormente religadas.

PRINCIPAIS RESULTADOS

Com as duas máquinas atuando como servidoras da rede local do IFG, é possível verificar a conectividade entre elas, assim como o desempenho do sistema implementado. Uma forma de verificar o comportamento do sistema é através da realização de alguns testes, onde são introduzindo erros no funcionamento do sistema e verificando como o sistema se comporta. Abaixo seguem alguns destes testes e o comportamento do sistema.

Teste 1: Desligamento do cabo de rede da máquina mestre

Retirando-se o cabo de rede da máquina mestre, perde-se a conexão com a rede dos laboratórios de informática. Deste modo, a máquina escrava não consegue identificar a atividade do número IP principal, e algumas medidas são tomadas imediatamente.

Como resposta a este tipo de falha, a máquina escrava configura seu IP como principal para assumir as tarefas que anteriormente eram executadas pela máquina mestre. Posteriormente, é mostrado na tela que a máquina está operando como servidora.

O sistema implementado, além de possuir as características do laboratório citado, evita alguns problemas de funcionamento em comparação com o *Heartbeat*. No caso de alguns testes que foram elaborados por Ferreira e Santos (2005) com o *Heartbeat,* foi constatado um erro de reconhecimento das máquinas, quando um dos cabos de rede é retirado e posteriormente recolocado. As autoras definem este erro como “síndrome do cérebro dividido”, que consiste na inicialização dos mesmos recursos por ambas as máquinas. No mesmo trabalho é sugerido o tratamento para este erro. No caso do sistema proposto e instalado nos laboratórios do IFG Campus Jataí, este erro não ocorre, pois é feita uma verificação da correta colocação dos cabos de rede ou do correto funcionamento das placas de rede dispostas para o sistema. Desse modo, nenhuma máquina assume o papel de outra sem antes garantir o bom estado de funcionamento destes dispositivos.

É importante destacar, que quando o cabo de rede é recolocado no sistema proposto, a máquina que antes era mestre agora passa a ser escrava e somente voltará a ser mestre caso ocorra alguma indisponibilidade por parte da máquina que no momento está atuando como mestre. Sendo assim, as máquinas invertem o seu papel de mestre e escrava a cada indisponibilidade presente na máquina servidora.

Teste 2: Desempenho da máquina mestre

Os scripts “principal” e “escuta” buscam no arquivo de configuração as informações de tempo para a execução dos comandos *ping* e *rsync*.

Para a realização dos testes de desempenho das máquinas, os intervalos de tempo para a execução do comando *ping* e atualização de backup foram alterados através do arquivo de configuração. Foram obtidos no total, 98 gráficos, dentre os quais 50 deles dizem respeito ao desempenho de processamento, memória e rede relacionados à máquina mestre, enquanto que os 48 restantes mostram o desempenho da máquina escrava.

Entretanto, por brevidade, somente os gráficos referentes à máquina mestre serão mostrados e analisados, por se tratarem do desempenho da máquina que estará prestando os serviços para a rede. Os resultados mais expressivos estão representados pelas Figuras 3, 4 e 5.

Na Figura 3, observa-se o gráfico de processamento da máquina mestre. Observam-se alguns picos no gráfico superior (sistema) e inferior (usuário), em intervalos de tempo regulares. Este valor é observado na maioria dos gráficos de CPU para esta máquina. Estas elevações

correspondem à execução do comando *rsync*, a cada 60 segundos. Entre elas, alguns ruídos podem ser observados, correspondentes a outras atividades sendo prestadas no momento, além da execução do comando *ping*, comprovando, portanto, que não há influência significativa deste comando sobre o processamento da máquina. Neste experimento o comando *ping* foi executado sem interrupção de intervalo de tempo.

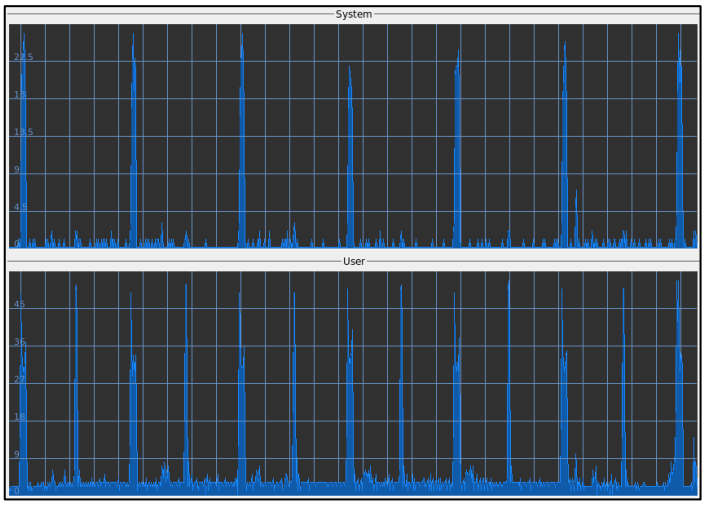

Figura 3: Desempenho de CPU com *ping* = 0 s e *rsync* = 60 s.

