MakeConfig
Configuração Diferenciada de Máquinas em Redes Unix Heterogêneas

1 11 11 11 1 1

Departamento de Ciência da Computação
Instituto de Matemática, Estatística e Ciência da Computação
Universidade Estadual de Campinas
CEP: 13081-970 Campinas, SP, Brasil
Tel: +55-192-39-8442/3115 FAX: +55-192-39-7470
e-mail: {..|.}@dcc.unicamp.br

Sumário

Manter consistência entre arquivos de configuração de máquinas componentes de uma rede Unix heterogênea é um problema bastante complexo. É proposta uma maneira alternativa de configurar as diversas máquinas utilizando somente ferramentas padrão, como make, *Bourne Shell*, awk, sed e cpp (pré-processador C), disponíveis em qualquer sistema Unix. Desta forma, tanto a configuração local quanto a atualização constante independem da existência de qualquer suporte adicional de mais alto nível. A solução complementa o uso de NIS, que permite apenas distribuição de arquivos idênticos, e suplanta com vantagens o programa rdist, por ser muito mais flexível. Apresentam-se algumas diferenças entre esta ferramenta e o sistema GNU cfengine e outros construídos com os mesmos propósitos.

Abstract

Keeping consistency among configuration files of machines on a heterogeneous Unix network is a complex task. An alternative way of configuring a collection of machines is proposed, one that uses only standard tools available on any Unix system, such as make, Bourne Shell, awk, sed and cpp (C pre-processor). This way, neither local configuration nor regular updating depend on the existence of any additional higher level support. The proposed solution complements NIS*, which only distributes identical files, and goes well beyond rdist, thanks to its flexibility. It is also shown some differences between this tool and the* GNU *cfengine, and also some other systems devised for this purpose.*

1 Introdução

Entende-se por configuração de uma máquina o estabelecimento de conteúdos para repositórios (tipicamente arquivos) que definem, para essa máquina, os usuários autorizados a utilizá-la, a rede à qual pertence, sua identificação dentro da rede, as máquinas com as quais pode se comunicar, os recursos que provê — e a quem estão disponíveis — e os que utiliza de outrem, convenções estabelecidas, etc. Configurar localmente uma máquina significa tomá-la de um estado qualquer — com o sistema recém-instalado ou com arquivos de configuração parcialmente consistentes — e criá-los ou alterá-los, de modo a atingir a configuração desejada.

Manter diversas máquinas de uma rede consistentemente configuradas é algo desejável, mas nem sempre simples. A complexidade dessa tarefa aumenta consideravelmente se essas máquinas executam sistemas operacionais diferentes. Mesmo que se restrinja o escopo a Unix, as diferenças de uma implementação para outra ainda são muito grandes. Para tentar simplificar a vida de administradores, foram criados o serviço de banco de dados *Network Information Service*[17, 18], o programa rdist[7, 19], a linguagem cfengine[4, 5], os sistemas GeNUAdmin[10], Config[16] e lcfg[2, 3], entre outros.

A seguir, cada um deles é descrito sucintamente. Apontam-se algumas de suas restrições, que fornecem uma lista de requisitos adicionais para um sistema de configuração de redes heterogêneas. Em seguida, uma maneira alternativa de atender a esses requisitos é proposta, acompanhada de diversos exemplos, e comparada aos outros sistemas.

1.1 N*etwork* I*nformation* S*ervice*

NIS é um serviço de banco de dados distribuído que fornece dados administrativos para conjuntos de máquinas de uma rede.

Um domínio NIS é um conjunto de máquinas que compartilham a mesma base de dados. Cada máquina, num dado intervalo de tempo, pertence a um único domínio, e obtém informações consultando uma das servidoras NIS desse domínio. Escolhe-se uma dessas máquinas como servidora principal (*master*), e zero ou mais delas como servidoras secundárias (*slave*).

Para que alterações feitas num arquivo de configuração, mantido na servidora principal, passem a vigorar, é necessário transformá-lo em tabelas (*maps*) NIS, no formato dbm, acionando um comando que, além da transformação, ainda as propaga aos servidores secundários.

Arquivos freqüentemente distribuídos utilizando esse serviço são os que definem usuários e senhas, grupos, endereços IP, números *ethernet*, fuso horário, *mail aliases*, parâmetros de *boot*, configuração de *automount*, nomes e *masks* de sub-redes e grupos de máquinas.

