Número #16 - 29 de Janeiro de 2007

slackware zine

www.slackwarezine.com.br

Editorial

Mais um ano de SlackwareZine! E a terceira edição seguida que sai na data certa! Para comemorar, uma diagramação completamente nova e duas matérias especiais sobre clusters (hmmm... talvez fosse mais interessante conseguir três matérias para comemorar três anos).

Claro que três anos, dezesseis edições normais, diversas edições especiais e dois eventos não são coisas que se fazem sozinhas; por isso o grupo editorial do zine agradece todos os que mandaram e-mails de incentivo, artigos e deram aquela força para que o projeto continuasse, e continuasse com qualidade.

Com a diagramação nova também trouxemos algumas idéias que sempre passam pela cabeça mas não havíamos tido oportunidade de colocar em prática. Uma é a seção com análise de livros, todo mundo sabe que livro não custa barato e (garanto) muita gente já teve péssimas experiências gastando uma fábula em um livro e não gostando do conteúdo; com essa seção, é possível ter pelo menos uma avaliação do livro antes de comprar, o que pode ajudar bastante.

Outra novidade é uma página com dicas. Dicas são pequenos conselhos úteis, com uma seção assim a gente incentiva a participação (afinal escrever uma dica interessante é mais rápido que um artigo completo) e acaba aumentando a produtividade e facilitando a vida de bastante gente.

Acho que são essas as novidades e espero que todos gostem. Continuem lendo, recomendando e mandando artigos para o slackwarezine, a revista técnica de técnicos para técnicos.

Piter PUNK

Usando DNS Externo/Interno Sem View

Faça com que as máquinas de sua rede possam ter nomes diferentes para redes diferentes

Creio que todo mundo já teve essa necessidade, uma rede de servidores quando começa a passar de 3 já começa a encher o saco ficar digitando o ip :P, então o melhor mesmo é usar nomes, alguns usam nomes de deuses gregos, outros nomes astros do rock, whatever, o problema começa quando uma maquina tem que ter um nome para fora (leia-se internet) e um nome para dentro (leia-se intranet)

Quando eu precisei fazer isso, dei uma pesquisada na internet/amigos e a conclusão foi "use views" legal, lá fui eu no dns master e configurei, funcionou \o/, para fora ele resolvia com o ip 200.200.200.200 para dentro ele resolvia com o ip 192.168.0.10.

Legal, agora só falta sincronizar com o dns secundario, adivinha com qual zona o dns secundario sincronizava? exatamente com a zona interna :(aí quando alguem consultava o dns secundario ele respondia com ip's tipo, 192.168.0.x . Triste e desanimado com a tal das views foi quando eu lembrei que eu trabalhei em uma empresa que usava isso, só que não usava view! foi quando eu conheci o $ORIGIN :) que será melhor explicado agora.

No named.conf não muda nada:

```
zone "dominio.com.br" IN {
                type master;
                file "dominio.com.br";
                allow-transfer {
                   192.168.0.15;
             };  // IP do dns secundario
};
```

No arquivo de zona, mudam algumas coisas, como pode-se ver no Quadro 1. Com essa configuração se você resolver um nome dominio.com.br ele retorna um ip valido na internet, se você tenta resolver um intra.dominio.com.br ele retorna um ip invalido 192.168.x , e a sincronização master x secondary funciona que é uma beleza :)

Ah, não se esqueça de adicionar no resolv.conf das maquinas internas isso:

```
domain intra.dominio.com.br
nameserver 192.168.0.1
```

Nem só de View vive um DNS :) isso funciona muito bem, nunca tive problemas, e é mais facil de fazer que View.

Lindolfo Rodrigues <lorn@uplexis.com.br>

Quadro 1: Arquivo de Zona

```
$ORIGIN dominio.com.br. ; o segredo toda está nessa variavel $ORIGIN
$TTL 1200
@       IN      SOA thor.dominio.com.br. hostmaster.dominio.com.br (
                2007120101      ; serial
                7200            ; refresh 1 day
                3600            ; retry   1 hour
                1200            ; expire  1 week
                1200 )          ; minimum 1 day
; dominio.com.br (address, mail exchange e nameservers):
                        IN      MX      5       srv3-sao.sao.terraempresas.com.br.
                        IN      MX      10      mx-sec.terraempresas.com.br.

                        IN      NS      thor.dominio.com.br.
                        IN      NS      odin.dominio.com.br.

thor                    IN      A       200.200.200.200
odin                    IN      A       200.200.200.201
hercules                IN      A       200.200.200.202

$ORIGIN intra.dominio.com.br.
thor            IN      A       192.168.0.3
odin            IN      A       192.168.0.1
hercules        IN      A       192.168.0.2
```

Divertindo-se com Awk

Aprenda a usar o Awk, uma poderosa linguagem para manipulação de arquivos texto. Veja como ele pode substituir outros comandos e facilitar a sua vida.

O Awk é uma linguagem de script criada com basicamente um objetivo: comparar texto e tomar uma atitude de acordo com o texto localizado (ou seja, um processador de textos). O nome da linguagem é a junção das iniciais dos sobrenomes dos seus autores: Alfred Aho, Pete Weinberger e Brian Kernighan. Quem já leu o "Livro do Dragão" de compiladores, o "C Programming Language", ou escreveu um Hello, World deve ficar muito grato a eles... e quem usa Awk também!

A sintaxe mais básica do Awk é:

```
awk '/padrão/ { ação }'
```

Ou seja, toda vez que ele achar algo que case com /padrão/, ele vai executar uma determinada ação. A gente pode testar isso fácil:

```
$ awk '/localhost/ { print $0 }' /etc/hosts
127.0.0.1               localhost
```

Esse comando lê o arquivo /etc/hosts e procura onde existir o padrão "localhost", quando achar, ele imprime a linha na tela (o tal do $0). Como achar um padrão e colar a linha na tela é uma atividade muito comum (vide o grep) essa é a ação default do Awk então podemos reescrever o comando acima como:

```
$ awk '/localhost/' /etc/hosts
127.0.0.1               localhost
```

Bom, agora já sabemos como usar o awk como um "grep" alternativo -;). E, onde está "localhost" pode-se usar qualquer padrão, inclusive expressões regulares:

```
$ awk '/^[0-9]/' /etc/hosts
127.0.0.1               localhost
192.168.0.1             optimus.mylab optimus
192.168.0.2             rachael.mylab rachael
...
192.168.0.10            megaman.mylab megaman
```

Com o mesmo arquivo de exemplo (/etc/hosts), podemos fazer mais uma brincadeira:

```
$ awk '/localhost/ { print $2 }' /etc/hosts
localhost
```

O $2 significa o segundo campo da linha, o primeiro é o $1 e o terceiro o $3 (e as pessoas inteligentes já sacaram que o quarto é $4, o quinto $5 e assim sucessivamente). E, ao contrário do que muita gente pensa, não é fácil de fazer com o "cut":

```
$ grep localhost /etc/hosts | cut -d' ' -f2
127.0.0.1               localhost
```

Não funcionou porque entre o 127.0.0.1 e o localhost tem Tabs, e não espaços. Trocando os tabs por espaços, também não dá certo:

```
$ grep localhost /etc/hosts | cut -d' ' -f2
```

O grep acabou de mostrar o segundo espaço da linha. Para mostrar o localhost, precisaríamos fazer:

```
$ grep localhost /etc/hosts | cut -d'  '  -f16
localhost
```

Claro, usando tr -s " " dá para sumir com os espaços excedentes... Para isso que a inteligência do awk vem a calhar. Com um só comando a gente consegue detectar e mostrar o segundo campo, não importando se são Tabs, espaços ou a quantidade deles. Além de poder usar o Awk como um "grep" alternativo, podemos usá-lo como um "cut" também -;).

E, em termos de "cut" existe uma feature avançadíssima do Awk que quando menos se espera é tremendamente útil. Por exemplo, o /etc/hosts aqui é assim:

```
127.0.0.1               localhost
192.168.0.1             optimus.mylab optimus
192.168.0.2             rachael.mylab rachael
...
192.168.0.10            megaman.mylab megaman
```

Se eu quiser listar todos os IPs e o apelido de cada host, poderia listar $1 e $3; mas, com isso, não iria listar o 'localhost' que está no segundo campo. O que fazer?

```
$ awk '/^[0-9]/ { print $1"\t"$NF}' /etc/hosts
127.0.0.1       localhost
192.168.0.1     optimus
192.168.0.2     rachael
...
192.168.0.10    megaman
```

NF é o número de campos de cada linha. Então imprimimos o primeiro campo ($1) e o último ($NF). O "\t" é para imprimir um TAB e deixar tudo alinhadinho. E ainda combinamos dois comandos em um -;).

