INSTALANDO O *LAM-MPI* NO DEBIAN

Brivaldo Junior

25 de Março de 2006

1 Introdução

A comunicação de processos por troca de mensagens é muito utilizada em programação paralela, especialmente em máquinas parelelas com memória distribuída. O MPI (Message-Passing Interface) é uma tentativa de padronização do paradigma da troca de mensagens, que foi sugerida por um grupo de trabalho formado por pessoas da indústria, governo e universidades.

O principal objetivo do MPI é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas que utilizem troca de mensagens. Além de garantir a portabilidade dos programas paralelos, essa interface deve ser implementada eficientemente nos diversos tipos de máquinas paralelas existentes (reais ou clusters de workstations).

O MPI é uma biblioteca com funções para troca de mensagens, responsável pela comunicação e sincronização de processos. Dessa forma, os processos de um programa paralelo podem ser escritos em uma linguagem de programação seqüencial, tal como C ou Fortran.

A implementação LAM (Local Area Multicomputer) do MPI provê um ambiente de programação e desenvolvimento de sistemas para computadores heterogêneos numa rede local. Dessa forma, podemos usar uma rede local para simular um computador paralelo.

Numa breve descrição, podemos dizer que o LAM-MPI inicia daemons remotamente nos computadores da rede local, os quais vão participar da simulação da "máquina paralela". Esses daemons permanecem "adormecidos"até que recebam uma mensagem para carregar um programa MPI para executar.

2 Como Realizar a Instalação

A primeira coisa que devemos fazer é instalar os pacotes do LAM-MPI e vamos com isso habilitar o uso de 2 linguagens de programação, o C/C++ e o Python. A lista de pacotes se encontra na tabela abaixo:

lam-mpidoc	Documentation for the Message Passing Interface standard
lam-runtime	LAM runtime environment for executing parallel programs
lam4	Shared libraries used by LAM parallel programs
lam4-dev	Development of parallel programs using LAM
lampython	MPI-enhanced Python interpreter (LAM based version)
mpichpython	MPI-enhanced Python interpreter (MPICH based version)

2.1 Instalação do LAM-MPI

Agora como `root` execute o comando:

```
apt-get install lam-mpidoc lam-runtime lam4 lam4-dev lampython
```

Mas isso não é o suficiente para nosso computador paralelo funcionar, é necessário também que um programa chamado *rsh* esteja ativo e configurado, e para isso vamos também instala-lo. A lista de pacotes se encontra na tabela abaixo:

rsh-client	rsh clients
rsh-server	rsh servers

2.2 Instalação do RSH

E também devemos instalar estes programas como `root`:

```
apt-get install rsh-server rsh-client
```

Obs: Estes passos de instalação devem ser executados em cada um dos ***nós*** do computador paralelo.

A instalação do *rsh-server* deve adicionar as linhas abaixo ao seu arquivo `inetd.conf`:

```
#:BSD: Shell, login, exec and talk are BSD protocols.
shell           stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rshd
login           stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind
exec            stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rexecd
```

Apenas para lembrar o arquivo `inetd.conf` fica localizado no diretório **`/etc`**.

Ainda não acabamos a fase de ajustes e configurações! Temos que definir em cada ***nó*** do computador paralelo que os outros ***nós*** podem acessa-lo.

Vamos editar o arquivo `/etc/hosts.equiv` e inserir dentro dele os *hosts* que farão parte do computador paralelo.

O arquivo deve ficar desta forma:

```
# /etc/hosts.equiv: list  of  hosts  and  users  that are granted "trusted" r
#                   command access to your system .
host1
host2
...
```

3 Testando tudo antes

3.1 Testando o Alcance aos Nós

Antes de partimos para programação em si, vamos realizar alguns testes que vão nos garantir a conectividade do nosso computador paralelo. O **primeiro teste** é saber se o *host* que faz parte do ***nó*** do computador paralelo esta *alcançável* dentro da rede, e podemos fazer isto de diversas formas, uma delas é disparando uma cadeia de pacotes ICMP chamados de *ping's*. Execute o comando *ping* para os *hosts* destino. Por exemplo:

```
brivaldo@bravo:~$ ping -c 3 192.168.200.50
PING 192.168.200.50 (192.168.200.50) 56(84) bytes of data.
64 bytes from 192.168.200.50: icmp_seq=1 ttl=64 time=0.305 ms
64 bytes from 192.168.200.50: icmp_seq=2 ttl=64 time=0.316 ms
64 bytes from 192.168.200.50: icmp_seq=3 ttl=64 time=0.289 ms

--- 192.168.200.50 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.289/0.303/0.316/0.018 ms
```

No caso, o *host* alvo tinha o IP 192.168.200.50 e não tivemos nenhum **packet loss**, isso quer dizer que nosso *host* que será um ***nó*** do computador paralelo esta *alcançável*.

