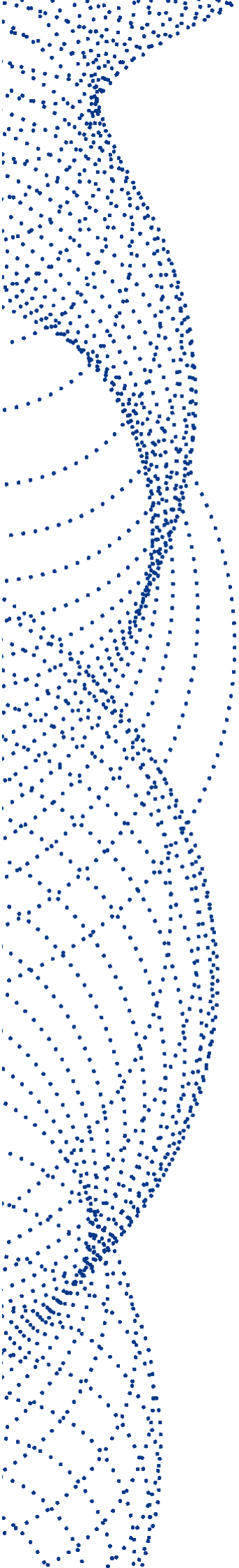


Escola de Engenharia
Universidade do Minho

Infraestruturas de Tecnologias de Informação

Projeto 3

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Professor
Henrique Santos

2021/2022

Constituição da Equipa de Trabalho
Grupo 4

Andreia Almeida

A89250

Beatriz Azevedo

A89281

João Silva

A89956

José Fernandes

A90255

Margarida Barros

A89177

Índice

Índice de Figuras

Índice de Tabelas

1.Introdução

O presente documento apresenta o projeto que se realiza no âmbito da Unidade Curricular de Infraestruturas de Tecnologias da Informação do Mestrado integrado em Engenharia e Gestão de Sistemas de Informação, lecionada pelo docente Henrique Santos, no ano letivo de 2021/2022.

O objetivo deste projeto passa por abordar a temática de Multiprocessamento através do desenho, implementação e avaliação de um sistema de multiprocessamento, baseado em cluster, e com recurso aos computadores pessoais.

Inicialmente foi realizada uma identificação dos requisitos que o sistema deverá cumprir.

Seguidamente foi abordado o conceito de Cluster e foram identificados e descritos dois tipos de clusters diferentes, o MOSIX e o Beowulf, sendo ainda exploradas as suas vantagens e desvantagens.

Posteriormente inicia-se então a fase de implementação dos clusters Beowulf e MOSIX, onde iremos começar por desenhar e descrever a arquitetura dos sistemas, seguida por uma descrição do processo de preparação e configuração das máquinas. De seguida é descrito todo o processo de implementação dos clusters e após isto é então realizada uma análise aos requisitos, verificando-se se os clusters cumprem ou não os requisitos traçados.

Por fim iremos realizar ainda uma descrição de todas as ferramentas utilizadas para a realização deste projeto.

2.Requisitos do Sistema

Este sistema deve responder aos seguintes requisitos:

Requisitos não funcionais:

- Tolerância a falhas (elevada disponibilidade): o sistema deve utilizar técnicas de tolerância a falhas quando este exige alta disponibilidade. Isto é, mesmo com a presença de falhas, é possível através dessas técnicas garantir que o sistema funcione corretamente. Estas técnicas envolvem, por exemplo mecanismos de redundância, pois permitem aumentar a confiabilidade do sistema. As várias formas de redundância de hardware, software, de tempo e de informação têm algum impacto no sistema nomeadamente, quanto ao desempenho por exemplo. As técnicas comuns são o mascaramento de falhas, deteção de falhas, localização, confinamento, recuperação, reconfiguração e tratamento.[1]
- Balanceamento da carga computacional, repartindo uma tarefa por diversos processadores disponíveis: permite a divisão de trabalho entre dois ou mais computadores, como forma de melhorar o desempenho do sistema. “Caso um dos servidores falhe, o balanceador de carga é capaz de detetar e redirecionar o fluxo para outros, sem que o serviço como um todo fique indisponível”. É possível também através deste, obter um bom nível de escalabilidade, pois podemos adicionar mais servidores de forma transparente aos utilizadores.[2]
- Partilha de recursos, permitindo a otimização na utilização dos mesmos: como existe um processador para cada máquina diferente, é possível distribuir os recursos necessários pelas várias máquinas.[1]
- Aumento do desempenho na execução de tarefas paralelizáveis: ao distribuirmos os processos pelos processadores existentes de cada máquina, irá permitir aumentar a capacidade de processamento uma vez que os processos serão executados paralelamente.[2]

[1] Taisy Silva Weber, Tolerância a falhas: conceitos e exemplos, http://www.inf.ufrgs.br/~taisy/disciplinas/textos/ConceitosDependabilidade.PDF, Acesso em 5 de Dezembro
[2] Matheus Souza Fonseca, Projeto de um balanceador de carga de servidores confiável para sistemas Linux , https://fga.unb.br/articles/0000/5580/TCC1_MatheusFonseca.pdf, Acesso em 5 de Dezembro

- Baseado em virtualização (tanto quanto o possível, para evitar o recurso a componentes dedicados): Permite a execução de vários sistemas operacionais a partir de máquinas virtuais. Este requisito também permite uma maior segurança e confiabilidade na medida em que como cada máquina virtual funciona de forma independente, qualquer problema que ocorra em uma máquina, não vai afetar a outras.[1]

3. Cluster

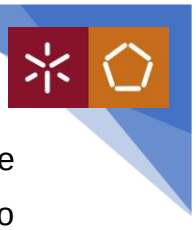
3.1. Definição de *Cluster*

Um cluster consiste num sistema que engloba dois ou mais computadores, chamados de nós, para que estes sejam executados paralelamente para cumprir o mesmo objetivo. Desta forma, é possível distribuir várias tarefas individuais paralelizáveis, entre nós do *cluster*. Para isso, é necessário conectar os nós individuais em uma rede local para ser possível a comunicação entre eles, funcionando, portanto, como um só multiprocessador. A computação em *cluster* mostra-se, desta forma, uma solução viável, uma vez que os nos podem ser constituídos por computadores simples que em conjunto formam um sistema de processamento que consegue lidar com diversas aplicações.[3]

Existem geralmente três tipos de computação em *cluster*:

- **Cluster de alta disponibilidade (*High Availability Computing Cluster)***: Oferece alta disponibilidade ao longo de um período de tempo. São sistemas que usualmente são tolerantes a falhas e resilientes, uma vez que em caso de falha, o sistema consegue continuar a fornecer um serviço pois este é concebido para lidar com a perda de um nó. Este tipo de cluster como mantem sempre um nível de redundância, permite ajudar a melhorar a confiabilidade do sistema.[1]
- ***Cluster* para Balanceamento de carga (*Load Balancing*):** Permite distribuir, de forma uniforme, as tarefas de processamento entre os nós de um cluster, de forma a otimizar o seu desempenho e evitar que um só nó receba uma grande quantidade de trabalho desequilibrada.[4]
- ***Cluster* de alto desempenho (*High Performance Computing Cluster*)**: Este tipo de cluster pode alcançar altos níveis de desempenho comparativamente com uma só máquina, isto porque estes não são

[3]Aaron Nordhoff, O que é um cluster? Uma visão geral do clustering na nuvem, https://www.capitalone.com/tech/cloud/what-is-a-cluster/, Acesso em 5 de dezembro.
[4] Emerson Alecrim, Cluster: conceito e características, https://www.infowester.com/cluster.php, Acesso em 5 de dezembro.



restritos por um definido número de núcleos de processador. Permite que o processamento de aplicações forneça bons resultados no menor tempo possível.[5]

3.2. Diferentes *Clusters*

3.2.1. Beowulf

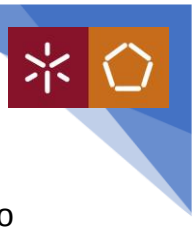
Um cluster Beowulf caracteriza-se por ter um processamento em larga escala e que reúne um conjunto de computadores conectados em rede para criar um supercomputador virtual via processamento paralelo. Para os clusters Beowulf, é essencial existir um servidor que seja responsável por controlar tudo o que está relacionado com o cluster.[6]

É necessário atender a alguns requisitos, nomeadamente, os nós serem utilizados somente para computação de alto desempenho e o sistema operacional deve ser de código aberto, sendo esta uma das razões para a utilização do SO Linux. Requer também o uso de uma biblioteca de mensagens como PVM ou MPI ou o uso de múltiplos processos com o Mosix.[7]

Tabela 1 - Vantagens e Desvantagens do Beowulf

Vantagens	Desvantagens
• Manutenção facilitada;[4] • Redução do número de problemas causados pela instalação de pacotes desnecessários;[4] • Menores custos das máquinas e de manutenção;[4] • Sistema operacional "open-source"[4]	• Uso limitado, por parte da máquina, ao processamento definido pelo servidor;[4] • Poucos softwares que suportem o Cluster Beowulf como um sistema único.[5] • Balanceamento de carga efetuado de forma manual.[5]

[5]Emerson Alecrim, Cluster: conceito e características, https://www.infowester.com/cluster.php, Acesso em 5 de dezembro.
[6]Andre, Introdução ao Processamento Paralelo e ao Uso de Clusters de Workstacions em Sistemas GNU/LINUX, https://listman.redhat.com/archives/fedora-users-br/2006-December/pdfXbBDApNiw1.pdf, Acesso em 5 de dezembro.
[7] Anonimo, Aglomerado Beowulf https://pt.wikipedia.org/wiki/Aglomerado_Beowulf#Como_implementar, Acesso em 5 de dezembro.



3.2.1.1. MPICH

É uma implementação aberta de alto desempenho do standard MPI, o padrão para bibliotecas de passagem de mensagens para aplicações de memória distribuída usados na computação paralela.[8]

MPI significa *Message Passing Interface*, já o CH vem do *Chamaleon* que é a camada de portabilidade utilizada no MPICH para fornecer portabilidade aos sistemas de passagem de mensagens existentes.

Para que o MPICH funcione, é necessário que todas as máquinas se conheçam previamente e que exista uma conexão estabelecida, normalmente utiliza-se o protocolo SSH (*Secure Shell*), para que a troca de mensagens seja segura e não consiga ser comprometida.