Parecido com o NF, o Awk tem uma segunda variável chamada NR: Número de Registro. Como normalmente um registro é uma linha, pode-se usar esse NR para indicar em qual linha o Awk está processando:

```
$ awk '{ if (NR==1) print $0 }' /etc/group
root::0:root
```

Usamos o comando "if" (se) e comparamos o número de registro, se for igual a 1, imprimimos a linha. Com isso conseguimos imprimir uma linha arbitrária qualquer. Com um pouco de imaginação, podemos fazer uma versão tosca do "head":

```
$ awk '{ if (NR<=10) print $0 }' /etc/group
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon
tty::5:
disk::6:root,adm,punk
lp::7:lp
mem::8:
kmem::9:
```

Ao invés do compararmos se o NR é igual a 1, verificamos se ele é menor ou igual a 10. Todas as operações de comparação "normal" funcionam: ==, !=, <=, >=, < e >, sem mistérios.

Depois de fazer o head, porque não inventar um novo comando tail? Esse é um pouco mais complicado, nós só sabemos o tamanho do arquivo quando ele já chegou no final, então temos que armazenar as linhas de texto em algum lugar. Esse exemplo vai ficar um pouco complicado:

```
$ awk '{ TEXT[NR]=$0 }
> END {
>       for (i=NR-9;i<=NR;i++) {
>         print TEXT[i]
>     }
> }' /etc/group
pop::90:pop
scanner::93:
nobody::98:nobody
nogroup::99:
users::100:
console::101:
_ntp::82:_ntp
postdrop:x:104:
haldaemon:x:61:
messagebus:x:60:
```

A primeira linha do nosso programa Awk realiza uma ação específica para todas as linhas encontradas: armazena o conteúdo da linha dentro da variável TEXT[NR] e, a gente lembra que NR é o número da linha. A segunda parte do programa é a mais "complicada". Nela a gente utiliza um padrão especial "END", que sempre "casa" com o final do arquivo (no nosso caso, quando todas as linhas foram lidas) utiliza um padrão especial "END", que sempre "casa" com o final do arquivo (no nosso caso, quando todas as linhas foram lidas); quando chegamos no final do arquivo, fazemos um laço de repetição:

```
for (i=NR-9;i<=NR;i++) { print TEXT[i] }
```

A tradução disso é:

```
i é igual a NR-9
para cada i que você encontrar e for menor ou
igual a NR
imprima o conteudo de TEXT[i]
e some 1 em i.
```

Ou seja, imprima as últimas 10 linhas do texto. É visível que isso de laço de repetição é algo muito útil, vale a pena guardar essa idéia. O padrão "END" também é extremamente útil e ele tem um irmão o "BEGIN" que "casa" com o início do arquivo (antes de qualquer linha ser lida). É uma boa para imprimir cabeçalhos ou algo do tipo.

Já que com a substituição do tail nós acabamos fazendo um script awk mais complexo, vamos chutar o pau da barraca e substituir o wc. O wc conta a quantidade de caracteres, palavras e linhas de um determinado texto:

```
$ wc /etc/fstab
 12  72 794 /etc/fstab
```

O primeiro número são as linhas, o segundo as palavras e o terceiro os caracteres. Contar as linhas todos nós já sabemos:

```
$ awk 'END { print NR }' /etc/fstab
12
```

Lembrando, o NR é o número do registro (linha); quando chegamos no evento END (final do arquivo), o NR contém o número da última linha. Agora vamos contar as palavras:

```
$ awk '{ WORD+=NF }
> END { print WORD }' /etc/fstab
72
```

Esse precisou de um pouco mais de imaginação, o que são as palavras dentro de uma linha se não os campos dela? Por isso, para cada linha lida, o conteúdo de NF é somado na variável WORD e, no final do arquivo, essa variável é impressa na tela. Para a última parte do wc, que é a contagem de caracteres, vamos precisar de um novo comando: length.

```
$ awk '{ CHAR+=length($0) }
> END { print CHAR }' /etc/fstab
782
```

O comando length($VAR) pega o conteúdo da variável e verifica o tamanho dela em caracteres. Ou seja, CHAR+=length($0) soma na variável CHAR todos os caracteres que tem na linha corrente. E, no final,

imprimimos a variável CHAR. Só tem um probleminha soma na variável CHAR todos os caracteres que tem na linha corrente. E, no final, imprimimos a variável CHAR. Só tem um probleminha... o número está errado! O wc informa 794 caracteres e o Awk informa só 782. O que acontece? A resposta é simples, o wc conta os caracteres de "salto de linha" (vulgo, o enter) e o Awk não. Para resolver essa disparidade, podemos somar no final a quantidade de saltos de linha que tem no arquivo:

```
$ awk '{ CHAR+=length($0) }
> END { print CHAR+NR }' /etc/fstab
794
```

E assim fica correto -:) Para finalizar, vamos colocar para imprimir todas as informações juntas, como o wc:

```
$ awk '{ CHAR+=length($0) ; WORD+=NF }
> END { print NR,WORD,CHAR+NR }' /etc/fstab
12 72 794
```

Pronto! Mais um comando substituído com sucesso!

Claro que o wc, o tail e o head ficaram muito maiores que os originais; mas serviram para mostrar vários recursos do Awk e a ensinar como usá-los. Já a substituição de grep+cut por Awk é bem interessante, além de ser extremamente mais versátil.

Umas últimas brincadeiras antes de fechar o artigo:

```
$ awk -F: '{ if ($3>=1000)
> { print $1 } }' /etc/passwd
punk
marina
toledo
garoto
tamiris
infomedia
slackshow
```

Isso mostra todos os usuários que não são do sistema na máquina. A novidade é o -F:, o -F determina qual vai ser o separador de campo, no caso, nós trocamos o espaço/tab pelo dois pontos, que é o caracter utilizado para separar as informações no /etc/passwd. Depois é só comparar o terceiro campo (UID) e verificar se é maior ou igual a 1000 (no slackware os usuários começam no 1000) e imprimir o primeiro -;).

Outro exemplo bom:

```
# ps aux | \
awk '/^usuario/ { system("kill -9 "$2) }'
```

Esse serve para derrubar um usuário (e todos os programas dele que estiverem sendo executados). A novidade aí é utilizar a instrução "system", com ela podemos executar um comando do sistema (no nosso caso o "kill -9") é fácil adaptar para ao invés de matar por um determinado usuário, matar pelo nome do programa (sim... para matar aqueles programas mal educados).

Esse é interessante para imprimir só o nome dos pacotes instalados, sem a versão e arquitetura:

```
$ ls -1 /var/log/packages/ | \
> awk 'BEGIN { FS="-" ; OFS="-" }
> { NF=NF-3 ; print $0 }'
a2ps
aaa_base
aaa_elflibs
...
x11-fonts-misc
x11-fonts-scale
x11-xdmx
...
ytalk
zlib
zsh
```

Utilizamos nesse comando o "BEGIN", que é executado antes de processar o arquivo. Como ele "casa" antes de qualquer linha de entrada ser processada, é o lugar ideal para colocar inicialização de variáveis. E, no nosso caso nós iniciamos duas delas: FS, que indica o separador de campo (é identico ao -F) e o OFS que é o separador de campo para saída; sem o OFS o nome do "x11-xdmx" sairia "x11 xdmx" que não é o que queremos. Já na outra linha vem um macete... os nomes dos pacotes do slackware são assim:

```
nome-versao-arquitetura-versao_do_pacote
```

E "nome" pode ter qualquer tamanho, separado por hífens. O que nós sabemos é que depois do nome, existem apenas três outros campos. Aí vem o truque. Para cada linha nós dizemos que ela possui três campos a menos (NF=NF-3) e depois é só imprimir a linha completa -;).

Bom, e com essas últimas dicas terminamos o artigo -:). Espero que tenha sido útil.