3.2 Testando a Execução de Comandos Remotos

Agora devemos proceder com um **segundo teste**, que é o de saber se o *host* recebe comandos remotos requisitados pelo comando *rsh* estão sendo executados corretamente, para isso vamos executar o seguinte comando como usuário comum:

```
rsh <host> <comando>
```

Por exemplo, no computador paralelo que estamos exemplificando:

```
brivaldo@bravo:~$ rsh 192.168.200.50 w
23:20:16 up  2:45,  1 user,  load average: 0,54, 0,32, 0,38
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    192.168.200.250  21:28   23:42   0.48s  0.48s -bash
```

Onde *w* é um comando que mostra quais usuários estão logados no computador. Se por acaso você não conseguir executar um comando remotamente usando o *rsh* e obtiver a seguinte resposta do sistema:

```
brivaldo@bravo:~$ rsh 192.168.200.50 w
Permission denied.
```

Verifique se do computador que você esta executando o *rsh* esta incluso na lista dos computadores que serão ***nós*** do computador paralelo, dentro do arquivo `/etc/hosts.equiv`.

Existe ainda um teste para saber se o Debian realmente esta utilizando o programa *rsh* ou o *ssh* no seu lugar. Se estiver usando o *ssh*, provavelmente o início do **segundo teste** deve falhar.

Para sabermos se estamos utilizando o programa correto, execute como `root`:

```
bravo:~# update-alternatives --display rsh
rsh - status is auto.
link currently points to /usr/bin/netkit-rsh
/usr/bin/ssh - priority 20
slave rsh.1.gz: /usr/share/man/man1/ssh.1.gz
/usr/bin/netkit-rsh - priority 100
slave rsh.1.gz: /usr/share/man/man1/netkit-rsh.1.gz
Current 'best' version is /usr/bin/netkit-rsh.
```

Se a `Current 'best' version` for o `netkit-rsh` estamos utilizando o programa correto. Caso contrarío execute o comando:

```
bravo:~# update-alternatives --config rsh

There are 2 alternatives which provide 'rsh'.

Selection    Alternative
-----------------------------------------------
1             /usr/bin/ssh
*+2           /usr/bin/netkit-rsh

Press enter to keep the default[*], or type selection number: 2
Using '/usr/bin/netkit-rsh' to provide 'rsh'.
```

E escolha o programa responsável pelo `rsh`.

3.3 Testando a Conexão entre os Nós

Antes deste **terceiro teste** é necessário criar um arquivo com todos os ***nós*** que participaram do computador paralelo, para que estes sejam depois iniciados pelo comando `lamboot`. Depois de criado este arquivo, vamos executar o comando de testes do LAM-MPI para saber se podemos realmente inicializar o nosso computador paralelo.

Execute o comando abaixo:

```
recon -v <arquivo com hosts>
```

Vamos a um exemplo, o *nó* que vai disparar o comando tem o IP 192.168.200.250:

```
brivaldo@bravo:~\$ recon -v hosts
n-1<22082> ssi:boot:base:linear: booting n0 (192.168.200.250)
n-1<22082> ssi:boot:base:linear: booting n1 (192.168.200.50)
n-1<22082> ssi:boot:base:linear: finished
-----------------------------------------------------------------------------
Woo hoo!

recon has completed successfully.  This means that you will most likely
be able to boot LAM successfully with the "lamboot" command (but this
is not a guarantee).  See the lamboot(1) manual page for more
information on the lamboot command.

If you have problems booting LAM (with lamboot) even though recon
worked successfully, enable the "-d" option to lamboot to examine each
step of lamboot and see what fails.  Most situations where recon
succeeds and lamboot fails have to do with the hboot(1) command (that
lamboot invokes on each host in the hostfile).
-----------------------------------------------------------------------------
```

Se você receber uma mensagem como esta, quer dizer que o *recon* teve sucesso nos testes e podemos inicializar nosso computador paralelo.

4 Iniciando o Computador Paralelo

Esta é a parte mais simples, uma vez que todos os passos anteriores foram cumpridos, basta somente executarmos o comando *lamboot* como no exemplo abaixo e esperar o computador paralelo ser inicializado!

```
brivaldo@bravo:~$ lamboot -v hosts

LAM 7.1.1/MPI 2 C++/ROMIO - Indiana University

n-1<22125> ssi:boot:base:linear: booting n0 (192.168.200.250)
n-1<22125> ssi:boot:base:linear: booting n1 (192.168.200.50)
n-1<22125> ssi:boot:base:linear: finished
```

E depois de feito isto, vamos apenas nos certificar de que esta tudo certo com o comando *lamnodes*, com ele vamos poder ver que computadores estão servindo de nós para o computador paralelo.

Veja a saída do nosso exemplo:

```
brivaldo@bravo:~$  lamnodes
n0      192.168.200.250:1:origin,this_node
n1      192.168.200.50:1:
```

Podemos ver que o **n0** é o *nó* que disparou o computador paralelo e que **n1** faz parte dos *nós* ativos.

Outra forma de ver que um *nó* esta com o LAM-MPI ativo, é logar fisicamente nele e executar o comando, como root:

```
netstat -nap | grep lamd
```

Este comando vai mostrar que o *lamd* esta escutando conexões remotas.

5 Conclusão

Agora é só programarmos nossos programas e utilizarmos o computador paralelo, não pretendo aqui ficar explicando como manipular o LAM-MPI, esta tarefa fica de exercício para quem for brncar com o poder computacional do computador paralelo.

Boa sorte!