O desempenho de memória RAM é mostrado na Figura 4. Na parte superior do gráfico está representada a memória livre, enquanto que a parte inferior mostra a memória utilizada. Nota-se, no primeiro gráfico, que o comando *ping*, executado sem interrupção de intervalo de tempo, também não tem influência sobre a memória RAM. Entretanto, os backups provocam picos de memória livre, representados pela cor azul, alternados com vales que representam o espaço de memória utilizada. No gráfico inferior é possível realizar uma análise no contexto geral. O espaço que representa a memória utilizada pelas atualizações de dados (rsync), como se pode observar no gráfico, é praticamente insignificante na escala representada. Sendo assim, o sistema proposto não aumenta da forma significativa a utilização da memória RAM da máquina servidora.

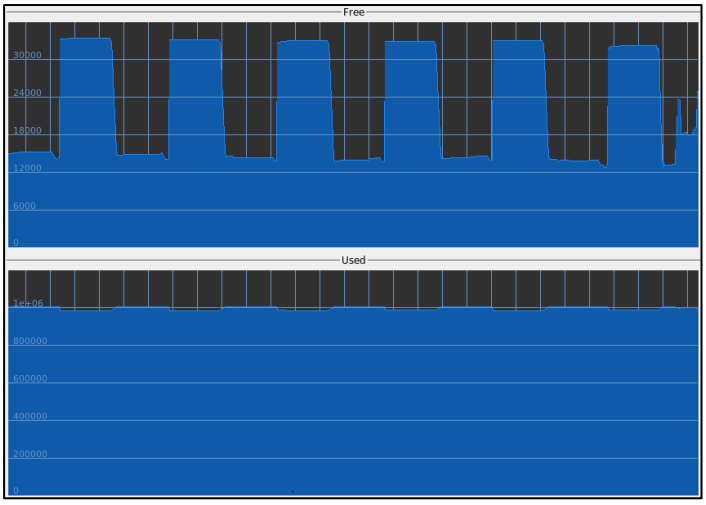

Figura 4: Desempenho de RAM com *ping* = 0 s e *rsync* = 60 s.

Na figura 5 está representado o desempenho de rede. Nos gráficos superiores são mostrados os dados e pacotes recebidos, e nos inferiores estão representados os pacotes transmitidos. Nota-se que os valores de transmissão de dados e pacotes são maiores que os valores de dados e pacotes recebidos. Algo que já era esperado, pois ao executar o sincronismo dos dados, a máquina servidora envia as alterações para a máquina escrava. Alguns picos excepcionais são observados, devidos aos processos comuns à rede, impossibilitando a boa visualização do restante dos picos regulares. Em contrapartida, as atualizações de dados realizadas em intervalos menores, como por exemplo, entre 30 segundos, provocam picos em intervalos bem menores. Além disso, é possível notar que o tráfego de rede, mesmo efetuando o sincronismo em intervalo de tempo curto (60 segundos), não é sobrecarregado, pois, nesse intervalo de tempo, poucas alterações são realizadas entre os dados, implicando em poucos dados a serem trafegados pela rede.

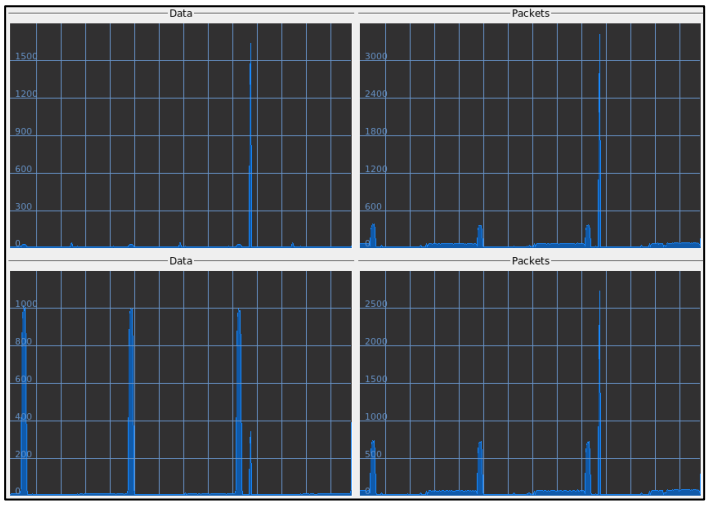

Figura 5: Desempenho de rede com ping = 0 s e rsync= 60 s.