1.2 O utilitário rdist

O programa rdist serve, basicamente, para manter arquivos iguais entre diferentes máquinas. Para tanto, cria-se um arquivo de entrada para esse programa na máquina servidora, especificando quais são os arquivos, as máquinas que devem recebê-los, e os diretórios onde eles serão armazenados em cada uma delas. Podem-se ainda determinar comandos para serem executados imediatamente após a cópia de alguns arquivos, na máquina que os recebeu.

Alguns pontos fracos dessa abordagem são:

- necessidade de configuração prévia da máquina que vai receber as cópias — ela deve ser capaz de aceitar a conexão do rdist, e para isso, precisa de uma entrada no arquivo .rhosts ou hosts.equiv que permita a conexão;

- necessidade de alteração do arquivo de configuração do sistema quando da adição ou remoção de máquinas da rede;
- dificuldade para especialização de configurações ligeiramente diferentes;
- atraso da propagação dos arquivos em caso de falha de uma ou mais máquinas — o sistema espera um *time-out* da tentativa de conexão.

A versão 6 desse programa, apresentada em [7], resolve parte desses problemas. O fato de não ser possível contar com seus recursos em grande parte das plataformas alvo, porém, faz com que não se possa considerar sua utilização como ferramenta de configuração heterogênea.

Poder-se-ia pensar que a existência de rdist torna o NIS obsoleto, mas isso não é verdade. Usar NIS não exige que cada máquina tenha uma cópia dos arquivos: as informações são obtidas via rede, sob demanda, a partir de arquivos otimizados para busca rápida. Isso pode ser mais eficiente do que ter de processar arquivos texto localmente.

1.3 O sistema cfengine

Criado por Mark Burgess na Faculdade de Engenharia de Oslo, Noruega, há cerca de três anos, foi recentemente incorporado ao projeto GNU, ligado à *Free Software Foundation*. É uma linguagem voltada à configuração de sistemas Unix, com primitivas para configuração de interfaces de rede, manutenção de hierarquias de *links* simbólicos, edição (básica) e remoção de arquivos, verificação e alteração de permissões de acesso, configuração de sistemas de arquivos obtidos via NFS[17] e execução de *scripts* e comandos.

A decisão por executar ou não uma ação é tomada com base em uma expressão envolvendo classes — que podem ser compreendidas como variáveis booleanas, que assumem valor verdadeiro caso a máquina em que o programa está sendo executado pertença à classe. Há classes pré-definidas, como versão de sistema operacional e nome da máquina, sendo que classes adicionais podem ser criadas, conforme necessário.

O principal inconveniente dessa solução é que o interpretador cfengine, que não é um *script*, mas um programa compilado, deve estar disponível na máquina que se deseja configurar, o que requer intervenção manual para obtê-lo de outra máquina semelhante ou para compilá-lo.

1.4 O sistema GeNUAdmin

Magnus Harlander sugere a manutenção, num servidor central, de bancos de dados a respeito das máquinas. Nesse servidor, *scripts* perl são utilizados para interpretar esses bancos de dados, verificar sua consistência, e gerar arquivos de configuração para determinadas máquinas. Apenas a servidora necessita um interpretador perl, para as demais, há scripts em sh que fazem o pouco trabalho local.

Cada arquivo de banco de dados cria um hierarquia de configurações, estruturada como uma árvore, em que a raiz especifica valores *default* para determinados atributos, seus descendentes podem determinar valores diferentes para alguns atributos, e assim sucessivamente, até as folhas, que representam máquinas ou grupos de máquinas.

Para cada um desses arquivos, é necessário um *script* especial em perl, que busca o valor dos atributos para uma determinada configuração, gerando o arquivo de configuração. Apesar de haver vários desses *scripts*, oferecendo apoio à maior parte das configurações mais comuns, qualquer desvio disso exige a implementação de um novo *script*.

Integram este sistema dois programas para manutenção do arquivo de senhas de usuários, um *daemon* que executa na servidora principal, e um programa que permite aos usuários alterarem

suas senhas. A partir do banco de dados de senhas, podem ser gerados arquivos para NIS ou para configuração local.

Como o sistema usa rsh para propagar as configurações, e depende de que o diretório de configuração seja visível na máquina alvo, é necessária uma pré-configuração dessa máquina.

1.5 O sistema Config

John Rouillard e Richard Martin construíram sobre o já comentado rdist um sistema que permite administração distribuída do sistema. O diretório central de configuração é mantido sob controle do sistema CVS[6], que realiza controle de versões e permite alteração concorrente de arquivos, com restrições de acesso.