O MPICH reconhece automaticamente arquiteturas *multicore* e otimiza a comunicação para essas plataformas e tem a vantagem de não ser necessária nenhuma operação de configuração especial.

Tabela 2 - Vantagens e Desvantagens MPICH

Vantagens[6]	**Desvantagens**[6]
• Cada processo possui as suas próprias variáveis locais; • Pode ser utilizada numa ampla gama de problemas; • Execução de arquiteturas de memória partilhada; • Computadores de memória distribuída são menos dispendiosos que os grandes computadores de memória partilhada.	• Requer mais mudanças de programação para passar da versão «serial» para a versão «parallel»; • Pode ser difícil de efetuar debug; • O desempenho é limitado pela rede de comunicação entre os nós.

3.2.2. MOSIX

O *Multicomputer Operating System for Unix*, MOSIX, é um conjunto de *software*, feito para Linux, que permite a implementação de clusters, sendo os seus pontos fortes a realocação de carga e o seu elevado desempenho. Por ser feito para Linux, não serão necessárias modificações das aplicações nem de

[8] Anónimo, MPICH Overview, https://www.mpich.org/about/overview/, Acesso em 5 de dezembro.

bibliotecas especiais. Para além disto, o MOSIX tem ainda um algoritmo de descoberta de recursos, o qual permite a cada nó de ter informações dos recursos disponíveis e o estado dos outros nós. Devido às informações dos recursos e dos nós disponíveis, este algoritmo pode efetuar a realocação de recursos entre os nós.[9]

As principais características do MOSIX é a *single-system image* - permite que os utilizadores façam *login* em qualquer nó e não precisam de saber onde estão a ser executados os programa - algoritmo de descoberta de recursos - permite saber quais os recursos e nós disponíveis - devido a isso permite que haja um balanceamento da carga e migração de dados, pode ocorrer uma migração dinâmica de processos - um nó mais potente pode assumir uma tarefa para evitar a sobrecarga de trabalho - para além destas características ainda tem o método de prioridade, que assegura que processos prioritários são executados preferencialmente.[7]

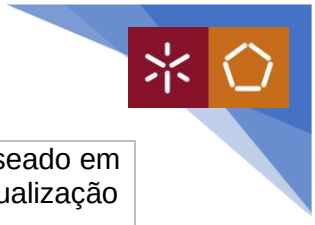
Tabela 3 - Vantagens e Desvantagens do MOSIX

Vantagens[7]	**Desvantagens**[7]
• Algoritmos de monitorização, como o algoritmo de descoberta de recursos; • Migração dinâmica de processos; • Possibilidade remoção e inclusão de nós; • Elevado desempenho; • Método de prioridade; • Benéfico para base de dados que não utilizam memória compartilhada; • Benéfico para processos que possuem tempos de execução enormes e baixo volume de comunicação entre processos;	• Só funciona no sistema operativo Linux; • Aplicações dependentes do hardware que necessitam de acesso a um periférico de um nó em especial não podem ser distribuídas; • Processos com elevado volume de comunicação, tem baixa performance;

3.2.3. Requisitos vs Clusters

1. Não cumpre
2. Cumpre

[9] Anónimo, About MOSIX, https://mosix.cs.huji.ac.il/txt_about.html, Acesso em 6 de dezembro.



Requisitos Cluster	Tolerante a falhas	Balanceamento da carga	Partilha de recursos	Aumento do desempenho na execução de tarefas paralelizáveis	Baseado em virtualização
Beowulf	1	1	2	2	2
Mosix	2	2	2	2	2

4. Implementação

4.1. Beowulf

4.1.1. Arquitetura do Sistema

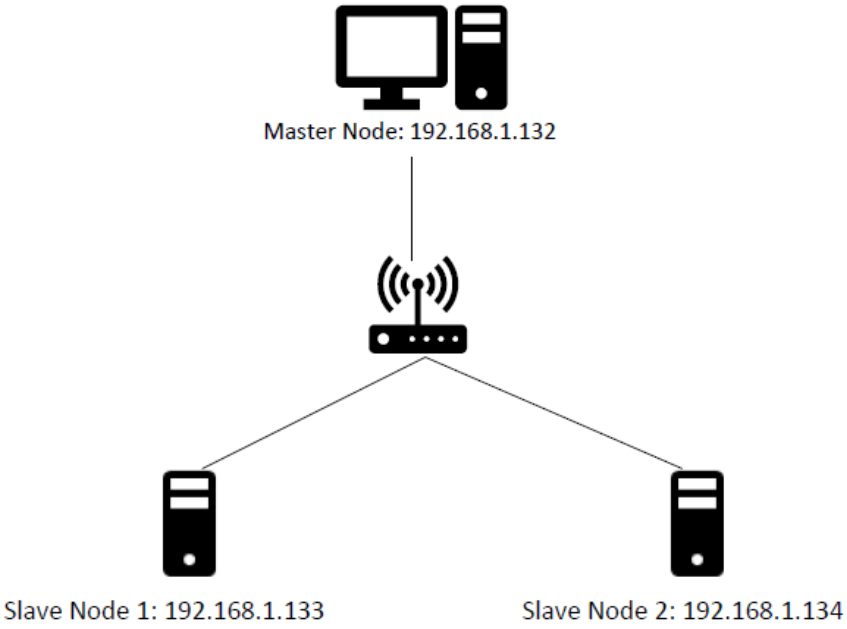


Figura 1 - Protótipo Beowulf

Na figura acima encontra-se representado o protótipo *Beowolf*, onde foram utilizadas três máquinas virtuais com a versão *Ubuntu 14.04.6 Server*. Esta imagem descreve um ambiente *cluster* com um *master node* e dois *slaves nodes* conectados entre eles.

O *master node* trata-se do nó de controlo do *cluster* e é este quem administra e distribui as tarefas entre os *slave nodes*, quem os monitoriza e quem controla ainda o *cluster* como um todo. Os *slave nodes* ficam responsáveis por processar todas as instruções enviadas pelo *master node* e posteriormente devolvê-las para que o master possa juntar todos os resultados enviados por todos os *slave nodes* e assim gerar o resultado final consolidado, retornando-o para o utilizador do sistema.

Todos os *nodes* são interligados por uma rede, de maneira a assegurar a troca de mensagens entre os membros do *cluster*.

Posto isto, todas as máquinas necessitaram de ser configuradas de maneira a suportarem o modelo de *cluster* em questão.

4.1.2. Instalação e configuração do SO

Primeiramente, de forma a implementarmos a arquitetura descrita, foram criadas 3 máquinas virtuais através da *VirtualBox*. De seguida, realizámos a instalação do sistema operativo *Ubuntu Server* em todas as máquinas.

Tabela 4 - Endereços IP | Protótipo Beowulf

Node	*Hypervisor*	*Virtual Machine*	*IP Virtual Machine*
Master	VirtualBox	Ubuntu Server 14.04.6	192.168.1.132
1	VirtualBox	Ubuntu Server 14.04.6	192.168.1.133
2	VirtualBox	Ubuntu Server 14.04.6	192.168.1.134

Posteriormente, as máquinas foram configuradas com os IP da tabela acima. Essa configuração foi feita através do comando *sudo nano /etc/networks/interfaces*. Como exemplo, podemos ver na imagem abaixo a configuração da máquina Master.

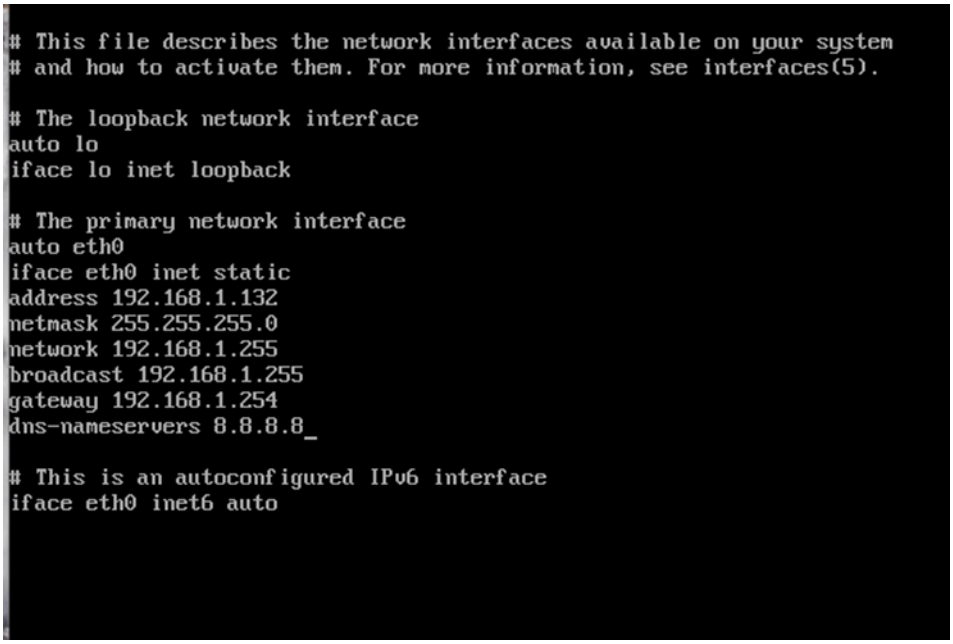


Figura 1 - Configuração da máquina master | Protótipo Beowulf

Ficamos então com 3 *hosts* como podemos ver pela figura abaixo, sendo *master* o *master node* e os *node1* e *node2* os *compute nodes*.

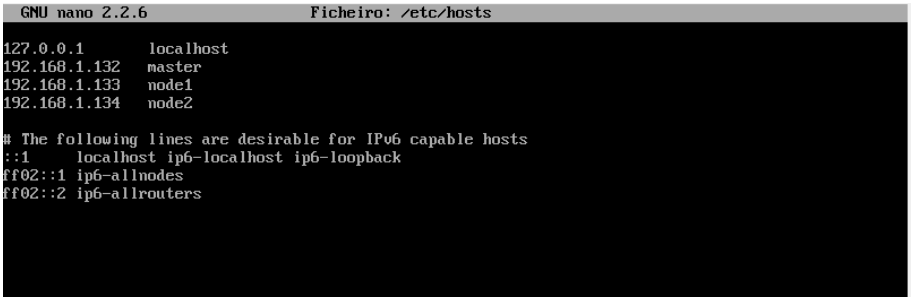


Figura 2 - *Nodes* existentes | Protótipo Beowulf

Para testar se todos os computadores estavam conectados na mesma rede, usamos o comando *ping* entre as três máquinas. Na figura abaixo, tem a demonstração de dois exemplos.