CONCLUSÕES

A construção de um sistema tolerante a falhas com a utilização de *Shell Script* se mostrou promissora, pois os comandos em Linux possuem grande flexibilidade e podem ser aproveitados na construção de software com propósitos variados.

Sistemas tolerantes a falhas podem possuir custos variados, dependendo do grau de disponibilidade e confiabilidade a ser alcançado. Porém, uma das preocupações no desenvolvimento do presente trabalho refere-se à minimização dos custos sem a aquisição de novos equipamentos ou softwares, sendo empregado o reaproveitamento de máquinas subutilizadas.

O desenvolvimento do sistema foi realizado levando em consideração uma máquina mestre, responsável por prover os serviços para uma rede, e outra máquina escrava, possibilitando uma substituição da máquina mestre caso ocorra alguma indisponibilidade.

De acordo com os resultados obtidos, observa-se a pouca influência que a execução do comando *ping* tem sobre o processamento, memória RAM e tráfego de rede da máquina mestre. Para a execução do backup, é possível notar maior influência nesses parâmetros, indicando assim, a necessidade de intervalos de tempo mais longos para sua realização. Os resultados apresentados no artigo são provenientes de intervalos de sincronismo de 60 segundos, porém, foram realizados testes com intervalos menores, tal que quanto mais curtos forem os intervalos

de sincronismo, maior influência a máquina mestre sofrerá sob os parâmetros de carga de CPU, utilização de memória e tráfego de rede.

Os scripts implementados podem ser aplicados em qualquer sistema com estrutura semelhante à dos laboratórios de informática do IFG. Entretanto, deve-se realizar um estudo nesse sistema para identificar os melhores intervalos de tempo entre as execuções destes scripts. Devem-se observar nestes testes, características como freqüência do tráfego de rede, processamento e memória das máquinas servidoras em questão, além da disponibilidade do sistema, como intervalos de interrupções planejadas para manutenção e a previsão de paradas acidentais.

Embora os objetivos de alcançar certo nível de disponibilidade e confiabilidade no sistema proposto foram atingidos, é possível notar que ainda há melhorias que podem ser realizadas, como por exemplo, a verificação, não só da rede, mas sim dos serviços oferecidos pela máquina servidora. Além disso, foram realizados experimentos com apenas duas máquinas, porém, é esperado que ao utilizar mais máquinas atuando como escravas o sistema se comporte de maneira ainda mais robusta, apesar de que testes contemplando estas especificações ainda não foram realizados.

REFERÊNCIAS

BONAN, A. R. **LINUX: Fundamentos, Prática e Certificação LPI – Exame 117-101. Guia de Certificação para Administração do Sistema.** Rio de Janeiro: Alta Books, 2010.

FERREIRA, F. S.; SANTOS, N. C. G. G. D; ANTUNES, M. **Clusters de Alta Disponibilidade abordagem OpenSource.** Departamento de Engenharia Informática – ESTG, Leiria, Portugal, 2005. Disponível em: <www.4learn.pro.br/guarino/sd/TCCHA.pdf>. Acesso em: 08 ago. 2010.

FERREIRA, R. E. Linux: **guia do administrador do sistema.** São Paulo: Novatec Editora, 2008.

MORIMOTO, C.E. **Servidores Linux Guia Prático.** Porto Alegre: Sul Editores, 2009.

NEGUS, C.; TUCKER, W.; FOSTER-JOHNSON, E.; HAGEN, W. V.; VYAS, J. **Linux – A Bíblia Edição Especial.** Tradução Daniela Botelho. Edição especial. Rio de Janeiro: Alta Books, 2008, 717 p.

SCHRODER, C. **Redes Linux: Livro de Receitas.** Tradução Renata Rodrigues e Raquel Marques. Rio de Janeiro: Alta Books, 2009, 566 p.

SZTOLTZ, L; TEIXEIRA, R. S.; RIBEIRO, O.F. **Guia do Servidor Conectiva Linux.** Paraná, Conectiva Linux, 2003. Disponível em <http://conectiva.com/doc/livros/online/9.0/servidor/book.html>. Acesso em: 12 nov. 2010.

WEBER, T. S. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas.** Instituto de Informática – UFRGS, Rio Grande do Sul, 2002. Disponível em: <http://www.inf.ufrgs.br/~taisy/disciplinas/textos/indiceroteiro.htm>. Acesso em: 09 ago. 2010.