Cada diretório controlado pelo CVS pode possuir um makefile, que será acionado utilizando o GNU-Make[9]. Com isso, podem-se gerar configurações específicas para determinadas máquinas — o que não parece ser a preocupação principal dos autores.

Arquivos podem ser alterados localmente numa máquina qualquer, armazenados no repositório principal utilizando CVS e propagados utilizando um *script* perl denominado Rdist (com 'R' maiúsculo), que toma como entrada um arquivo que reúne informações específicas de cada máquina, como sistema operacional que ela executa, tipo de processador, quantidade de memória, números de rede (IP e Ethernet) e serviços oferecidos, gerando como saída macros em formato compreendido pelo rdist tradicional. Essas macros completam o arquivo de entrada do rdist, escrito pelo administrador, para distribuir a configuração aos clientes.

Valem para esse sistema os mesmo comentários já feitos sobre o rdist.

1.6 O sistema lcfg

Paul Anderson propõe um sistema que armazena informações sobre configuração das máquinas num banco de dados central — atualmente implementado como uma tabela NIS.

São definidas classes de alto nível, cada classe possuindo atributos próprios, cujos valores são definidos individualmente para cada máquina. Justamente por a configuração ser em alto nível, é necessário escrever um *script* especial, que utiliza serviços de um *daemon* chamado *Object Manager* para iniciar serviços especiais, atualizar arquivos de configuração do sistema, atualizar pacotes locais, instalar *patches*, etc.

Esses *scripts* são executados no momento do *boot* da máquina, ou periodicamente, utilizando o comando cron. Há dezenas deles já prontos, aparentemente direcionados para o sistema Solaris.

A principal desvantagem do sistema é a necessidade de o *Object Manager* estar presente em cada máquina. Não se nota preocupação com coexistência de múltiplos sistemas operacionais.

2 Requisitos

Parece-nos que um sistema ideal para configuração de máquinas deva utilizar apenas ferramentas padrão — awk[1, 8, 15, 14, 12], sed[8, 15, 14, 12, 13], make[13, 9], cpp[11], sh[12, 15, 13, 14] —, disponíveis em qualquer instalação Unix, pois deve ser capaz de partir de uma máquina recém-instalada para um estado de acordo com o especificado na configuração centralizada.

Fica patente, portanto, que uma máquina com seu sistema operacional recém-instalado necessita galgar passos no sentido de obter mais inteligência local. Em outras palavras, o processo de configuração local implica uma mudança do estado da máquina para um nível mais alto de abstração, do ponto de vista do administrador.

Usuários típicos não desejam ter de saber que máquina serve determinado sistema de arquivos ou impressora, qual é a quantidade exata de memória real e virtual que a máquina possui, que versão do sistema operacional ela executa, enfim, desejam poder utilizar da mesma maneira, na medida do possível, qualquer máquina da rede.

Oferecer a transparência desejada pelos usuários deve ser o objetivo de toda a configuração do sistema: quanto melhor a configuração, mais o sistema se aproxima de um ambiente virtualmente homogêneo, em que diferenças locais passam tão despercebidas quanto possível.

O servidor principal, no qual reside a especificação da configuração centralizada, deve ser passivo, para que não sejam necessárias grandes alterações a fim de configurar novas máquinas, e para que máquinas fora de operação não atrapalhem o processo. Isso também facilita a manutenção de servidores secundários de configuração, que podem ser utilizados em caso de falha do principal.

É imprescindível que arquivos possam ter diferenças de uma máquina para outra. É também necessário que alguns arquivos de configuração possam ser utilizados em apenas alguns sistemas ou máquinas.

3 A proposta

A idéia é utilizar o recurso mais primário para atualização de arquivos disponível em Unix: o programa make. Essa ferramenta utiliza um arquivo de entrada (makefile) com regras que definem maneiras de construir ou atualizar arquivos a partir de outros. É normalmente utilizada para automatizar a compilação de pacotes, pois definem-se regras para gerar arquivos objetos a partir dos fontes, depois executáveis a partir de objetos e bibliotecas.

Uma descrição simplista e incompleta mas suficiente de um makefile: regras começam com o nome do arquivo que se deseja criar ou atualizar, seguido por dois pontos (:) e uma lista dos arquivos dos quais ele depende. Caso o arquivo destino não exista ou tenha data mais antiga que qualquer das dependências, os comandos que seguem a regra são executados.

Podem-se também definir variáveis num makefile, com a sintaxe nome da variável, sinal de igual (=), e conteúdo da variável. Para recuperar esse conteúdo, posteriormente, o nome da variável deve vir entre parênteses ou chaves, precedidos por um cifrão, por exemplo, `${VAR}`.