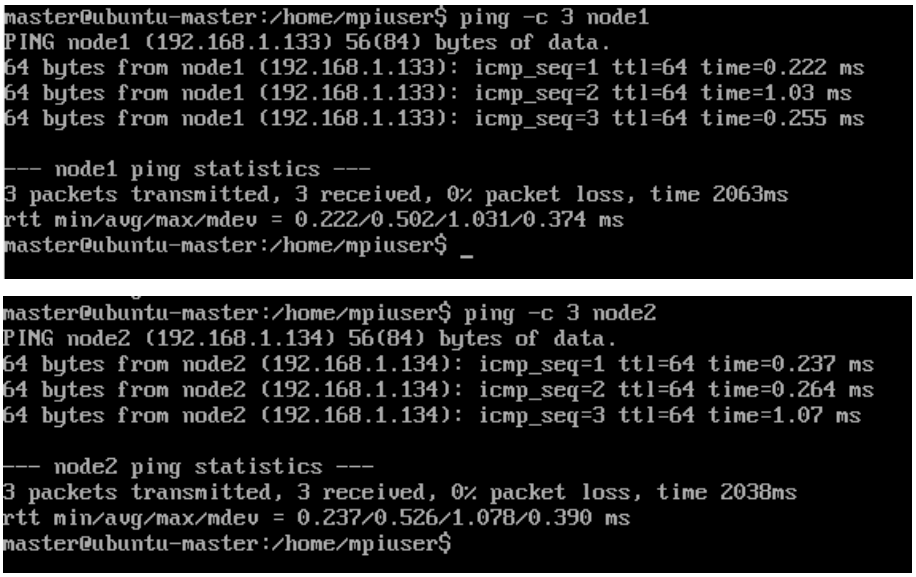


Figura 3 - Comando *ping* em duas máquinas distintas | Protótipo Beowulf

4.1.3. Definição de um utilizador para executar MPI Jobs

Posteriormente, como podemos observar na figura abaixo, criamos um novo utilizador com o *username* "mpiuser" e *user* ID 888. Todos os MPI *users* devem ter o mesmo *username* e *user* ID porque posteriormente daremos acesso ao MPI *user* na diretoria NFS.

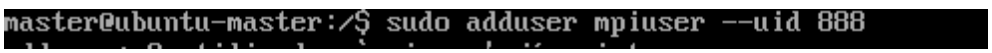


Figura 4 - Comando *sudo adduser mpiuser --uid 888* | Protótipo Beowulf

Este comando criou ainda uma nova diretoria */home/mpiuser* que será a diretoria *home* do utilizador mpiuser, que iremos utilizar para executar jobs no cluster.

4.1.4. Instalação e configuração do *Network File System (NFS)*

O NFS foi instalado através do comando *sudo apt-get install nfs-kernel-server* e para ser possível montar um NFS no *compute node*, a *package nfs-common* terá de ser instalada em todos os *compute nodes* através do comando *sudo apt-get install nfs-common*.

```
master@ubuntu-master:/$ sudo apt-get install nfs-kernel-server
```

Figura 5 – Instalação do NFS | Protótipo Beowulf

```
node@ubuntu-node:~$ sudo apt-get install nfs-common
sudo: unable to resolve host ubuntu-node
[sudo] password for node:
Desculpe, tente novamente.
[sudo] password for node:
A ler as listas de pacotes... Pronto
A construir árvore de dependências
A ler a informação de estado... Pronto
nfs-common já está na versão mais recente.
0 pacotes actualizados, 0 pacotes novos instalados, 0 a remover e 30 não actualizados.
node@ubuntu-node:~$
```

Figura 6 - Instalação da package *nfs-common* | Protótipo Beowulf

Iremos utilizar o NFS para partilhar a diretoria *home* do MPI *user* com os *compute nodes*, sendo por isso importante que esta diretoria seja da posse do MPI *user* para que todos os MPI *users* possam ter acesso a esta diretoria. Como criamos esta diretoria com o comando *adduser*, esta já está na posse do MPI *user* como podemos verificar abaixo.

```
master@ubuntu-master:~$ ls -l /home/ | grep mpiuser
drwxr-xr-x 6 mpiuser mpiuser 4096 Jan  7 16:06 mpiuser
master@ubuntu-master:~$ _
```

Figura 7 - Posse da diretoria *home* por parte do MPI *user* | Protótipo Beowulf

Iremos então partilhar a diretoria */home/mpiuser* do *master node* com os outros *nodes*, mas para isso tivemos de alterar o ficheiro *exports* no *master node* e adicionar a linha */home/mpiuser *(rw,sync,no_subtree_check)*.

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/mpiuser *(rw,sync,no_subtree_check)
```

Figura 8 - Ficheiro *exports* | Protótipo Beowulf

Após a instalação tivemos de reiniciar o NFS *kernel server* através do comando *sudo service nfs-kernel-server restart*. Isto exporta também as diretorias listadas no */etc/exports*. Quando o ficheiro *exports* foi modificado tivemos de executar o comando *sudo exportfs -a.*

```
master@ubuntu-master:~$ sudo exportfs -a
```

Figura 9 - Comando *sudo exportfs -a* | Protótipo Beowulf

A diretoria */home/mpiuser* está então partilhada através do NFS, podemos confirmar isto executando o comando *showmount -e master*, se o resultado for */home/mpiuser* significa que a diretoria está a ser partilhada.

```
node@ubuntu-node:/$ showmount -e master
Export list for master:
/home/mpiuser *
node@ubuntu-node:/$
```

Figura 10 - Diretoria a ser partilhada | Protótipo Beowulf

Para que a diretoria */home/mpiuser* seja automaticamente montada quando os *compute nodes* são iniciados tivemos de editar o ficheiro *fstab* adicionando a linha *master:/home/mpiuser /home/mpiuser nfs.*

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/dev/mapper/ubuntu--node2--vg-root /               ext4    errors=remount-ro 0       1
# /boot was on /dev/sda1 during installation
UUID=05a81014-4c1c-4d30-acb4-0613e1b38226 /boot           ext2    defaults        0       2
/dev/mapper/ubuntu--node2--vg-swap_1 none            swap    sw              0       0
master:/home/mpiuser /home/mpiuser nfs
```

Figura 11 - Ficheiro *fstab* | Protótipo Beowulf

Como a *firewall* é ativa no *Ubuntu* por defeito, esta irá bloquear o acesso quando um cliente tentar aceder a uma diretoria partilhada por NFS. Tivemos

então de adicionar uma regra com UFW para permitir o acesso de uma *subnet* específica. Para isso utilizamos o comando *sudo ufw allow from 192.168.1.0/24*.

Figura 12 - Comando *sudo ufw allow from 192.168.1.0/24* | Protótipo Beowulf

Após isto, demos *reboot* aos *compute nodes* (node1 e node2) e verificamos através do comando *ls /home/mpiuser,* se os *nodes* 1 e 2 tinham acesso aos dados do *master node* (master). Na figura abaixo encontra-se um exemplo do node1.

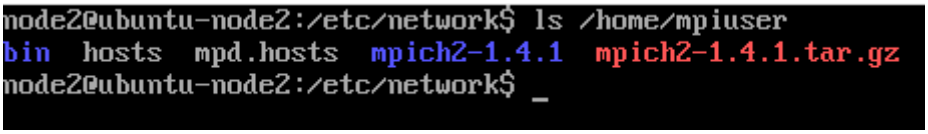


Figura 13 - Acesso do *node1* aos dados do *master node* | Protótipo Beowulf

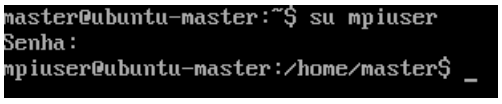
Como listou os ficheiros da diretoria */home/mpiuser* do *master node* significa que os *nodes* 1 e 2 têm acesso aos dados do master node.

4.1.5. Configurar SSH sem passwords para comunicação entre nodes

Primeiramente instalamos o SSH *server* em todos os nodes através do comando *sudo apt-get install ssh*.

Figura 14 - Instalação do SSH Server | Protótipo Beowulf

Depois tivemos de gerar uma SSH *key* para todos os MPI *users* em todos os *nodes*. A SSH *key* é por defeito criada na diretoria *home* do utilizador. No nosso caso, como já vimos anteriormente, a diretoria *home* do *user* é a mesma para todos os *nodes* por isso se gerarmos uma SSH *key* para o MPI *user* num dos *nodes*, todos os *nodes* terão automaticamente uma SSH *key*. Geramos então uma SSH *key* para o MPI *user* no *master node* (master).



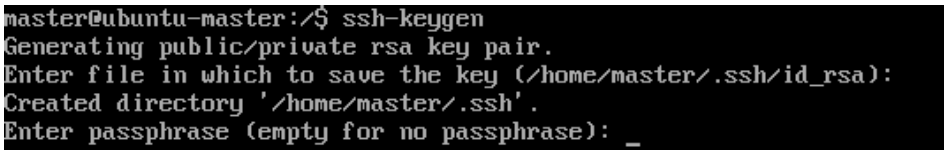


Figura 15 - Criação de uma SSH *key* | Protótipo Beowulf

O *master node* necessita de ser capaz de iniciar sessão automaticamente nos *compute nodes* e para ativar isso, a *public* SSH *key* do *master node* precisa de ser adicionada à lista de *hosts* conhecidos de todos os *compute nodes.* Como todos os dados da SSH *key* é armazenada num local (/home/mpiuser/.ssh/) no *master node*, em vez de termos de copiar a *public* SSH *key* do *master* para todos os *compute nodes* separadamente, podemos apenas copiar a mesma para o ficheiro "authorized keys" do próprio *master node*.

```
mpiuser@ubuntu-master:/home/master$ ssh-copy-id localhost_
```

Figura 16 - Cópia da *public* SSH *key* | Protótipo Beowulf

Graças ao comando acima, a própria SSH *key* do *master* foi copiada para */home/mpiuser/.ssh/autorized_keys*, mas uma vez que a */home/mpiuser* é partilhada com os *nodes* via NFS, todos os *nodes* têm agora a *public* SSH *key* do *master*, o que significa que será possível iniciar sessão nos *compute nodes* a partir do *master node* sem ser necessário introduzir uma *password*. O comando abaixo demonstra o início de sessão no *node1* via SSH.

```
mpiuser@ubuntu-master:/home/master$ ssh node1
Welcome to Ubuntu 14.04.6 LTS (GNU/Linux 4.4.0-142-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

 System information disabled due to load higher than 2.0

New release '16.04.7 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2019.
Last login: Fri Jan  7 16:18:37 2022 from node1
mpiuser@ubuntu-node:~$ echo $HOSTNAME
ubuntu-node
mpiuser@ubuntu-node:~$
```

Figura 17 - Início de sessão no *node1* via SSH | Protótipo Beowulf

4.1.6. Configuração da Gestão de Processos

Primeiramente instalamos o MPICH em todos os *nodes* através do comando *sudo apt-get install mpich2.*

Usamos a versão 3 do mpich2 e o Hydra para a gestão de processos. Para configurar o Hydra, tivemos de criar um ficheiro no *master node* que contém todos os *host names* dos *compute nodes*. Criamos esse ficheiro na diretoria *home* do MPI *user*.