Piter PUNK
<piterpk@terra.com.br>

Optimizing Linux Performance

Um guia para maximizar o desempenho de aplicações para Linux

Ok, não é propriamente um artigo técnico; mas cobre um livro eminentemente técnico e sobre um assunto que acredito ser do interesse de vários administradores de sistemas e desenvolvedores: análise de performance. E o livro é bom.

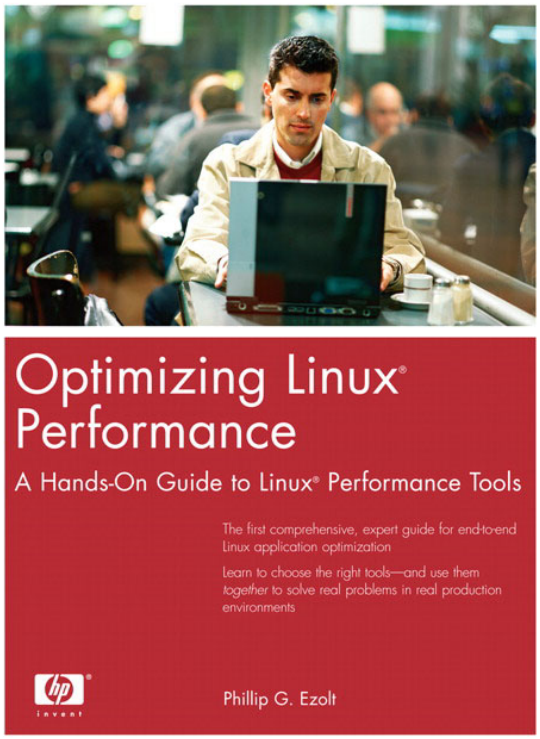
O próprio autor do livro oferece a motivação para a leitura do mesmo:

". Um sistema bem afinado pode fazer mais trabalho com menos recursos;
. Uma aplicação bem afinada pode rodar em hardware mais antigo;
. Um desktop bem afinado pode economizar o tempo dos usuários e;
. Um servidor bem afinado pode disponibilizar um serviço de maior qualidade para mais usuários."

E, para conseguir um sistema "bem afinado" é preciso primeiro saber o que está acontecendo, onde está o problema e como arrumá-lo.

Pode-se dizer que o livro vai muito bem na parte de identificação do problema, mostrando diversas ferramentas e o que cada uma delas indica; mas falha na hora do "como arrumá-lo". Como alguns comandos dão estatísticas de diferentes áreas do sistema, eles são revisitados várias vezes durante o livro (e sem repetir informações); assim, o comando ps é apresentado tanto na parte de análise de CPU, como na de memória e na de uso de disco. O vmstat e o oprofile também são presenças recorrentes.

Onde o livro deixa a desejar é na parte de como corrigir os problemas encontrados. Para quem tem como público-alvo "administradores de sistemas, desevolvedores de software e usuários finais" os exemplos de análise de performance e solução dados no final do livro não são muito animadores: Um problema em um filtro de imagem no GIMP, Problemas de Latência na Nautilus e Lentidões Periódicas devido ao Pre-Link. Administradores vão ficar deprimidos ao ver que nenhum dos exemplos cobre servidores de arquivos, web ou banco de dados que são os maiores pepinos que caem no colo dos sysadmins. E... bom, apesar dos exemplos serem de aplicações para o usuário final, não imagino muitos usuários editando o código dos filtros, rodando oprofile para identificar a linha de código com problema... aliás, não imagino o que um usuário final faria com um livro de análise de performance.



Optimizing Linux Performance
Phillip G. Ezolt
Prentice-Hall/HP Books
384 páginas

Mesmo com essa ressalva, o livro é uma boa compra e vale a pena. Apresenta uma série de boas ferramentas e de como usá-las. Apesar da escolha infeliz dos exemplos (ou feliz se você for um desenvolvedor) é visível o conhecimento do autor na área de análise de performance e otimização. E ele consegue ser bem didático e passar parte desse conhecimento para o leitor, o que é uma qualidade rara.

Piter PUNK
<piterpk@terra.com.br>

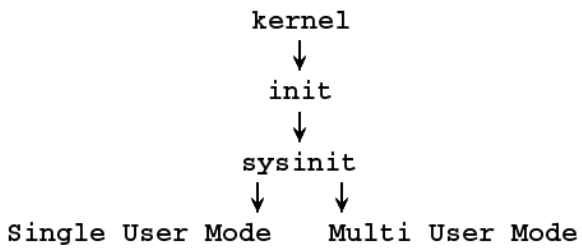
A Magia do init

O init é o primeiro processo chamado pelo kernel e, além de coordenar a inicialização do sistema, chamando os scripts de inicialização apropriados; também é o pai de todos os outros processos da máquina

Neste artigo vamos ver o porque do init ser tão importante e suas várias utilidades, desde a execução dos scripts de inicialização até o restart automático daquele serviço importantíssimo que nunca pode estar parado.

O init ou pai de todos os processos - como é mais conhecido - é responsável pela inicialização dos primeiros processos do sistema operacional. Esta fama é devido ao fato de ser o primeiro processo entrar em execução, nada mais digno de ser o dono do primeiro pid, o PID 1.

Vamos ver como é o fluxograma de execução do init:



Esse processo irá ficando mais claro conforme a leitura do artigo.

Definições do /etc/inittab

Toda a configuração do init é feita através do arquivo /etc/inittab seguindo a sintaxe:

```
id:runlevels:actions:process
```

Como vocês podem ver, cada campo é separado por dois pontos. Vamos entender o que significa cada um desses campos:

id Apenas uma identificação para diferenciar a chamada para o init. É possível utilizar até 4 caracteres nesse campo.

runlevels Aqui colocamos o número do runlevel que será executada a chamada. De 0-6, s ou S.

actions Para este campo temos 15 opções: respawn, wait, once, boot, bootwait, off, ondemand, initdefault, sysinit, powerwait, powerfailtnow, crtlaltdel e kbrequest.

process O processo em si que queremos que seja executado.

Quadro 1: Lista de "actions"

respawn Usado para iniciar e reinicializar o processo se o mesmo finalizar.
wait Usado para iniciar o processo. O init fica aguardando até que o processo seja finalizado.
once Usado para inicar o processo relacionado ao runlevel em específico.
boot O processo será executado durante o boot. (Para este campo, o runlevel é ignorado.)
bootwait O processo será executado durante o boot. Porém, o init espera pelo seu termino para seguir a diante. (Para este campo, o runlevel também é ignorado.)
off Para desabilitar o processo. (É possível fazer isto também comentando a linha.)
ondemand Serve para iniciar serviços quando é iniciado o runlevel A, B ou C. Que são runlevels especiais e quando iniciados não mudam o nível de execução da máquina.
initdefault Diz ao init qual o initdefault utilizado. (O campo process é ignorado).
sysinit Processo a ser executado ainda no Kernel Boot.
powerwait Processo que será executado ao ter uma falha de energia elétrica. O init fica aguardando até que o processo seja finalizado.
powerfail Igualmente para o de cima, apenas o init não espera o término do seu processo.
powerokwait Processo a ser executado quando a energia elétrica estiver estabilizada.
powerfailnow Processo a ser executado quando as baterias do no-breaks (se existir) estiverem vazias.
ctrlaltdel Processo a ser executado quando pressionarmos as teclas Ctrl+Alt+Del.
kbrequest Processo a ser executado quando o init receber um SIGNAL do teclado a ser definido.

Agora vamos analisar um inittab de verdade!

Para analisarmos, vou pegar como exemplo o arquivo de configuração /etc/inittab do Slackware 11.0, modificado pelo Patrick.

```
# Default runlevel. (Do not set to 0 or 6)
id:3:initdefault:
```

Como vimos no fluxograma, o init procura pelo initdefault no /etc/inittab para saber qual runlevel padrão irá executar.