Para acompanhar a apresentação da proposta, suponha que exista um diretório exportado para toda a rede, possivelmente somente para leitura, em que se encontra um makefile, e que esse diretório seja visível como /network/local/etc na máquina que se deseja configurar.

O makefile contém diversos *targets*, como `all`, `daily`, `hourly`, etc. Esses *targets* são acionados utilizando o comando cron, de hora em hora, uma vez por dia, etc, propagando alterações na configuração central. Pode-se ainda adicionar uma invocação ao makefile no momento do *boot*.

No mesmo diretório em que se criou o makefile, armazenam-se arquivos preparados para configurar as máquinas, por exemplo, domainname, resolv.conf, shells, hosts.equiv, enfim, arquivos que terão exatamente o mesmo conteúdo em todas as máquinas. Para esses, basta definir regras que simplesmente copiem arquivos do diretório de configuração para o diretório /etc. Opcionalmente, pode-se guardar uma cópia do seu antigo conteúdo, caso exista. Convém, após criar um arquivo novo, que suas permissões sejam corrigidas, para evitar surpresas desagradáveis. Um exemplo de configuração por cópia literal pode ser encontrado na figura 1.

É conveniente que todas as referências ao diretório de configuração sejam feitas através de uma macro, como na figura 1, para evitar que, ao se decidir alterar o *mount-point* do diretório, tenha-se de alterar todo o makefile. Isso também torna possível configurar uma máquina logo após a instalação do sistema, quando quase nada é acessível. Basta obter manualmente o diretório de

Figura 1: Arquivos configurados por simples cópia

```
DIR=/network/local/etc

daily: /etc/domainname
/etc/domainname: $(DIR)/domainname
    @-mv -f $@ $@.orig
    @ cp $? $@
    @ chmod 644 $@
daily: /etc/resolv.conf
/etc/resolv.conf: ...
```

Figura 2: Configuração do arquivo motd

```
hourly: /etc/motd
/etc/motd: $(DIR)/motd
# preservando o arquivo original
    @-mv $@ $@.orig
    @ (head -1 $@.orig; cat $?) >$@
# ou descartando-o
#       read firstline <$@; (echo "$$firstline"; cat $?) >$@
    @ chmod 644 $@
```

configuração (NFS *mount* ou cópia do conteúdo para disco local) e especificar sua localização provisória na invocação do comando make, se diferente da que consta no makefile.

O conteúdo do arquivo motd aparece no momento do *login*, e costuma ser utilizado pelos administradores para informar usuários sobre alterações no sistema, paradas programadas, etc. O problema de simplesmente copiar esse arquivo de uma configuração central é que, em geral, a primeira linha dele é gerada pelo sistema, no momento da instalação ou do *boot*, e seria conveniente preservá-la. Os comandos não ficam muito mais complicados, como mostra a figura 2. Caso não se quisesse guardar o conteúdo do arquivo original, bastaria adicionar e remover marcas de comentário como indicado no makefile.

Um caso um pouco mais complicado é o arquivo printcap, utilizado para definir as impressoras disponíveis. Seu conteúdo não pode ser o mesmo em todas as máquinas, já que ao menos uma delas é a servidora da impressora (se ela não estiver conectada diretamente à rede), e requer muito mais informação de configuração que as demais. Decidiu-se, nesse caso, utilizar o cpp — mas poder-se-ia ter utilizado um *script* em awk — fornecendo-lhe um parâmetro que permitisse decidir qual a máquina em questão, como mostra a figura 3. Observe que o comando mkdir cria os diretórios necessários para os quais há referências no arquivo de configuração gerado. Para que a máquina parana exporte a impressora ibm1 às demais, o arquivo printcap deve ser semelhante ao da figura 4.