Figura 18 - Comando *touch hosts* | Protótipo Beowulf

Para conseguir enviar *jobs* para os outros *nodes*, adicionamos então os *host names* dos *compute nodes* no ficheiro *hosts*.

Figura 19 - Ficheiro *hosts* | Protótipo Beowulf

4.1.7. Verificação dos Requisitos

Balanceamento da carga computacional, repartindo uma tarefa por vários processadores disponíveis

O Beowulf tem a característica de repartir processamento pelos nós de computação, no entanto esta repartição apenas é possível de ser feita manualmente. O Beowulf não consegue implementar o balanceamento de cargas automático, o que implica um desperdício de recurso de computação em algumas situações.

Partilha de Recursos, permitindo a otimização na utilização dos mesmos

Através do teste cpi, utilizado no balanceamento de carga computacional, juntamente com o teste *hellow* podemos verificar o balanceamento da carga com a partilha dos recursos. O teste *hellow* é um *script* simples onde é pedido a todos os nós de computação do *cluster* uma mensagem “Hello word from process (número do nó correspondente) of (números de nós no total). O teste é executado na raiz da pasta partilhada, com o utilizador mpiuser e com o comando *mpiexec -f hosts -n 3 /home/mpiuser/mpich2-1.4.1/examples/hellow*. Na análise da resposta conseguimos verificar o número de recursos existentes e o nome dos *hosts*.

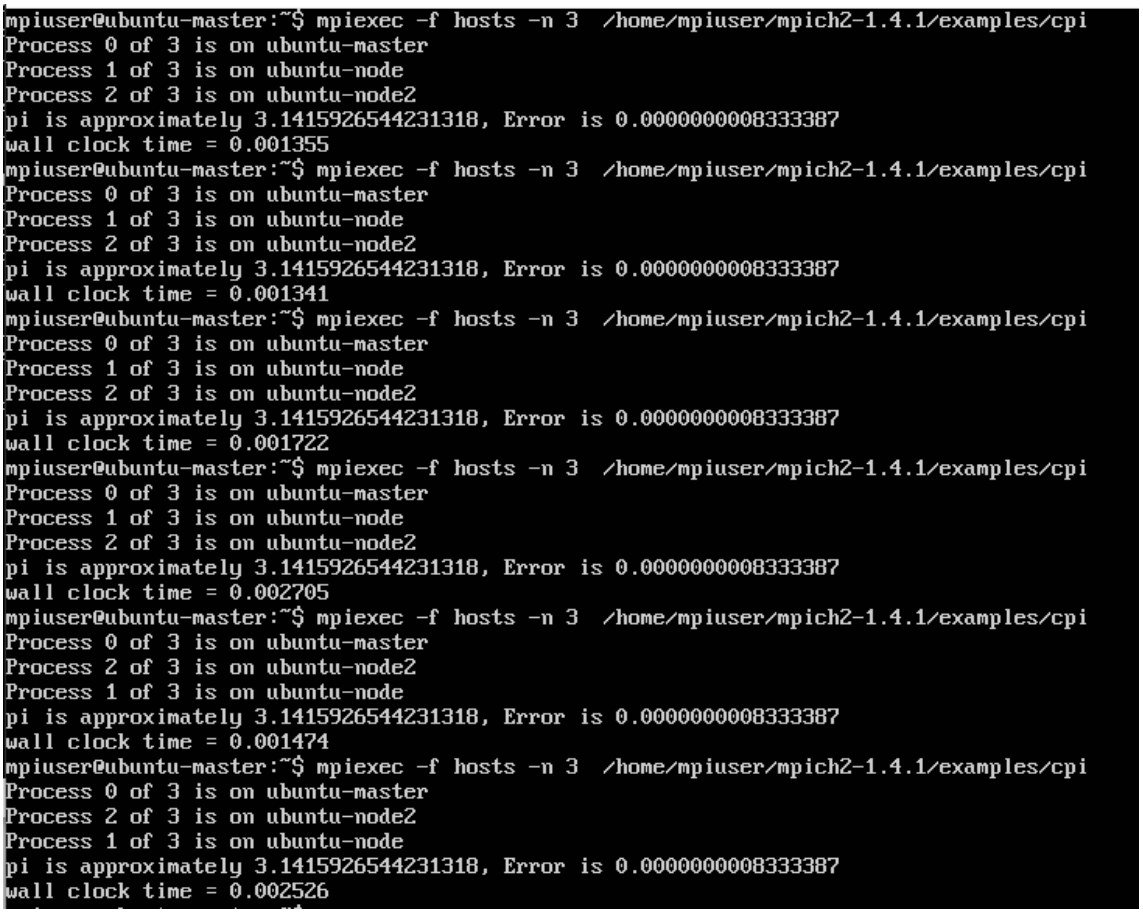


Figura 20 - Beowulf Cluster | Protótipo Beowulf

Tolerância a falhas

Para verificação do requisito a tolerância a falhas foram realizados vários testes. Na realização dos testes utilizamos o teste do *mpich2 pmandel* e utilizamos também, o comando *top* no *node* e no *node2* para verificar os processos a serem executados no terminal dos nodes. Durante a realização do teste simulamos a falha de um nó de computação, com o desligar do *node*, verificamos que ocorreu uma falha critica no Beowulf. Com este teste conseguimos afirmar que o Beowulf não é tolerante falhas.

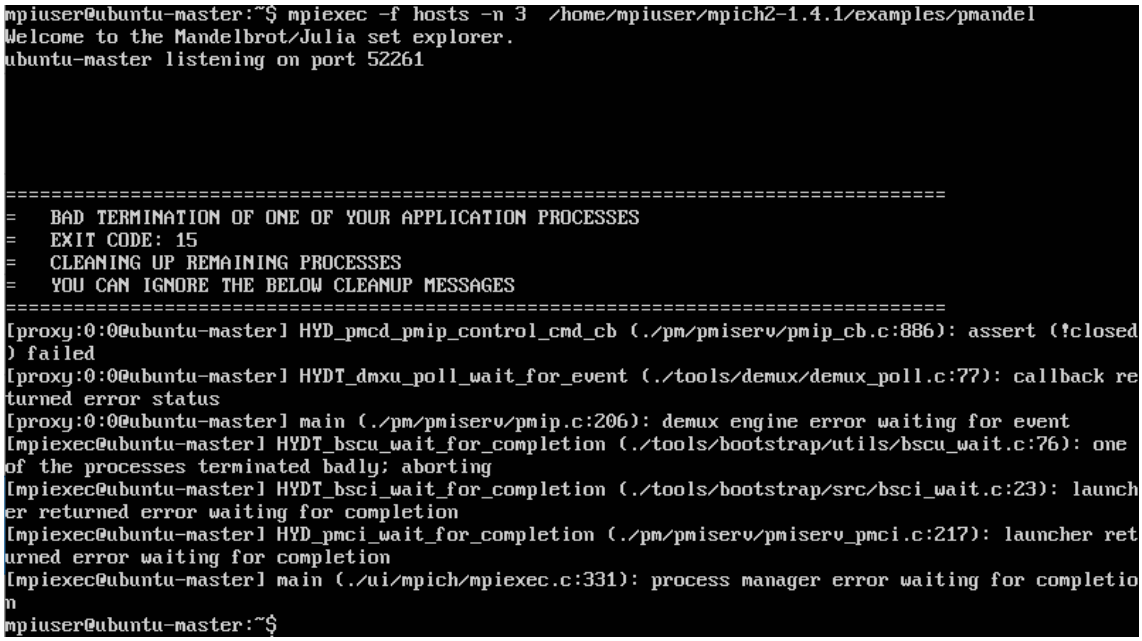


Figura 21 – Falha crítica no sistema | Protótipo Beowulf

Aumento do desempenho na execução de tarefas paralelizáveis

Na verificação de requisitos de execução de tarefas paralelizáveis verificou-se através do teste cpi, como referido anteriormente esta *script* de computação executa processos em paralelo. Nesta análise conseguimos verificar que com cálculo médio do valor do PI, quanto maior o número de nós a executar os processos, maior será o tempo de execução. Este aumento de valor do tempo não deveria acontecer, por forma a efetuar o aumento do desempenho na execução de tarefas paralelizáveis.

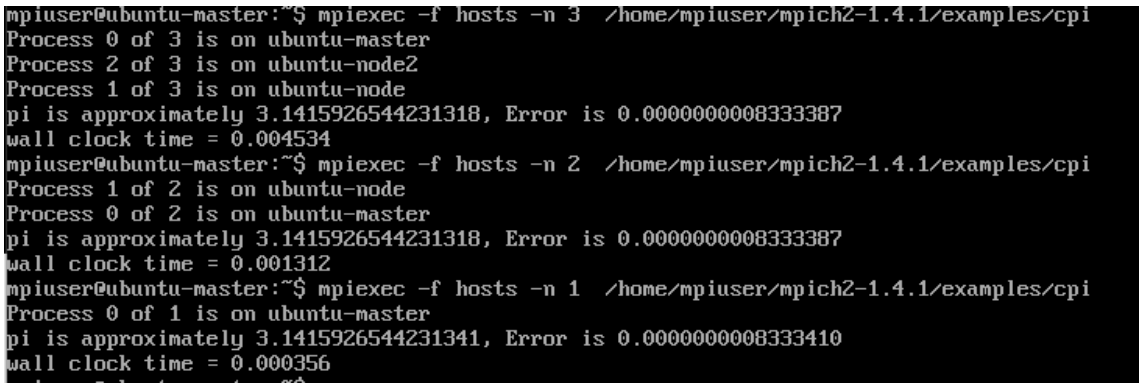


Figura 22 - Teste do aumento do desempenho | Protótipo Beowulf

Baseado em Virtualização

A implementação do Beowulf foi realizada em diferentes máquinas de computação com a técnica de virtualização., logo o requisito “Baseado em Virtualização” é realizado com sucesso.