Vamos fazer uma pausa e ver os runlevel's:

No /etc/inittab que estamos analisando, depois dos créditos e copyright, temos uma tabela comentada. (Habitue-se a ler scripts, arquivos de configuração antes de sair alterando. Muitos desenvolvedores colocam muita informação útil nos comentários, principalmente o Patrick ;)).

```
# These are the default runlevels in \
       Slackware:
#   0 = halt
#   1 = single user mode
#   2 = unused (but configured the same \
       as runlevel 3)
#   3 = multiuser mode (default Slackware \
       runlevel)
#   4 = X11 with KDM/GDM/XDM (session \
       managers)
#   5 = unused (but configured the same as \
       runlevel 3)
#   6 = reboot
```

O Patrick fez essa tabela e padronizou os runlevels 2, 3, 4 e 5. Na verdade, esses são os runlevels que podemos modificar. Os runlevels 0, 1 e 6 são reservados. Ou seja, não importa qual distribuição esteja, init 6 sempre rebootará seu sistema, já os inits 3 ou 4 podem ser que tenham comportamentos diferentes.

Em qualquer momento podemos mudar de runlevel apenas digitando init X. Onde X é o número do runlevel, de 0-6 e ainda podemos utilizar S ou s para single user mode.

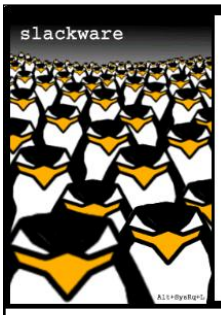
Voltando a primeira linha (descomentada) do inittab, ela diz ao init que o nosso runlevel padrão é o 3 (multiuser mode).

```
# System initialization (runs when system \
       boots).
si:S:sysinit:/etc/rc.d/rc.S
```

Este script será executado ainda em System boot. O S no campo runlevel não diz nada, pois os campos sysinit, boot e bootwait ignoram o campo runlevel.

```
# Script to run when going single user \
       (runlevel 1).
su:1S:wait:/etc/rc.d/rc.K
```

Este script será rodado apenas quando o runlevel for Single user. (O init irá esperar sua execução finalizar para pular para o próximo processo.)



slack users

Editores

Clayton Eduardo dos Santos
Deives Michellis
Leandro Toledo
Lindolfo Rodrigues
Piter Punk
Tiago Machado Costa

Artigos nessa edição

Clayton Eduardo dos Santos
Leandro Toledo
Lindolfo Rodrigues
Piter Punk
Sulamita Garcia

Dicas

Lindolfo Rodrigues
Piter Punk
Tiago Machado Costa

Imagem de Capa

Guaxinim
(baseado em ilustrução de Piter Punk)

```
# Script to run when going multi user.
rc:2345:wait:/etc/rc.d/rc.M
```

Script executado nos runlevels de 2 a 5, Multi user. (O init irá esperar sua execução finalizar para pular para o próximo processo.)

```
# What to do at the "Three Finger Salute".
ca::ctrlaltdel:/sbin/shutdown -t5 -r now
```

Esta linha diz que ao apertarmos a combinação Ctrl+Alt+Del, nosso sistema irá reinicializar em 5 segundos. (Podemos alterar esta combinação de teclas para o processo que desejarmos.)

```
# Runlevel 0 halts the system.
l0:0:wait:/etc/rc.d/rc.0
```

Ao entrar no runlevel 0, executar o script para desligar o sistema.

```
# Runlevel 6 reboots the system.
l6:6:wait:/etc/rc.d/rc.6
```

Ao entrar no runlevel 6, executar o script de reinicilizar o sistema.

```
# What to do when power fails.
pf::powerfail:/sbin/genpowerfail start
```

Para aquelas pessoas que tem aqueles super power ultra no-breaks, esta opção pode ser muito interessante. O no-break avisa que a uma falha de energia elétrica, o init detecta e executa o genpowerfail, que desliga automaticamente o computador.

```
# If power is back, cancel the running \
      shutdown.
pg::powerokwait:/sbin/genpowerfail stop
```

Assim como a opção de cima, se a energia elétrica estabilizar, este script cancela o desligamento do computador.

```
# These are the standard console login
# getties in multiuser mode:
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

Os id's de c1 a c5 são responsáveis de executar o /sbin/agetty, consoles para podermos logar no sistema (os famosos Alt+F1, Alt+F2, ..., Alt+F6). O id c6 também, porém tem o diferencial de ser inicializado no runlevel 4 (ambiente gráfico). O action _respawn_ diz ao sistema para reinicilizá-lo caso ele caia.

```
# Runlevel 4 used to be for an X window \
      only system, until we discovered
# that it throws init into a loop that keeps \
      your load avg at least 1 all
# the time. Thus, there is now one getty \
      opened on tty6. Hopefully no one
# will notice. ;^)
# It might not be bad to have one text \
      console anyway, in case something
# happens to X.
x1:4:respawn:/etc/rc.d/rc.4
```

Esta linha diz ao sistema para executar o script responsável pelo login gráfico, apenas no runlevel 4 (ambiente gráfico). O action _respawn_ diz ao sistema para reinicilizá-lo caso ele caia.

Serviços Automáticos

Com o init, podemos colocar qualquer processo para ser inicializado e reinicializado automaticamente, caso ele morra. O action _respawn_ é responsável por isso. Vamos ver um exemplo:

```
AP:3:respawn:/etc/rc.d/rc.httpd restart
```

Colocamos o apache para ser monitorado e reinicializado caso ele caia. Chamei o id de AP e essa ação é valida apenas para o runlevel 3.