Esse mesmo mecanismo poderia ser utilizado para obter o arquivo crontab, em que se definem comandos que devem ser executados periodicamente. No entanto, sistemas operacionais distintos em geral sugerem comandos diferentes nesse arquivo. Portanto, para processá-lo, seria interessante ter alguma informação sobre o sistema operacional sendo executado, não apenas o nome da máquina. Se um trecho do makefile for semelhante ao da figura 5, o arquivo crontab

Figura 3: Configuração do arquivo printcap

```
CPP = /usr/lib/cpp
hourly: /etc/printcap
/etc/printcap: $(DIR)/printcap
    @ -mv $@ $@.orig
    @ $(CPP) -P -DHOST_`uname -n` $? >$@
    @ mkdir -p `grep sd= $@ | \
                sed -e 's/^.*sd=//g' -e 's/:.*$//g'`
```

Figura 4: Arquivo printcap, a ser pré-processado com cpp

```
#ifdef HOST_parana
ibm1|ibm1|IBM Lexmarq 10 on parana:\
    :mx#0:sh:\
    :br#9600:lf=/var/adm/ibm1-errs:lo=ibm1.lock:\
    :lp=/dev/bpp0:sd=/var/spool/printers/ibm1:
#else
ibm1|ibm1|IBM Lexmarq 10 on parana:\
    :lp=:rp=ibm1:rm=parana:sd=/var/spool/printers/ibm1:
#endif
```

Figura 5: Configuração do arquivo crontab

```
HOSTINFO += -DHOST_`uname -n`
HOSTINFO += -DOSR_`uname -s`=`uname -r | \
            sed -e 's/[^0-9]//g' -e s/$$/00000/ \
                -e 's/\([0-9][0-9][0-9][0-9][0-9]\).*/\1/'`
HOSTINFO += -DOS_`uname -s``uname -r | sed 's/\./_/g'`

NEWDIR = $(DIR)
hourly : /var/spool/cron/crontabs/root
/var/spool/cron/crontabs/root : $(DIR)/crontab
    @ $(CPP) -P -DDIR=$(NEWDIR) $(HOSTINFO) $? | \
      egrep . | sort +1n +0n | crontab `basename $@`
```

Figura 6: Trecho do arquivo crontab

```
0 * * * * sh DIR/update hourly
0 0 * * * sh DIR/update daily
#ifndef HOST_tiete
35 3 * * * rdate tiete >/dev/null
#endif
```

Figura 7: *Script* update

```
#!/bin/sh
# usage: <DIR>/update [NEWDIR=<NEWDIR>] [<TARGET>] ...
# DIR is where the configuration directory is now
# NEWDIR is where it will be in the future (default=DIR)
# TARGET is the name of a target in <DIR>/makefile
DIR=‘dirname $0‘
case ‘uname -sr‘ in
 "SunOS 5"*) MAKE=/usr/ccs/bin/make; CPP=/usr/ccs/lib/cpp;;
 *) MAKE=make; CPP=/usr/lib/cpp;;
esac
exec ${MAKE} -f $DIR/makefile \
     CPP=${CPP} DIR=${DIR} "$@"
```

poderá fazer testes não apenas testando se HOST_parana está definido, mas também se o sistema é IRIX (OS_IRIX) ou se é Solaris 2 (OSR_SunOS >= 50000), tomando o cuidado de só efetuar esse teste após verificar que OSR_SunOS está definido, como será mostrado na figura 9. Note que os comandos egrep e sort são apenas cosméticos, para remover linhas em branco e ordenar as entradas por hora e minuto. A variável NEWDIR é bastante conveniente para a primeira configuração de uma máquina, quando a configuração é visível num determinado diretório, mas, em reconfigurações futuras, será acessível num diretório diferente. Com o trecho do makefile dado, uma parte do arquivo crontab poderá ser a figura 6.

O *script* update é o responsável por acionar, a cada dia e a cada hora, o makefile, com o *target* indicado. Para fazer isso, sugere-se que update seja semelhante à figura 7. Na primeira configuração da máquina, deve-se tomar o cuidado de incluir um parâmetro da forma NEWDIR=/nome/do/diretório, para indicar o diretório a ser utilizado para configurações futuras. Por exemplo, se o diretório central foi manualmente copiado para /tmp/config, mas será futuramente obtido via NFS como /network/local/etc, o comando a acionar, conforme comentário no próprio *script*, é /tmp/config/update NEWDIR=/network/local/etc pois a definição de NEWDIR na linha de comando faz com que a redefinição no makefile seja ignorada, ou seja, o valor da variável DIR, obtido da própria linha de comando (/tmp/config), não será utilizado no processamento do arquivo crontab, prevalecendo a definição /network/local/etc.

A estrutura case serve, no caso, para definir parâmetros específicos de cada sistema operacional, como a localização dos comandos make e cpp, e de outros que se julgarem necessários. Quando arquivos residem em diretórios diferentes, ou têm nomes diferentes de um sistema para outro, pode-se utilizar o mesmo artifício.