4.2. MOSIX

4.2.1. Arquitetura do Sistema

Na figura seguinte está representada a arquitetura que foi utilizada para a implementação do MOSIX, na qual estão representadas 3 máquinas virtuais, todas elas com o sistema operativo *Ubuntu 14.04.4 Server*.

A cada máquina virtual foi atribuído um endereço IP, também foi instalado o *MOSIX 4.4.4* e foram configuradas de forma a suportarem o *cluster* em questão. As 3 máquinas virtuais, criadas através da *VirtualBox*, fazem parte de um *cluster mosix* cujo node de iniciação tem o endereço IP 192.168.1.20.

Cada máquina virtual está ligada ao router, por forma a assegurar a troca de mensagens entre os diferentes nodes do cluster.

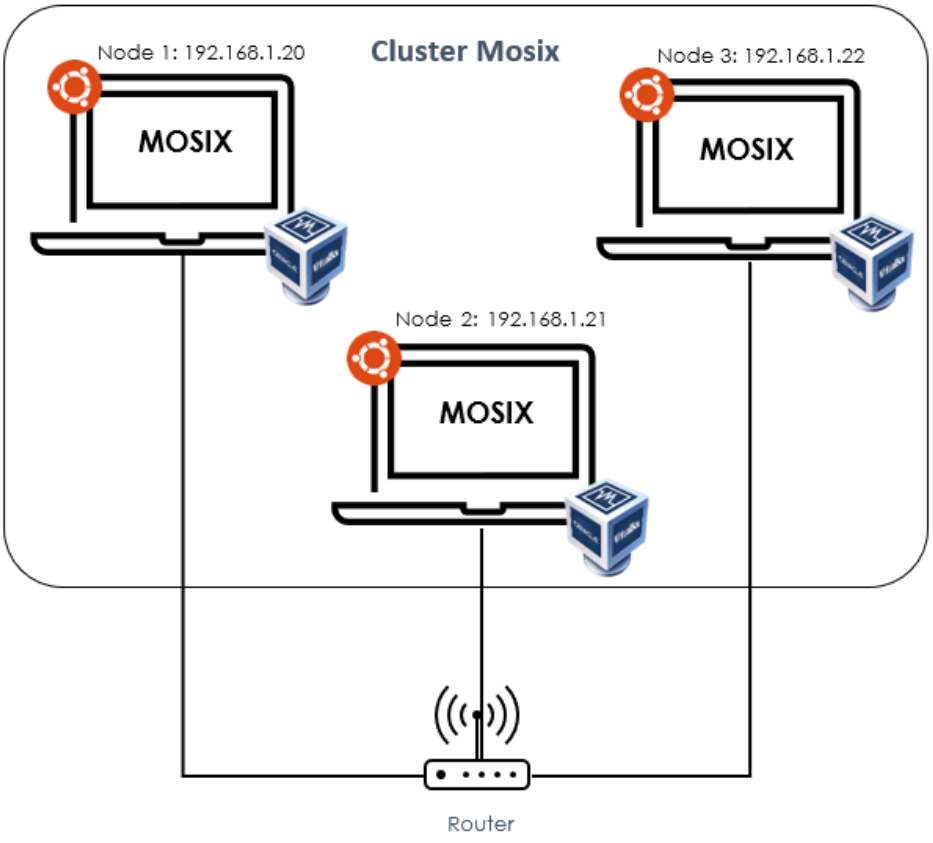


Figura 2 - Protótipo MOSIX

4.2.2. Instalação e configuração do SO

Primeiramente, de forma a implementarmos a arquitetura descrita, foram criadas 3 máquinas virtuais através da VirtualBox. De seguida, realizámos a instalação do Sistema operativo Ubuntu Server em todas as máquinas.

Tabela 5 - Endereços IP | Protótipo MOSIX

Node	*Hypervisor*	*Virtual Machine*	*Mosix*	*IP Virtual Machine*
1	VirtualBox	Ubuntu Server 14.04.4	Versão 4.4.4	192.168.1.20
2	VirtualBox	Ubuntu Server 14.04.4	Versão 4.4.4	192.168.1.21
3	VirtualBox	Ubuntu Server 14.04.4	Versão 4.4.4	192.168.1.22

Posteriormente, as máquinas foram configuradas com os IP da tabela acima. Essa configuração foi feita a partir do comando /etc/network/interfaces. Como exemplo, podemos ver na imagem abaixo a configuração da máquina 1.

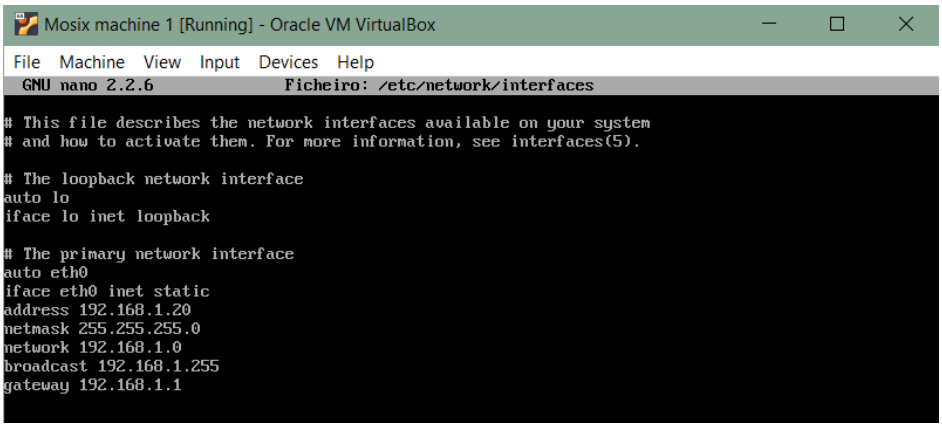


Figura 3 - Configuração da máquina 1 | Protótipo MOSIX

Para nos certificarmos que essa configuração foi feita corretamente, usamos o comando *ifconfig* em cada máquina, como apresentado na figura abaixo.

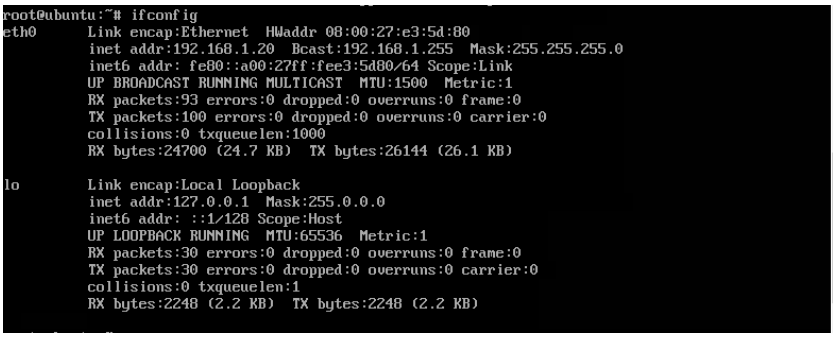


Figura 4 - Comando *ifconfig* | Protótipo MOSIX

Para testar se todos os computadores estavam conectados na mesma rede, usamos o comando ping entre as várias máquinas. Na figura abaixo, tem a demonstração de um exemplo.

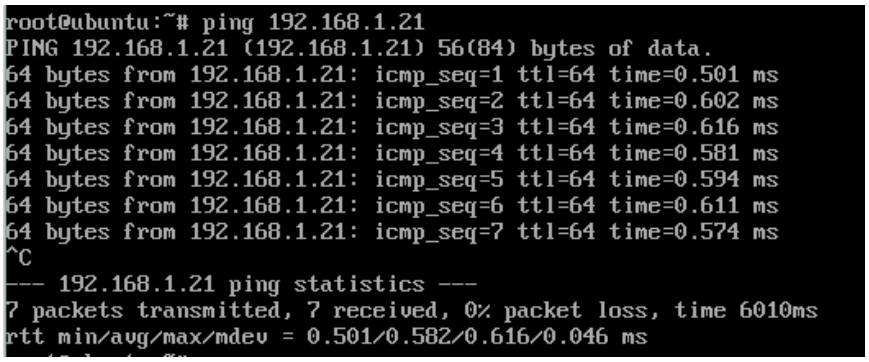


Figura 5 - Comando *ping* | Protótipo MOSIX

Após a conclusão das configurações e os testes de conectividade, a equipa teve que fazer a instalação de alguns pacotes básicos, nomeadamente através dos seguintes comandos, *apt -get update*, *apt -get install build-essencial* e *apt -get install gcc*.

4.2.3. Instalação do MOSIX

Para o processo de instalação do MOSIX, o grupo criou uma diretoria com o nome de “mosix” em todas as máquinas existentes, através do comando *mkdir /etc/Mosix*. Nessa diretoria, criamos uma subdiretoria “var”.

Após isso, fizemos o download do Mosix- 4.4.4 para a diretoria criada em cada máquina, a partir do comando *wget www.mosix.cs.huji.ac.il/mos4/MOSIX-4.4.4.tbz.*

Depois de obter o ficheiro, foi necessário descompactá-lo através do comando *tar -xjvf MOSIX-4.4.4.tbz*. Na figura seguinte, é possível verificar que foi feita o download do MOSIX, bem como a descompactação do ficheiro.