```
# init q
```

O comando init q irá reler o /etc/inittab com a nova configuração. E o apache será reiniciado sempre que cair (e iniciado no boot).

Leandro Toledo
<toledo@slackwarezine.com.br>

Clusters de Balanceamento de Carga em Linux

O que fazer quando um servidor não agüenta mais executar as tarefas que deveria? A solução tradicional é comprar outra máquina mais poderosa mas nem sempre essa máquina existe ou cabe no orçamento; essa é a hora de pensar em balanceamento de carga e distribuir tarefas.

Cluster é um termo largamente utilizado para representar um conjunto de computadores combinados em um único sistema unificado, através de software e rede. Em termos mais simplificados, quando dois ou mais computadores são utilizados juntos para resolver um problema, isto é considerado um cluster. Clusters são tipicamente utilizados para Alta Disponibilidade (HA - High Availability) para maior confiabilidade, ou Computação de Alta Performance (HPC - High Performance Computing) para fornecer maior poder computacional do que um único computador pode fornecer."[1]

Um outro tipo de cluster é o cluster de balanceamento de carga(LB - Load Balance). Este cluster utiliza várias máquinas para atenderem as requisições de determinado serviço, agindo como um grande computador com capacidade para muitas requisições paralelas. Este conceito acaba proporcionando, de maneira indireta, HA e HPC, por combinar várias máquinas em um ambiente que pode processar mais requisições do que uma única máquina, e por aumentar a confiabilidade de um sistema.

Os componentes de um ambiente de LB são dispostos em nodo(s) diretor(es) - o(s) director(s) ou load balancer - e os servidores reais. O papel do director é receber as requisições dos clientes e repassar para os servidores reais, de acordo com algumas regras estabelecidas. A quantidade de conexões repassada para um dos servidores reais pode seguir vários métodos - desde o algoritmo mais simples, onde cada máquina recebe uma conexão por vez, ate a combinação de medidas e prioridades, atribuindo um peso a cada servidor real.

No Linux, o projeto mais completo para ambientes de load balance é o LVS - Linux Virtual Server. O LVS é um modulo inserido no kernel (Linux Kernel Configuration -> Networking -> Networking Options -> IP: Virtual Server Configuration), e é controlado através do lvsadmin, um comando define as regras para montar o ambiente: quem são os servidores reais, que serviços eles atendem e qual a carga de conexões para cada um. O próprio node director pode atender estas requisições ou não, depende da configuração.

Estes ambientes são facilmente escaláveis. Uma máquina pode ser inserida no ambiente sem impactar nos serviços e máquinas já incluídos. E se uma máquina deixa de responder no ambiente, o director a coloca de "quarentena", removendo-a da lista de servidores reais e checando periodicamente se ela voltou ao ambiente. Enquanto ela não responder, o director não envia mais requisições a esta máquina. A alta disponibilidade então é conseguida desta forma, mesmo que uma máquina deixe de responder, as outras continuarão atendendo o serviço. Precisamos então nos preocupar com a alta disponibilidade do nodo director, pois se ele deixar de responder, todo o ambiente pára.

Persistência

Muitas questões precisam ser avaliadas quando pretendemos montar um ambiente de cluster. Além, claro, da capacidade das máquinas para processar o volume de dados em tempo satisfatório, precisamos pensar no fluxo das informações. Por exemplo, o ambiente de load balance parece perfeito para melhorar a vida dos usuários, porém algumas situações ele pode não agir da forma esperada.

Geralmente, quando falamos de LB, estamos pensando em conexões isoladas. Desta forma qualquer servidor real pode atender a qualquer conexão que estiver chegando, sem se preocupar de onde a conexão está vindo ou para onde ela vai. Porém para alguns serviços e situações isto não é sempre verdade. Consideremos por exemplo o protocolo FTP.

Quando um cliente estabelece uma conexão FTP, existe uma conexão de controle (que segue pela porta

21) para enviar comandos e a conexão dos dados propriamente ditos (que seguem pela porta 20). Quando a conexão é ativa, o cliente informa ao servidor qual porta que atende à conexão de dados para transferí-los. Porém para uma conexão passiva, o servidor informa ao cliente a porta que ele está atendendo, e então o cliente inicia uma conexão para aquela porta. Para ambientes em VS/TUN e VS/DR, o director atua apenas na conexão cliente-servidor, não no caminho inverso, então é impossível para o director capturar a porta do pacote que vai diretamente ao cliente.

Para resolver isto, precisamos ativar a persistência entre as conexões. Desta forma, o LVS irá manter uma tabela com o registro desta conexão em uma tabela hash, contendo os dados do IP e porta da origem, IP e porta do destino virtual, IP e porta do destino no servidor real. Para FTP, o valor das portas é 0, então qualquer conexão vinda do cliente é redirecionada para o mesmo servidor real e associados à mesma conexão, de acordo com a entrada salva na tabela. Para outros serviços, as portas de destino são mantidas porém a do cliente pode variar que o LVS vai manter a persistência para o mesmo servidor real.

Porém as vezes nem isto é suficiente. Por exemplo, existem grupos de proxys para acessos em determinados sites, para diminuir a carga. Só que para o cliente, ele pediu uma requisição para um servidor e outro atendeu. Então um outro tipo de persistência seria de um range de rede, que é mantido para saber que conexões para aqueles ips precisam ser persistentes.

Como o LVS redireciona as conexões

O LVS pode redirecionar as conexões vindas dos clientes, para os servidores reais, através de NAT, túnel IP ou roteamento direto (Direct Routing). No caso de túnel IP ou DR, existe um efeito colateral que precisamos estar atentos.

Pelo NAT, quando o director recebe uma conexão para um serviço que está configurado, ele escolhe um servidor real de acordo com as regras, modifica o IP de destino com o IP do servidor real escolhido e reenvia. Ao receber a resposta, o director volta o pacote para o cliente, contendo o IP original como origem da resposta. Isto geralmente é usado em redes locais, e o mascaramento geralmente ocorre do director com IP válido para a Internet e uma rede local de IPs inválidos, por exemplo.

Pelo túnel IP, o director adiciona outro encapsulamento IP sem modificar o cabeçalho original. Isto é perfeito para redes geográficamente distribuídas, onde os servidores reais não tem necessariamente o mesmo endereço de rede. Isto porém exige que os servidores reais tenham suporte a encapsulamento IP.

No DR, o director redireciona os pacotes diretamente ao servidor real. A partir disso a comunicação com o cliente fica a cargo do servidor real, e o director não atua mais nesta conexão. Para fazer isto, o director modifica o cabeçalho do pacote, adicionando o MAC do servidor real como endereço físico para o IP de destino, e o retransmite pela rede. Isto pode causar alguns problemas.

Nos clusters de DR e tunelamento IP, o IP de serviço é compartilhado entre o director e os servidores reais. Todos tem uma interface configurada com este IP. Em algumas configurações que os servidores reais estão na mesma rede que o director, e os servidores reais responderem uma requisição ARP, haverão problemas. Os pacotes serão respondidos hora pelo director, hora por um servidor real, hora por outro, e o cluster inteiro não vai funcionar direito. Por isto, nestes clusters, precisamos garantir que apenas o director irá responder as requisições ARP para o IP de serviço.

Existem algumas formas como isto pode ser configurado, e explicá-las é assunto suficiente para outro artigo. Caso queira saber mais sobre isto, consulte [2].

Algoritmos

O LVS pode distribuir as conexões baseado em vários tipos de algoritmos: Round-Robin, por peso, pelo servidor que tem menos conexões e outros. Todos tem suas vantagens e desvantagens, devendo o administrador escolher de acordo com o poder de processamento dos servidores reais, com a quantidade de conexões recebidas e com o tipo de carga destas conexões. Os mais comuns são estes:

Round-Robin: Seqüencial, cada servidor recebe uma conexão por vez. Desta forma, em um cluster de três servidores reais, a primeira conexão será enviada para o servidor real 1, a segunda para o servidor real 2, a terceira para o servidor real 3, a quarta para o servidor 1 e assim por diante. Este algoritmos trata todos os servidores reais igualmente, sem levar em conta o número de conexões recebidas ou o tempo de resposta que cada servidor fornece.

Round-Robin com Pesos: Cada servidor recebe um peso e o director irá distribuir as conexões com base nestes pesos. Assim, um servidor real com peso 4 irá receber 2 vezes mais conexões que um servidor real com peso 2, que irá receber 2 vezes mais conexões que um servidor real com peso 1. É um algoritmos melhor estruturado para ambientes em que os servidores reais com diferentes velocidades. Porém se as conexões forem de cargas variadas, o servidor real com maior prioridade pode acabar recebendo as conexões de maior peso e acabar sobrecarregado, enquanto outros servidores de menor prioridade permanecem desocupados.

Menos conexões: as conexões são redirecionadas para o servidor real com menos conexões ativas. Este é um algoritmo dinâmico, pois precisa checar quantas conexões estão ativas em determinado instante. É um bom algoritmo para quando a quantidade de conexões

recebidas varia bastante.

Menos conexões com pesos: combina a busca pelo servidor real com menos conexões, porém atribuindo pesos a estes. Um servidor real com capacidade duas vezes maior que um outro terminará de processar suas conexões e ficará ocioso, enquanto outros podem ficar sobrecarregados. Combinando estes dois tipos, o algoritmo dinamicamente descobrirá qual o servidor pode atender melhor em determinado momento.

Existe também outros algoritmos como o hash[3] - o servidor monta uma tabela hash e redireciona as conexões para os servidores por esta tabela, com ou sem pesos -, e o de "sem filas" - buscando sempre um servidor idle ou com menor tempo de resposta.

Configurações

Finalmente, vamos colocar a mão na massa. Para quem quer saber como funciona a base de tudo, o ipvsadm. O software pode ser encontrado em [4], caso sua distribuição não tenha o pacote pronto. Aliás, caso não tenha, envie como sugestão para o mantenedor ;)

Após a receita padrão - make && make install - você terá instalado o ipvsadm, ipvsadm-save e ipvsadm-restore. Você pode pensar no ipvsadm como similar ao iptables - ele irá adicionando as regras e buscá-las na ordem que você especificar. Vamos detalhar melhor com exemplos. Para o protocolo FTP, lembrando de ativar a persistência:

```
ipvsadm -A -t virtual.domain.org:ftp -p 540
ipvsadm -a -t virtual.domain.org:ftp -r
192.168.1.2 -m
ipvsadm -a -t virtual.domain.org:ftp -r
192.168.1.3 -m
```

A primeira linha adiciona um serviço nas regras com a opção -A, dizendo que para isto o protocolo usado é o tcp (-t), e que vai responder pelo endereço virtual.domain.org(poderia ser um IP, geralmente o IP de serviço) no serviço de ftp. Ativamos a persistência com -p, e neste caso configuramos o timeout para 540 segundos(poderia não ser especificado e o ipvsadm assumiria o valor padrão de 300 segundos).

As linhas seguintes adicionam servidores reais com a opção -a ao serviço especificado após o -t, dizendo que o endereço do servidor real é o IP que segue a opção -r e usando o método de mascaramento por NAT(-m).

A partir daí, podemos adicionar vários serviços com algoritmos diferentes, e servidores reais distintos. Porém o ldirectord agrupa esta configuração toda em um único arquivo, se encarregando de carregar as regras e controlar os servidores reais.

Ultramonkey, LVS e ldirectord

O Ultramonkey é um projeto que combina o LVS, gerenciado através do ldirectord, e o heartbeat para gerenciar as regras e serviços no director. O heartbeat ira fazer a alta disponibilidade do director, e a alta disponibilidade dos servidores reais é intrínseco ao LVS. Ele permite várias combinações e opções para melhor adaptação a diferentes ambientes e necessidades.

Quadro 1: Servidor HTTP