O arquivo /etc/exports, por exemplo, no Solaris se chama /etc/dfs/dfstab, e tem um for-

Figura 8: Configuração do arquivo exports

```
hourly : $(EXPORTS)
$(EXPORTS) : $(DIR)/exports
    @-mv $@ $@.orig
    @ sed -e 's/, */","/g' -e 's/","/,/' $? | \
      $(CPP) -P $(HOSTINFO) - | \
      egrep . | sed 's/","/,/g' >$@
    @ chmod 644 $@
    @ $(EXPORTALL)
```

Figura 9: Exemplo de arquivo exports

```
#if defined(OSR_SunOS) && OSR_SunOS >= 50000
#define EXPORT(dir,parm) share -F nfs -o parm dir
#define access rw
#else
#define EXPORT(dir,parm) dir -parm
#endif

#define SECURE parana:atibaia

#ifdef HOST_parana
EXPORT(/opt/config, ro=allhosts)
#endif

#ifdef HOST_atibaia
EXPORT(/home/staff, access=allhosts,root=SECURE)
EXPORT(/usr, ro=allhosts,root=SECURE)
#endif
```

mato completamente diferente. O comando para colocá-lo em vigor, em SunOS e IRIX, é /usr/etc/exportfs -a, em AIX é /usr/sbin/exportfs -a, em Solaris é /usr/sbin/shareall... Suponha então que tenham sido adicionados ao arquivo update as definições para EXPORTS e EXPORTALL, indicando, respectivamente, o nome do arquivo e o comando. Suponha ainda que nesse mesmo *script* é dada a definição da variável HOSTINFO — aquela que define os parâmetros que devem ser passados ao cpp, com informações sobre nome da máquina e versão do sistema operacional. O trecho do makefile responsável pela atualização do arquivo exports pode ser algo semelhante à figura 8. Já o arquivo exports pode ter um cabeçalho inteligente, para que todas as declarações tenham o mesmo formato, como mostra a figura 9. Os dois comandos sed no makefile tomam o cuidado de remover espaços após as vírgulas e de impedir que todas elas, com exceção da primeira, fossem interpretadas pelo cpp como separadores de argumentos, o que permite uma maior liberdade para escrever o arquivo de configuração.

Falta ainda uma maneira de evitar que alguns arquivos sejam utilizados em certos sistemas ou máquinas, e de permitir que determinadas máquinas usem arquivos especiais para sua configuração. Para isso, sugere-se que o próprio makefile seja pré-processado. O *script* update

pode se encarregar disso. A nova versão desse arquivo escontra-se na figura 10. Um pouco mais elaborado, esse exemplo permite que cada sistema escolha o conjunto de macros que devem ser utilizadas na invocação do make, bastando para isso atribuir-lhes um valor e adicionar seu nome à variável da *shell* `VARS`. Recomenda-se que o makefile pré-processado seja armazenado num diretório seguro: /tmp seria uma escolha ruim, pois, com alguma sorte e perícia, seria possível obter acesso privilegiado.

Não é necessário ir muito longe para encontrar exemplos em que o uso de um makefile não é exatamente conveniente. Um caso particular é para manutenção de *links* simbólicos. Como o make não olha para o momento da criação do *link*, mas sim do arquivo ao qual ele se refere — ao menos nos sistemas estudados —, não há uma forma de definir uma estrutura de *links* sem utilizar arquivos adicionais. Supondo que o número de *links* num sistema é pequeno, e sabendo que as operações de criação e remoção de *links* são baratas, pode-se definir um arquivo *links* no diretório de configuração central, com linhas semelhantes à figura 11. Por simplicidade, vamos supor que esse arquivo não precisa ser pré-processado. A regra do makefile responsável pela manutenção dos *links* é dada na figura 12. Não haveria problemas em se utilizar o diretório /tmp dessa vez, pois o arquivo é apenas para marcar o momento da última atualização. Funcionaria bem, a não ser que algum usuário tivesse a infeliz idéia de criar um diretório com esse nome.