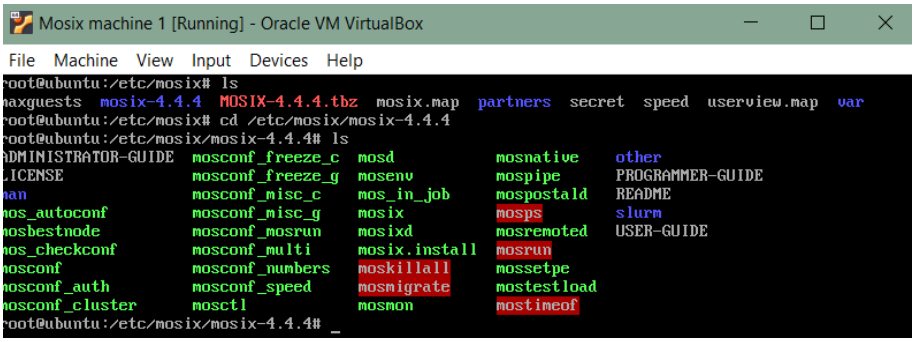


Figura 6 - Descompactação do MOSIX | Protótipo MOSIX

Feito o download e a descompactação do ficheiro, é realizada a instalação do MOSIX com o comando *./mosix.install*.

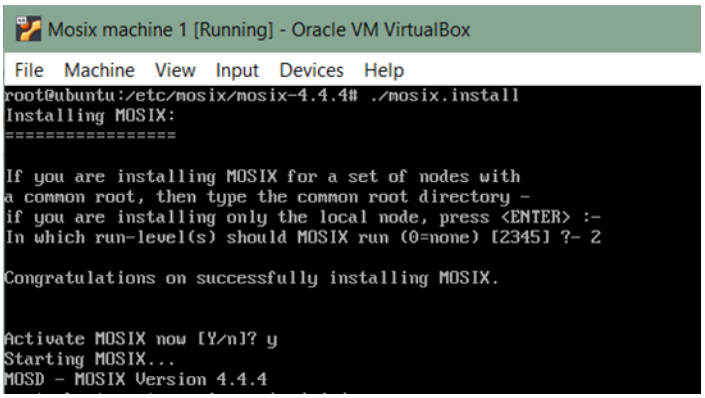


Figura 7 - Instalação do MOSIX | Protótipo MOSIX

Após a instalação, é apresentado um painel de configuração do MOSIX, que podemos aceder através do comando *mosconf*, como podemos ver na figura abaixo.

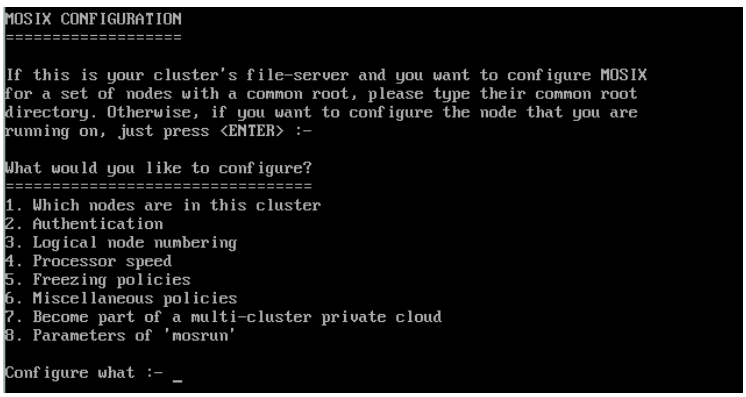


Figura 8 - Painel de Configuração do MOSIX | Protótipo MOSIX

Prosseguindo para a configuração do MOSIX, a equipa, primeiramente, acedeu à opção 1 (“Which nodes are in this cluster”), depois foi escolhida a opção “n” de modo a ser possível adicionar ao *cluster* um conjunto de *nodes*.

Seguidamente, é pedido no parâmetro “First host name or IP Address”, o primeiro IP da rede que integra o *cluster*, sendo que o IP colocado foi o da máquina 1, IP “192.168.1.20”. De seguida, colocamos o número de nodes que o *cluster* ia ter. Esta configuração foi realizada em todas as máquinas.

Figura 9 - Nº de *nodes* no *cluster* e IP onde começa | Protótipo MOSIX

O passo seguinte foi responder com a opção “2” (*Authentication*), sendo que nesta parte são feitas as configurações relacionadas com a segurança do sistema, colocando a senha “1234” para que seja possível a conexão dos computadores que possuem a mesma chave. Este procedimento deve ser realizado para as três máquinas.

Depois das três máquinas prontas, usamos o comando *mosmon*, como demostrado na figura abaixo, para verificar o processamento do *cluster*, bem como se os 3 nodes estavam corretamente identificados.

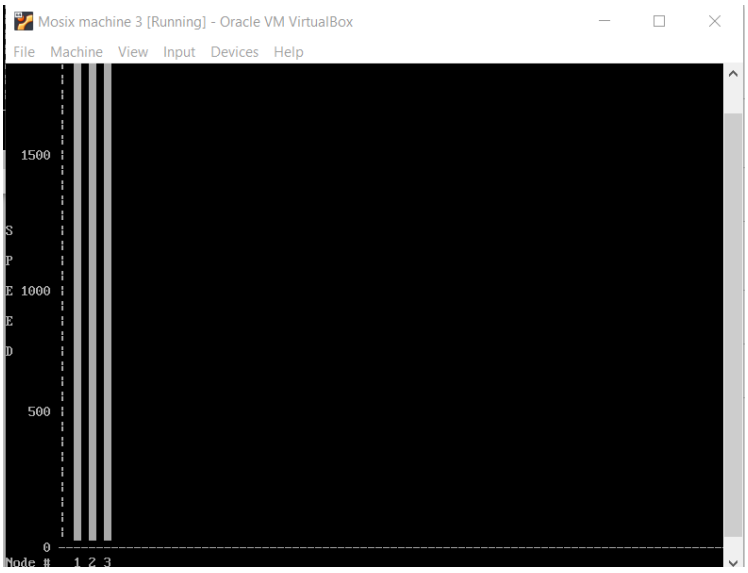


Figura 10 - Processamento do *cluster* | Protótipo MOSIX

Por último, na figura seguinte verificamos o estado do MOSIX, na máquina 1 e 2, através do comando */etc/init.d/mosix status*.

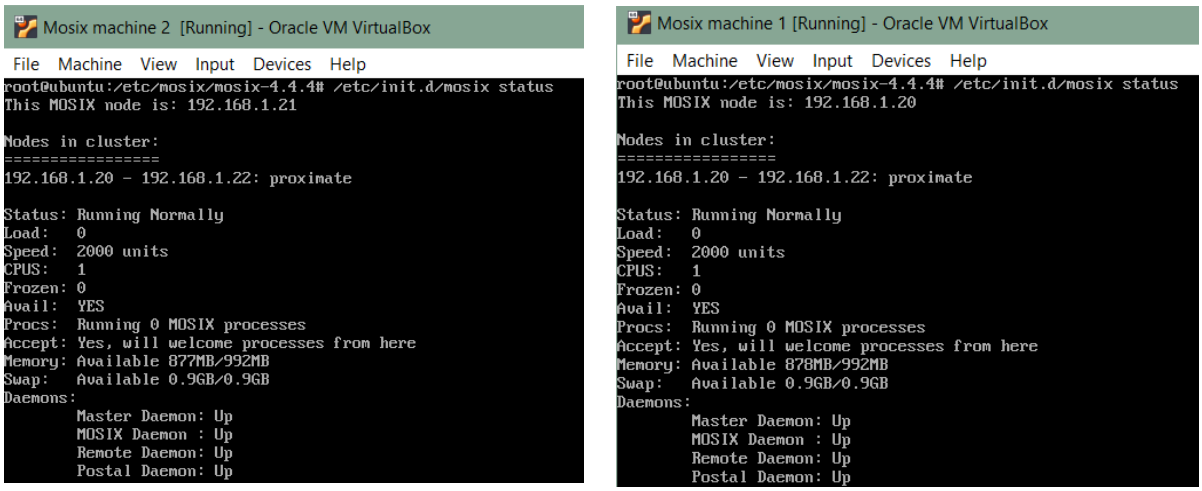


Figura 11 - Estado do MOSIX na máquina 2 e 3 | Protótipo MOSIX

4.2.4. Verificação dos Requisitos

Balanceamento da carga computacional, repartindo uma tarefa por vários processadores disponíveis

Para que possa ser testado o balanceamento automático da carga pelo cluster MOSIX, foi efetuado o comando *mosrun mostestload &,* na máquina 1 nove vezes, sendo o *mostestload* um programa de teste que gera carga artificial e consome memória para testar a operação do MOSIX.

Através do comando *mosps* é possível visualizar como é que a distribuição de carga está feita, sendo a máquina 1 a máquina *here*.

É possível de visualizar ambos os comandos a serem efetuados na imagem seguinte.

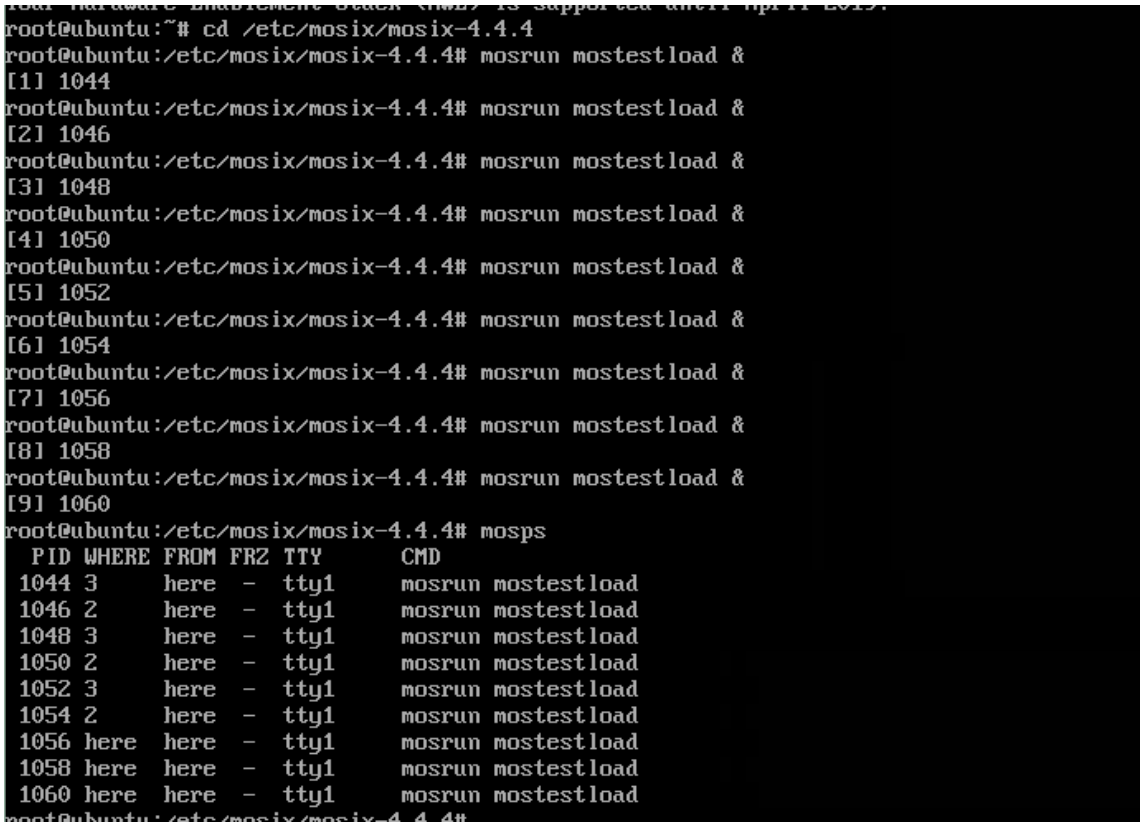


Figura 12 - Comando *mosrun mostestload &*e comando *mosps* na máquina 1 | Protótipo MOSIX