```
# Virtual Server for HTTP
virtual=192.168.6.240:80                IP Virtual (de serviço)
        fallback=127.0.0.1:80           Se nenhum servidor real
                                        responder, quem responde
        real=192.168.7.4:80 masq        Servidor real, IP, porta e
                                        modo de conexão
        real=192.168.7.5:80 masq        Servidor real, IP, porta e
                                        modo de conexão
        service=http                    Qual é o serviço
        request="index.html"            Que página buscar(opção
        receive="Test Page"             para serviços http)

        scheduler=rr                    Qual o escalonador
                                        (algoritmo) - RoundRobin
                                        neste caso
        #persistent=600                 Persistência (desabilitada
                                        neste caso)
        protocol=tcp                    Protocolo
        checktype=negotiate
```

Além de gerenciar as regras do LVS, o ldirectord também checa periodicamente se os serviços oferecidos pelos servidores reais estão funcionando corretamente. Ele possui suporte a servidores HTTP, HTTPS, FTP, IMAP, POP, SMTP, LDAP, NNTP e MySQL. De acordo com o serviço configurado, ele estabelece uma conexão apropriada com o serviço e checa se está tudo funcionando. Os software pode ser encontrado em [5]. O arquivo de configuração é o ldirectord.conf, geralmente instalado no /etc. Vamos ver algumas opções do arquivo de configuração.

Para o serviço apresentado no Quadro 1, o director irá se conectar nos servidores reais e buscar a página index.html. Se algum erro acontecer, ele removerá o servidor real da sua lista.

No Quadro 2 o serviço é FTP, algumas opções

mudaram, para incluindo usuário e senha para checar não apenas se a porta está ouvindo, mas se o serviço está executando a contento.

Como este serviço é o MySQL, adicionamos qual banco de dados e qual query enviar para checar o serviço.

Caso estejam curiosos de o que é o checktype, ele é o parâmetro que define como vai ser a checagem. Ele pode ser negotiate, connect, N, off ou on. Connect irá apenas tentar fazer uma conexão TCP/IP, então não precisamos das opções request e receive. Se está configurado para negotiate, o ldirector irá enviar uma requisição conforme

Quadro 2: Servidor FTP

```
virtual=192.168.6.240:21                IP Virtual(de serviço)
        fallback=127.0.0.1:21           Se nenhum servidor real
                                        responder, quem responde
        real=192.168.7.4:21 masq        Servidor real, IP, porta e
                                        modo de conexão
        real=192.168.7.5:21 masq        Servidor real, IP, porta e
                                        modo de conexão
        service=ftp                     Qual é o serviço
        request="welcome.msg"           Que arquivo requisitar
        receive="Welcome"
        login="anonymous"               Com qual usuário
        passwd="anon@anon.anon"         Com qual senha
        scheduler=rr                    Qual o escalonador
                                        (algoritmo) - RoundRobin
                                        neste caso
        #persistent=600                 Persistência (desabilitada
                                        neste caso)
        protocol=tcp                    Protocolo
        checktype=negotiate
```

Quadro 3: Servidor MySQL

```
Virtual = 192.168.6.240:3306            IP Virtual (de serviço)
       fallback=127.0.0.1:3306 masq     Se nenhum servidor real
                                        responder, quem responde
       real=192.168.7.4:3306 masq       Servidor real, IP, porta e modo
                                        de conexão
       real=192.168.7.5:3306 masq       Servidor real, IP, porta e modo
                                        de conexão
       login = "readuser"               Com qual usuário
       passwd = "genericpassword"       Com qual senha
       database = "portal"              Qual BD checar
       request = "SELECT * FROM link"   Qual a query
       scheduler = wrr                  Qual o escalonador (algoritmo):
                                        RoundRobin com pesos neste caso
       checktype = negotiate
```

especificada em request, e na resposta procurar a string definida em receive. Se a opção for configurada com um número N, então a cada N conexões o ldirectord irá efetuar uma checagem. Off desliga a checagem, e on serve mais para ativar a checagem caso você tenha parado momentaneamente.

Embora extenso, este artigo tem um apanhado geral de um assunto com muitas mais possibilidades: otimização, sincronização, defesas contra DoS.

Sulamita Garcia
<sulamita@linuxchix.org.br>

Referências:

http://www.austintek.com/LVS/LVS-HOWTO/\
      mini-HOWTO/LVS-mini-HOWTO-pt.html
http://listas.linuxchix.org.br/mailman/\
      listinfo/linux-ha
[1] http://www.beowulf.org/overview/index.html
[2] http://www.linuxvirtualserver.org/\
      docs/arp.html
[3] http://pt.wikipedia.org/wiki/\
      Tabela_hash
[4] http://www.linuxvirtualserver.org/\
      software/ipvs.html
[5] http://www.ultramonkey.org/download/3/

Construindo um Cluster de OpenMosix com o Slackware 11.0

Nesse artigo, iremos implementar um cluster openMosix baseado em kernel 2.4.26 utilizando como base, o bom e velho slackware. :)

Introdução

É desnecessário destacar a importância que os computadores possuem nos dias atuais. Também é desnecessário dizer que existe um "lobby" muito grande entre "alguns" fabricantes de hardware e software que insistem em dizer ao usuário que ele precisa de uma máquina Dual-Core com 1 ou 2 gigabytes de memória RAM para navegar na internet, ler seus e-mails e escrever documentos de texto com qualidade gráfica satisfatória, o que, na prática, caracteriza uma venda casada "disfarçada". No entanto, ao contrário dos exemplos anteriores, é importante dizer que algumas aplicações são de fato, extremamente dependentes de recursos computacionais, como por exemplo, a renderização de animações 3D complexas e softwares de modelagem e simulação específicos empregados em aplicações científicas.

Havia um tempo em que se pensava que aplicações como essas só poderiam ser concluídas mais rapidamente caso novos hardwares fossem adquiridos, com processadores mais velozes e maior quantidade de memória RAM. É evidente que essa é uma solução, no entanto, dependendo da aplicação utilizada, ela não é a única e tampouco a mais vantajosa.

Com o surgimento da computação distribuída, a utilização de recursos de processamento, memória e armazenamento de computadores distintos aplicados na execução de um objetivo comun tornou-se possível, dando início a uma nova realidade que possui como principais benefícios a possibilidade real de variação do fator escalabilidade e uma maior viabilidade do ponto de vista econômico.

Atualmente existem várias aplicações fantásticas que fazem uso desse tipo de tecnologia de maneira extremamente eficiente. As grandes "vedetes" são os clusters e os servidores de terminais gráficos, que permitem a utilização de hardware anteriormente considerado obsoleto mas que, quando utilizado dessa forma, ainda possuem uma sobrevida considerável.

Como o assunto em questão são os clusters, vou tentar dar uma visão geral do assunto, ainda que breve, mas espero que suficientemente esclarecedora. Existem basicamente três tipos principais de cluster:

Clusters de alta disponibilidade: Consistem de um conjunto de duas ou mais máquinas com serviços "redundantes" e interligadas entre si. O motivo da existência de uma conexão exclusiva entre essas máquinas "redundantes" é a frequente monitoração dos serviços fornecidos pela máquina mestre realizada pelas máquinas escravas, através de mensagens do tipo "keep alive", por exemplo. Se uma das máquinas escravas ou secundárias, detectar que o serviço provido pela máquina mestre não está operacional por algum motivo, esta assume o controle do serviço no lugar da máquina primária.

Clusters de balanceamento de carga: Esse tipo de cluster tem como finalidade distribuir requisições de serviço originadas de máquinas clientes entre um conjunto de servidores idênticos da melhor forma possível, considerando como métrica a disponibilidade de recursos destes, minimizando dessa forma, a ocorrência de eventos de indiponibilidade de serviços, por exemplo. A medida que a demanda de acesso ao serviço aumenta, basta que o administrador adicione novos nós ao cluster de servidores.

Clusters voltados para computação de alto desempenho: É nessa categoria que o cluster openMosix (entre outros) se enquadra. A finalidade desse tipo de cluster é distribuir processos entre as máquinas que compoem o cluster. Cada uma dessas máquinas é também chamada de nó. A métrica utilizada para a eleição da máquina ideal é a disponibilidade de recursos de memória e cpu de cada um dos nós no momento em que o processo deve ser exportado. Desse modo, máquinas ociosas que estejam executando processos considerados simples são fortes candidatas a receberem processos remotos de outros nós do cluster.

O openMosix diferencia-se de outros tipos de cluster, por não exigir mudanças no código fonte das aplicações nele utilizadas. Quando aplicado ao kernel Linux, ele se encarregada de exportar os processos entre os nós do cluster e receber os resultados de volta. Outros tipos de cluster populares, como o Beowulf, por exemplo, utilizam bibliotecas específicas (PVM e MPI) que realizam essa tarefa, mas exigem que a aplicação seja escrita considerando aspectos de paralelismo, o que, de certo modo, limita consideravelmente sua utilização.

Instalação

A implementação do cluster openMosix é relativamente simples, afinal, trata-se na prática, de apenas um patch de kernel. No entanto, alguns pontos da documentação oficial estão desatualizados e podem causar alguma confusão. O sistema de arquivos nativo do openMosix, o oMFS, por exemplo, foi retirado a partir da versão 2.4.26-1 por questões de segurança. Um outro "problema" ocorre durante o processo de compilação do kernel, em função da versão do gcc utilizado no slackware 11. Desse modo, teremos também que trabalhar com uma versão anterior do compilador para que o código compile sem problemas. Mãos a obra:

Durante todo o processo de instalação/configuração, por questões de praticidade, estou considerando que todos os downloads estão sendo armazenados no diretório /root e que "root" seja o usuário corrente.

Baixando os arquivos necessários:

```
# wget http://www.if-usp.lkams.kernel\
       .org/pub/linux/kernel/v2.4/\
       linux-2.4.26.tar.bz2
# wget http://ufpr.dl.sourceforge.net/\
       sourceforge/openmosix/\
       openMosix-2.4.26-1.bz2
# wget http://ufpr.dl.sourceforge.net/\
       sourceforge/openmosix/\
       openmosix-tools-0.3.6-2.tar.gz
```

Resolvendo o problema de compatibilidade do gcc:

```
# wget http://ftp.belnet.be/packages/\
       slackware/slackware-9.1/\
       slackware/d/\
       gcc-3.2.3-i486-2.tgz
# removepkg gcc-3.4.6-i486-1.tgz
# installpkg gcc-3.2.3-i486-2.tgz
```

O primeiro passo consiste em instalar o kernel correspondente à versão mais recente do openMosix e em seguida aplicar o patch sobre ele. Nesse caso, estamos falando do kernel 2.4.26, cujo download foi realizado na seção anterior. Vamos ao trabalho:

```
# cd /usr/src
# cp /root/linux-2.4.26.tar.bz2 .
# tar -xvjf linux-2.4.26.tar.bz2
# cd linux-2.4.26
# cp /root/openMosix-2.4.26-1.bz2 .
# bunzip2 openMosix-2.4.26-1.bz2
# patch -Np1 < openMosix-2.4.26-1
# cp config .config
# make menuconfig
# make dep
# make bzImage
# make modules
# make modules_install
```

Em seguida, após algumas xícaras de café, vamos adicionar uma entrada referente ao kernel recém-compilado ao lilo:

```
# cd arch/i386/boot
# cp bzImage /boot/vmlinuz-2.4.26
# vim /etc/lilo.conf
```

Adicione uma entrada no seu menu de boot apontanto para /boot/vmlinuz-2.4.26, semelhante ao exemplo a seguir:

```
# LILO configuration file
# generated by 'liloconfig'
#
# Start LILO global section
boot = /dev/hda
prompt
timeout = 500
# VESA framebuffer console \
       @ 1024x768x256
vga = 773
# Linux bootable partition config\
       begins
image = /boot/vmlinuz
  root = /dev/hda1
  label = Slackware
  read-only # Non-UMSDOS filesystems\
       should be mounted read-only for\
       checking
# Linux bootable partition config ends
# Sua entrada referente ao openMosix\
        começa aqui
# openMosix bootable partition config\
        begins
image = /boot/vmlinuz-2.4.26
  root = /dev/hda1
  label = openMosix
```

```
  read-only # Non-UMSDOS filesystems\
        should be mounted read-only for\
        checking
# openMosix bootable partition config\
      ends
### E termina aqui... :) ###
```

Atualize o Lilo:

```
# lilo
```

Se tudo der certo, você deve ver algo do tipo:

```
Added Slackware *
Added openMosix
```

Atenção: É importante ter certeza que os parâmetros "prompt" e "timeout" estejam presentes em seu lilo.conf, para que você possa optar pelo boot utilizando o kernel openMosix.

O próximo passo é instalar algumas ferramentas de gerenciamento do cluster openMosix. Inicialmente, vamos criar um link simbólico necessário durante o processo de compilação das ferramentas que serão utilizadas no cluster:

```
# ln -s /usr/src/linux-2.4.26 /usr/src/\
      linux-openmosix
```

Agora, vamos instalar as ferramentas:

```
# cd /root
# tar -xvzf openmosix-tools-0.3.6-2.tar\
      .gz
# cd openmosix-tools-0.3.6-2.tar.gz
# ./configure
# make && checkinstall
# installpkg openmosix-tools-0.3.6-2-\
      i386-1.tgz
```

Agora iremos editar alguns arquivos de configuração necessários para o funcionamento do cluster:

```
# vim /etc/openmosix.map

# Static openMosix configuration
# ============================
#
# Each line in this file should contain\
      3 fields, statically mapping
# IP addresses to openMosix node-numbers:
#
# 1) The first openMosix node-number in\
      this range.
# 2) The IP address of the above node\
      (or node-name from /etc/hosts).
# 3) The number of nodes in this range.
#
# Note: If you don't create a valid IP\
      <-> node-number mapping, the\
#       autodiscovery daemon will be\
      started, automatically assigning
#       node-numbers to all visible\
      openMosix machines.
#
#Example: 10 machines with IP addresses\
      192.168.1.50 - 192.168.1.59
#          which will have openMosix\
      node-numbers 1-10:
#
# 1        192.168.1.50     10
#
# MOSIX-#  IP  number-of-nodes
# ============================
1       192.168.20.15     1
2       192.168.20.16     1
3       192.168.20.17     1
4       192.168.20.18     1
```

Essa é uma maneira de se especificar os nós do cluster, ou seja, do lado esquerdo identificar o número do nó, no centro especificar o ip do nó e no final especificar quantos nós devem ser considerados a partir do ip especificado. Desse modo, a seguinte sintaxe também seria válida, considerando que o range de ips que compoem o cluster aqui exemplificado são sequenciais:

```
# MOSIX-#  IP  number-of-nodes
# ============================
# Essa instrução é equivalente à anterior
1       192.168.20.15     4
```

O último passo consiste em adicionar a instrução responsável pela inicialização do nó do cluster em um script executado após o boot:

```
# vim /etc/rc.d/rc.local
# Starting openMosix node
/usr/local/sbin/setpe -w -f /etc/\
      openmosix.map
```

Agora vamos reinstalar o gcc "original", disponível em seu CD/DVD de instalação do slackware11 e remover o release instalado no início do processo:

```
# removepkg gcc-3.2.3-i486-2.tgz
# installpkg gcc-3.4.6-i486-1.tgz
```

Por fim, vamos reiniciar a máquina e bootar com o kernel do openMosix:

```
# init 6
```

Adicionando mais nós ao seu cluster

Adicionar novos nós ao seu cluster é uma operação simples se você já chegou até aqui. Basta repetir os procedimentos anteriores em cada um dos nós e manter o arquivo /etc/openmosix.map com o mesmo conteúdo dos outros nós, sempre verificando se todos os nós que irão compor o cluster estão especificados nesse arquivo. Em nosso exemplo, o cluster é constituído de quatro nós, que atendem nos ips 192.168.20.15, 192.168.20.16, 192.168.20.17 e 192.168.20.18.

Testando o cluster

Agora iremos testar a funcionalidade do cluster, escrevendo e depois executando um pequeno script sugerido na própria documentação do openMosixi [1]:

```
# vim script_cluster.sh

awk 'BEGIN{for(i=0;i<10000;i++)\
       for(j=0;j<10000;j++);}'
```

Vamos dar permissão de execução a ele:

```
# chmod +x script_cluster.sh
```

Abra um segundo terminal e execute o seguinte comando:

```
# msmon
```

O msmon, como o próprio nome sugere, é um monitor gráfico que mostra a carga de processamento de cada um dos nós que compoem o cluster. É uma excelente ferramenta para verificar o funcionamento do cluster em si e ao mesmo tempo, diagnosticar problemas de rede ou de sintaxe (dos arquivos de configuração) em nós isolados, uma vez que somente os nós corretamente configurados serão listados no eixo horizontal. Se um dos nós de seu cluster não aparecer nesse eixo, você possivelmente tem problemas com sua rede, com o arquivo /etc/openmosix.map do nó "desaparecido" ou ainda pode ter esquecido de executar o comando "setpe" após o boot nesse nó.

Em seguida, vamos executar o script no primeiro terminal algumas vezes consecutivas com o seguinte comando (umas 4 ou 5 vezes devem bastar):

```
# ./script_cluster.sh &
```

Verifique em seu segundo terminal, que roda o mosmon que, se tudo correu bem, a carga de processamento de cada um dos nós que compoem o cluster recebe uma carga de processamento, o que indica o correto funcionamento do mesmo.

Lembre-se que a distribuição de processos entre os nós do cluster e a comunicação entre cada um deles, depende do bom funcionamento da sua rede. Para que o cluster possa funcionarcorretamente, é preciso ter certeza de que os nós estão "conversando" entre si, para tanto, algumas portas associadas ao serviço "mosmig" devem estar liberadas no seu firewall, portanto trate de liberá-las no script /etc/rc.d/rc.firewall (ou equivalente):

```
# vim /etc/rc.d/rc.firewall

# Permitindo acesso externo ao openMosix
iptables -A INPUT -p TCP --dport 4660\
       -j ACCEPT
iptables -A INPUT -p UDP --dport 5428\
       -j ACCEPT
```

Essa é uma política "temporária" que deve ser utilizada somente durante o processo de instalação/configuração do cluster, de modo a verificar o seu correto funcionamento. Evidentemente que, após ter certeza de que tudo está funcionando a contento, uma política mais restritiva, que permita a comunicação somente entre os nós pertencentes ao cluster é uma boa pedida. Uma subnet destinada somente para esse fim também é uma boa idéia...

O time de desenvolvedores do openMosix atualmente trabalha em uma versão para kernel 2.6, no entanto, ainda em estágio beta de desenvolvimento, consulte a seção de desenvolvimento [2] no site do projeto para maiores informações.

Existem inúmeras possibilidades de personalização e configuração de clusters openMosix. Existem também várias ferramentas gráficas interessantes de monitoramento, como o openMosix View [3], que possui vários módulos distintos que facilitam a vida de administradores de clusters e também algumas ferramentas de teste de performance, estabilidade e redundância. Como esse artigo é introdutório, não iremos abordar esses aspectos, interessantes em aplicações mais específicas que necessitem de ajustes finos e configurações mais elaboradas.

Existe a possibilidade de se utilizar os chamados "Instant openMosix Clusters" que nada mais são do que distribuições Linux com o patch do openMosix aplicados ao kernel e com ferramentas userland já instaladas. Você pode conferir alguns exemplos em [4]. Baixe a ISO, queime os CDs e "monte" o seu cluster com quantos nós quiser.

Vale lembrar que nem todo tipo de aplicação pode ser exportada entre os nós do cluster. A documentação oficial do openMosix fala de maneira superficial sobre alguns aspectos relacionados a isso [5],[6], no entanto esse com certeza é um tema interessante que complementa esse artigo e certamente deve aparecer em uma das próximas edições do slackwarezine.

Até lá... :)

Clayton Eduardo dos Santos
<claytones@terra.com.br>

Referências

[1] http://openmosix.sourceforge.net/
[2] http://openmosix.sourceforge.net/\
    development.html
[3] http://www.openmosixview.com/
[4] http://openmosix.sourceforge.net/\
    instant_openmosix_clusters.html
[5] http://howto.x-tend.be/openMosix-\
    HOWTO/x1254.html
[6] http://howto.x-tend.be/openMosix-\
    HOWTO/x1317.html

Convertendo Números de Uma Base para Outra

É comum utilizar o bc para fazer contas rápidas (e algumas não tão rápidas assim). Mas o bc é bem mais que isso, e com ele é possível também converter números entre base decimais, veja como é simples:

Decimal->Binario:
```
echo "obase=2 ; numero" | bc
```

Binario->Decimal:
```
echo "ibase=2 ; numero" | bc
```

Hexa->Binario:
```
echo "ibase=16 ; obase=2 ; numero" | bc
```

obase é base de saída (o "o" é de output) e ibase é a base de entrada (o "i" é de input). A base 10 é o "padrão" do bc, então não precisa ser indicada.

"Desentupindo" a fila de e-mails no Postfix

Sabe quando seu sistema de email tá lerdo, e você não sabe porque? vai ver o "queue" de e-mail, e tem mais de 300 emails, que estão errados ou que deram erro na hora da entrega e o MTA cisma em ficar tentando mandar? apague esse emails da fila com isso:

```
mailq | awk '/MAILER/ {
              ID = substr($1,0,10)
              system("postsuper -d "ID)
}
```

Testando a Autenticação SMTP no Console

Todo mundo que já fez um servidor de SMTP autenticado fica com um problema em mãos na hora de testar. Nem sempre existe um leitor de e-mail que suporte autenticação à mão. Como perl vem instalado em vários servidores, essa dica pode ser bastante útil.

```
perl -MMIME::Base64 -e 'print encode_base64("\000user\@lala.com.br\000senha")'
```

Lembre de substituir "user" pelo nome de um usuário para testes, o "lala.com.br" pelo domínio correto e a "senha" pela senha (dãããã).

Depois usar a saída desse comando na hora da negociação com o servidor SMTP.

```
AUTH PLAIN saidadocomando
```

Convertendo Datas de EPOCH para um Formato Humanamente Legível

Internamente os sistemas Unix-Like (e provavelmente outros) contam o tempo em segundos a partir da zero hora do dia primeiro de janeiro de 1970. A maior parte dos programas são legais e convertem um número com mais de um dezena de algarismos para um formato humano. Outros programas (alguém falou squid?) gostam de escrever direto na quantidade de segundos, o que dificulta um pouco a leitura de logs.

Se isso acontecer, um jeito fácil de descobrir uma data é usando:

```
date --date='1970-01-01 SEGUNDOS sec GMT'
```

Onde SEGUNDOS é a quantidade de segundos que o programa apresenta para você.

Autores

Clayton Eduardo dos Santos, Linux desde 2003 e com Slackware desde 2004. Atualmente desenvolve seu projeto de pesquisa de Doutorado no Departamento de Engenharia Elétrica na USP de São Carlos. É também entusiasta da era MSX e um dos editores do SlackwareZine.

Leandro Toledo, Iniciou com computadores em 1993 e Linux em 1998, usando Slackware 3.4 kernel 2.0.30. Atualmente trabalha no ramo financeiro, multinacional Fidelity BPO, como Analista de Projetos utilizando soluções livres. Seu principal projeto é o SlackwareZine.

Lindolfo Rodrigues aka Lorn, é usuário de Linux desde 2000, começou com slackware 8.0 atualmente trabalha na Uplexis Tecnonlogia, como Programador Perl, é um Monge Perl fanático ;) está no último semestre de BCC da FASP

Piter PUNK aka Roberto Freires Batista, trabalha com Linux desde 1996. Atualmente é consultor na área de sistemas operacionas na NTUX Informática. É o principal desenvolvedor do slackpkg e contribui ativamente no desenvolvimento do Slackware Linux. Também é editor do SlackwareZine e publicou diversos artigos em revistas e sites da área.

Sulamita Garcia, é Especialista Unix na EDS, e responsável pelo Linuxchix Brasil. É autora de diversos artigos e palestras sobre Slackware, Alta Disponibilidade e Linux em geral.