Outro caso em que se deve ter atenção especial é após a instalação do sistema. Alguns arquivos de configuração locais são alterados durante a instalação, e portanto ganham datas de atualização mais recentes que dos arquivos no diretório central. Para forçar sua reconstrução, utiliza-se um artifício do make: dependências alternativas. Se uma regra tiver dois sinais de doins pontos juntos (::), ao invés de apenas um, podem ser definidas outras regras para o mesmo arquivo alvo — desde que essas outras também utilizem pares de dois pontos. Caso a primeira não force a reconstrução do arquivo, a segunda é considerada, e assim sucessivamente. Não é permitido que um mesmo arquivo seja alvo de regras normais e alternativas. A figura 13 mostra uma maneira de forçar a atualização de arquivos definidos na variável `SETUPFILES`. Note que a regra que define explicitamente a regra para construção do arquivo nsswitch.conf deve receber um sinal de dois pontos a mais, já que, quando `SETUPFILES` é expandida para uma lista de arquivos, entre eles nsswitch.conf, cada um é considerado um alvo distinto, apesar de aparecerem juntos na mesma regra — é como se várias regras houvessem sido definidas, uma para cada arquivo da lista, com as mesmas dependências e os mesmos comandos de reconstrução.

4 Obtendo a configuração central

A forma mais primária de se obter o diretório que contém a configuração central do sistema é montá-lo manualmente num diretório qualquer, e utilizar o próprio sistema para alterar o arquivo /etc/fstab ou /etc/vfstab, para que, no momento do *boot*, esse diretório seja acessível em local conhecido.

Outra opção é utilizar o automount[17], definindo localmente seus arquivos de configuração ou obtendo-os através do NIS. Essa opção, porém, pode fazer com que o servidor principal seja sobrecarregado, nos momentos em que todos tentam simultaneamente obter o diretório de configuração. Para evitar esse problema, o diretório pode ser replicado para servidores secundários com auxílio de rdist, e o automount pode se encarregar de obter o servidor mais próximo.

Figura 10: *Script* update que pré-processa o makefile

```
#!/bin/sh
# usage: <DIR>/update [NEWDIR=<NEWDIR>] [<TARGET>] ...
# DIR is where the configuration directory is now
# NEWDIR is where it will be in the future (default=DIR)
# TARGET is the name of a target in <DIR>/makefile
DIR=‘dirname $0‘
HOSTINFO="
-DHOST_‘uname -n‘
-DOSR_‘uname -s‘=‘uname -r |
  sed -e ’s/[^0-9]//g’ -e s/$/00000/ \
      -e ’s/\([0-9][0-9][0-9][0-9][0-9]\).*/\1/’‘
-DOS_‘uname -s‘‘uname -r | sed ’s/\./_/g’‘
"
MAKE=make
CPP=/usr/lib/make
EXPORTS=/etc/exports
EXPORTALL="exportfs -a"
MAKEFILE=/etc/makefile.conf
VARS="DIR HOSTINFO MAKE CPP EXPORTS EXPORTALL"
case ‘uname -sr‘ in
 "SunOS 5"*) MAKE=/usr/ccs/bin/make; CPP=/usr/ccs/lib/cpp;
             EXPORTS=/etc/dfs/dfstab; EXPORTALL=shareall;;
 "SunOS 4"*) EXPORTALL="/usr/etc/exportfs -a";;
 AIX*) EXPORTALL="/usr/sbin/exportfs -a";;
esac
${CPP} -P ${HOSTINFO} ${DIR}/makefile >${MAKEFILE} &&
chmod 600 /etc/makefile.conf &&
eval ’exec ${MAKE} -k -f /etc/makefile.conf’ \
        ‘for var in $VARS; do \
           eval echo ’$var=\"${’$var’}\"’; \
         done‘ \
     ’"$@"’
```

Figura 11: Arquivo de configuração de *links*

```
/usr/X /usr/local/X11R6
/usr/include/X11 /usr/X/include
/usr/TeX /usr/local/TeX-3.1415
```

Figura 12: Configuração de *links*

```
/etc/.links.time : $(DIR)/links
    cat $? | \
    while read to from; do \
      rm $$to && ln -s $$from $$to \
    done && \
    touch $@ && chmod 000 $@
```

Figura 13: Forçando a configuração inicial

```
#if defined(OSR_SunOS) && OSR_SunOS >= 50000
#define OS_Solaris
#endif

RERUN = $(DIR)/update NEWDIR=$(NEWDIR)

all : daily hourly

# ...

#ifdef OS_Solaris
daily : /etc/nsswitch.conf
# *** primeira alternativa ***
/etc/nsswitch.conf :: $(DIR)/nsswitch.conf
    @-mv $@ $@.orig
    @ cp $? $@
    @ chmod 644 $@
#endif

SETUPFILES = /etc/motd /etc/passwd
SETUPFILES += /var/spool/cron/crontabs/root $(EXPORTS)

#if defined(OS_Solaris)
SETUPFILES += /etc/shadow /etc/default/cron /etc/system
SETUPFILES += /etc/nsswitch.conf /etc/default/login
#endif

# *** segunda alternativa ***
$(SETUPFILES) :: /etc/.setup.time
    mv $@ $@.orig
    @ $(RERUN) SETUPFILES=/dev/null $@

/etc/.setup.time :
    touch $@
```

5 Conclusões

A proposta apresentada neste texto não é um sistema pronto para permitir configuração automática, é apenas uma sugestão de como proceder para enfrentar essa dificuldade.