Foi possível de verificar que o balanceamento de carga aconteceu, sendo que foram distribuídos processos pelas 3 máquinas: a máquina 1 (*here*) ficou responsável 3 processos (1056, 1058 e 1060); a máquina 2 ficou responsável por 3 processos (1046, 1050 e 1054); e a máquina 3 ficou responsável por 3 processos (1044, 1048 e 1052).

Foi executado o comando *top* na máquina 2 e verificou-se que a esta máquina está a efetuar 3 processos em modo remoto (*mosremoted*) que pertencem à máquina 1.

É possível de visualizar o comando a ser efetuado na imagem seguinte.

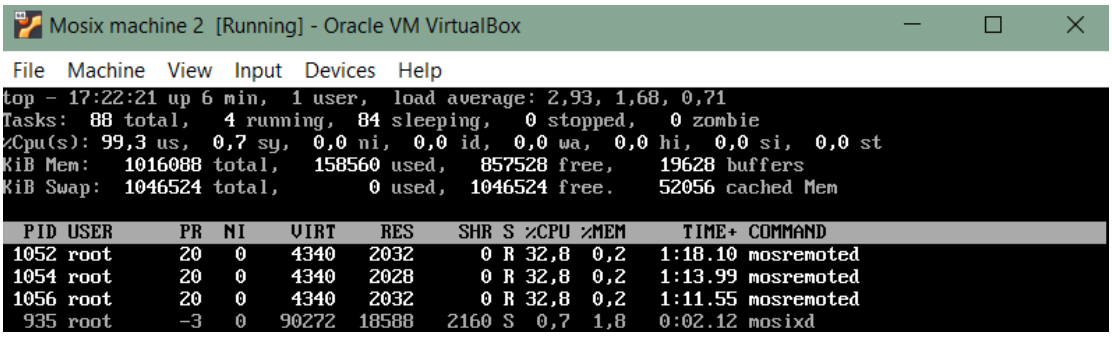


Figura 13 - Comando *top* na máquina 2 | Protótipo MOSIX

Partilha de Recursos, permitindo a otimização na utilização dos mesmos

Através do teste efetuado para o cumprimento do requisito "***Balanceamento da carga computacional, repartindo uma tarefa por vários processadores disponíveis***," é também possível verificar que a partilha de recursos acontece. O cluster MOSIX em questão, contém 3 nós, de máquina virtuais diferentes, sendo que cada nó contém um processador. Na execução de um processo, o cluster verifica a disponibilidade dos diferentes nós, e efetua a migração do mesmo para o nó com maior disponibilidade, efetuando assim a partilha de recursos e permitindo a otimização na utilização dos mesmos.

Tolerância a falhas

Para verificar que o requisito de tolerância a falhas ocorre, foi desligado o serviço MOSIX de uma das máquinas, neste caso a máquina 2, ficando apenas duas máquinas ligadas ao *cluster* MOSIX. Para desligar a máquina 2 do serviço MOSIX foi efetuado o comando representado na figura seguinte.

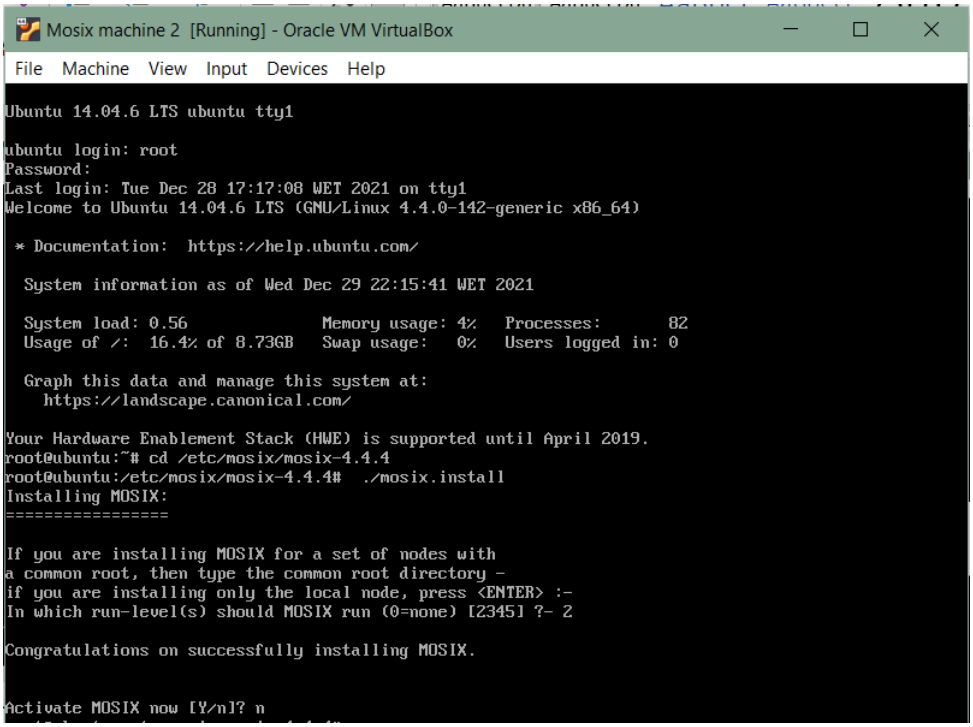


Figura 14 - Serviço MOSIX desligado na máquina 2 | Protótipo MOSIX

Através do comando *mosmon* foi possível visualizar que um dos nodes desapareceu. E agora apenas dois nodes é que estão com carga associada.

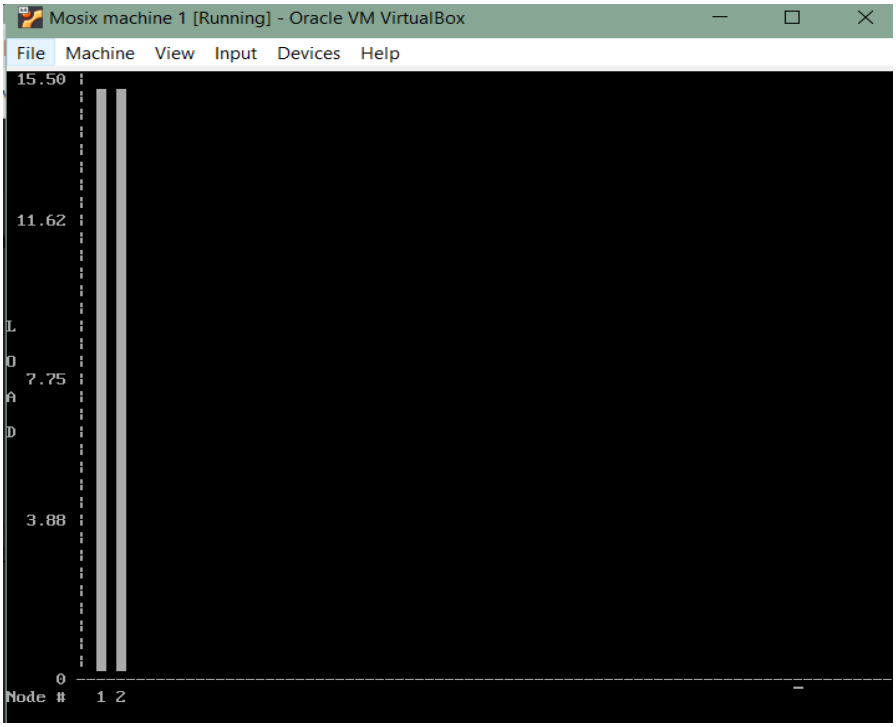


Figura 15 - Visualização de nodes existentes | Protótipo MOSIX

De forma a perceber que após a falha de um dos nodes o sistema continua a executar todos os processos, foi efetuado o comando *mosps* antes e depois

da máquina 2 se desligar do serviço MOSIX. Assim, é possível verificar como a distribuição dos processos ocorre, tal está representado na imagem seguinte.

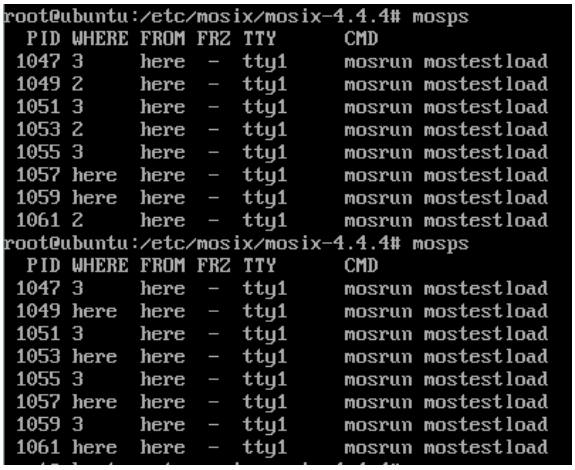


Figura 16 - Distribuição dos processos pelos nodes, antes e depois | Protótipo MOSIX

É possível concluir que o requisito de tolerância a falhas é cumprido, visto que a distribuição dos processos, após a máquina 2 ter sido desligada, acontece. Assim sendo todos os processos continuam a existir, apenas são redistribuídos pelas duas máquinas que continuam ligadas ao cluster.