Seu maior inconveniente é a complexidade que pode surgir para expressar algumas configurações específicas, que requeiram processamentos mais sofisticados, assim exigindo maior habilidade de programação em sh e utilitários básicos. Nesse caso, a proposta pode ser utilizada para configurar apenas alguns arquivos básicos, preparando o terreno para que as outras ferramentas, mais poderosas, mas com pré-requisitos mais exigentes, possam ser utilizadas.

Agradecimentos

A Islene Calciolari Garcia, pelo intenso acompanhamento e pelo grande número de críticas e sugestões no desenvolvimento desse texto.

Ao Serviço de Apoio ao Estudante da Unicamp, pelos três anos de bolsa de iniciação científica, de maio de 1992 a abril de 1995, oferecidos ao então graduando Alexandre Oliva, sob orientação de Paulo Lício de Geus.

À FAPESP — Fundação de Amparo à Pesquisa do Estado de São Paulo —, pela bolsa de mestrado, que vem permitindo ao atual mestrando Alexandre Oliva prosseguir, paralelamente às suas atividades de estudante, o desenvolvimento deste e de outros projetos iniciados anteriormente.

Referências

[1] A. Aho, B. Kernighan, and P. Weinberger. *The AWK Programming Language*. Addison-Wesley, 1988.

[2] P. Anderson. Local system configuration for syssies. CS-TN-38 Version 1.12.

[3] P. Anderson. Towards a high-level machine configuration system. In *USENIX Systems Administration (LISA VIII) Conference* [20], pages 19–26.

[4] M. Burgess. Cfengine: a site configuration engine. *USENIX Computing systems*, 8(3), 1995.

[5] M. Burgess. *Cfengine: host configuration manager*. Oslo College, Faculty of Engineering, Oslo, Norway, Jan. 1996.

[6] P. Cederqvist. *GNU CVS Manual*. Signum Support AB, Box 2044, S-580 02, Linkoping, Sweden, Oct. 1993.

[7] M. A. Cooper. Overhauling rdist for the '90s. In *USENIX Systems Administration (LISA VI) Conference*, pages 175–188, Long Beach, CA, USA, Oct. 1992. USENIX Association.

[8] D. Dougherty. *sed & awk*. O'Reilly & Associates, 1992.

[9] Free Software Foundation. *GNU Make Manual*, 0.48 edition, Apr. 1995.

[10] M. Harlander. Central system administration in a heterogeneous unix environment: GeNUAdmin. In *USENIX Systems Administration (LISA VIII) Conference* [20], pages 1–8.

[11] B. Kernighan and D. Richie. *The C Programming Language.* Prentice Hall, second edition, 1988.

[12] S. G. Kochan and P. H. Wood. *Unix Shell Programming.* Hayden Books, Indianapolis, Indiana, USA, first edition, 1985.

[13] S. G. Kochan and P. H. Wood. *Exploring the Unix System.* Hayden Books, Indianapolis, Indiana, USA, second edition, 1989.

[14] J. Muster, P. Birns, and Lurnix. *Unix Power Utilities for Power Users.* Management Information Source, Berkeley, CA, USA, first edition, 1989.

[15] J. Peek, T. O'Reilly, and M. Loukides. *Unix Power Tools.* O'Reilly & Associates and Random House, Sebastopol, CA and New York, NY, USA, first edition, Mar. 1993.

[16] J. P. Rouillard and R. B. Martin. Config: A mechanism for installing and tracking system configurations. In *USENIX Systems Administration (LISA VIII) Conference* [20], pages 9–17.

[17] H. Stern. *Managing NFS and NIS.* O'Reilly & Associates, Sebastopol, CA, USA, second edition, Apr. 1992.

[18] SunSoft, Sun Microsystems, Inc., Mountain View, California. *Naming Services Transition Kit 1.2, Administrator's Guide*, Sept. 1995.

[19] SunSoft, Sun Microsystems, Inc. rdist: remote file distribution program. Manual Page, 1995.

[20] USENIX Association. *USENIX Systems Administration (LISA VIII) Conference*, San Diego, California, USA, Sept. 1994.