Caso a máquina 2 se volte a ligar, o cluster volta a fazer o balanceamento de carga pelas 3 máquinas, como é possível de observar nas duas imagens representadas abaixo.

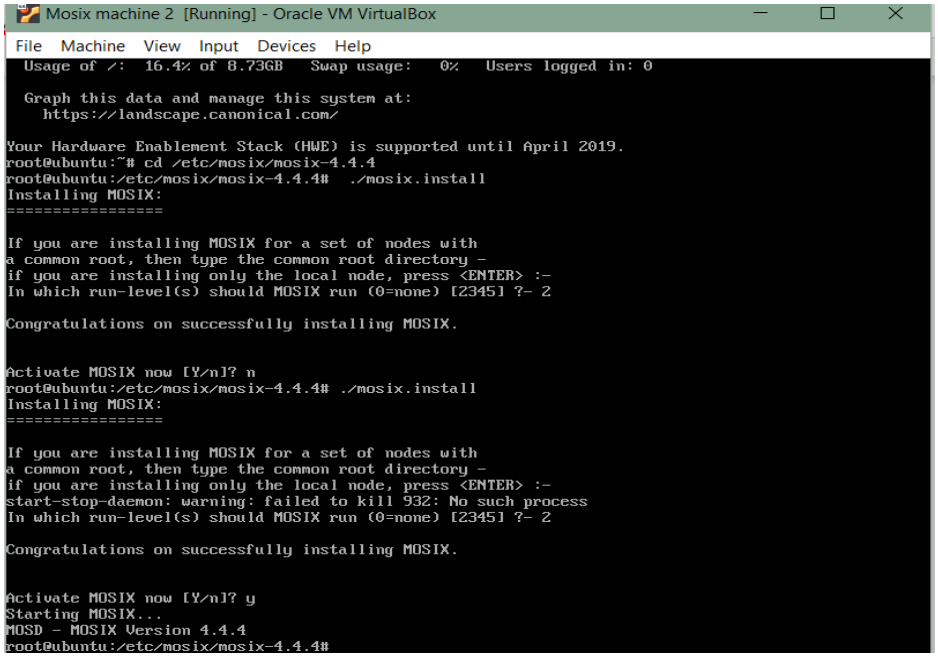


Figura 17 - Ativação do serviço MOSIX n máquina 2 | Protótipo MOSIX

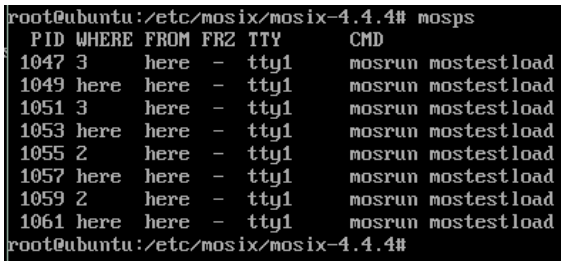


Figura 18 - Novo balanceamento de carga pelos 3 nodes | Protótipo MOSIX

Aumento do desempenho na execução de tarefas paralelizáveis

Para se verificar que existe um aumento de desempenho quando os *nodes* funcionam em *cluster*, recorreu-se à utilização de uma *script* e à comparação dos tempos de execução da mesma num node e os tempos de execução no cluster.

Foi criada uma *shell script* "iti.sh", representada na figura abaixo. Para ser possível de ter todas as permissões foi também executado o comando *chmod -x iti.sh.*

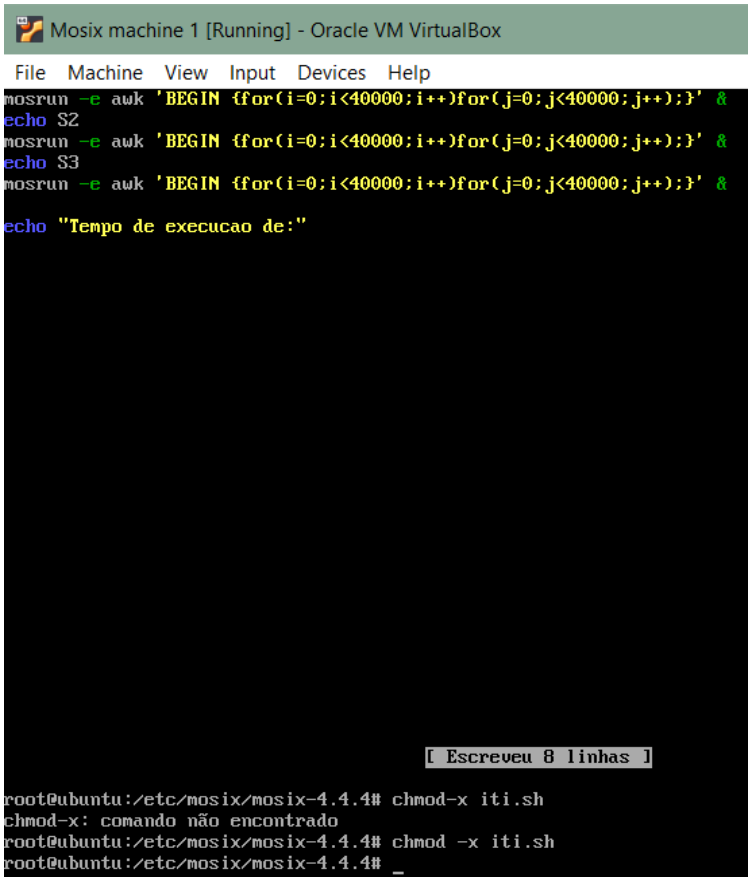


Figura 19 - *Script* iti.sh na máquina 1 | Protótipo MOSIX

Primeiramente, a *script* foi executada no *cluster* utilizando o comando *time sh iti.sh* , por forma a verificar o tempo que demora a execução da mesma.

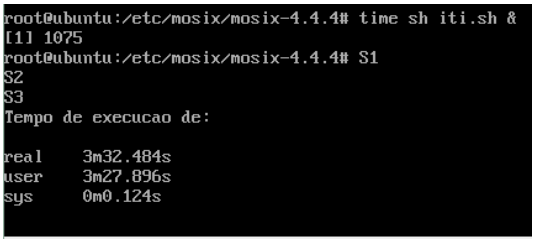


Figura 20 - Tempo de execução da *script* pelo *cluster* | Protótipo MOSIX

Seguidamente foi desligado um dos nodes do cluster, e no mesmo node foi efetuada a execução da mesma script através do comando *time bash script.sh &*.

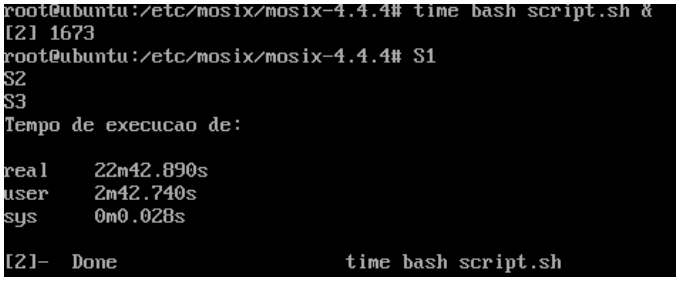


Figura 21 - Tempo de execução da *script* por um *node* | Protótipo MOSIX

Como foi possível observar pelos tempos de execução, quando a execução é feita pelo cluster o tempo da mesma é muito mais inferior (3m32s) do que quando apenas é efetuada por um node(22m42s). Assim pode-se concluir que o requisito de aumento de desempenho é cumprido com sucesso.

Baseado em Virtualização

Tendo em conta que todas as máquinas do cluster MOSIX são máquinas virtuais, então o requisito “Baseado em Virtualização” é efetuado.

5. Ferramentas

Oracle VM VirtualBox

É um *software open source* e multi-plataforma que permite criar, gerir e executar máquinas virtuais para a instalação de sistemas operativos distintos. Esta ferramenta foi fundamental para a execução do Beowulf e MOSIX.

6. Conclusão

Na realização da implementação do *cluster* Beowulf a equipa encontrou algumas dificuldades na realização do mesmo. A inexperiência da equipa com o sistema operativo Linux (Ubuntu Server) tornou-se um obstáculo no começo. A equipa encontrou problemas com o dns do router doméstico onde se encontrava a implementação das máquinas, isto é, o dns do operador do router não possibilitava a atualizações nem instalações nas máquinas virtuais. O problema foi ultrapassado com a configuração do dns (8.8.8.8). Quase no término do projeto a equipa deparou-se com uma barreira de extrema dificuldade, a implementação do mpich2 com o Ubuntu Server 20.04, esta dificuldade levou a um retrocesso do projeto. Ou seja, recomeçar o projeto da implementação do *cluster* Beowulf nas máquinas virtuais com o Ubuntu Server 14.04. A escassa informação da utilização do mpich2 também se tornou um pequeno obstáculo, mas facilmente ultrapassada devido ao conhecimento adquirido ao longo do projeto

Na implementação do *cluster* MOSIX a equipa voltou a deparar-se com algumas dificuldades. A primeira dificuldade que encontramos neste projeto foi a instalação do MOSIX 4.4.4 no Ubuntu Server 20.04.3, uma vez que faltavam alguns pacotes que impediam a instalação do mesmo. Visto que não conseguimos instalar o MOSIX 4.4.4, reunimos com o docente que nos aconselhou a instalar a versão inferior do MOSIX, o MOSIX-3.4.0.12.for_kernel-3.14. Após instalar esta versão do MOSIX, deparamo-nos com a impossibilidade de criar um cluster. Para resolução deste problema procedemos à instalação do *kernel* 3.12, no entanto não funcionou como pretendíamos, visto que continuamos com o mesmo problema.

Após entrar em contacto com o docente para o esclarecimento de dúvidas procedemos à instalação de uma versão anterior do Ubuntu, o Ubuntu Sever 14.04.4, que juntamente com o MOSIX4.4.4 permitiu a criação de um cluster.

Para além disto, o período de férias não veio facilitar a realização do trabalho, no entanto consideramos que o grupo conseguiu ultrapassar os obstáculos que foram surgindo.

7. Distribuição de Trabalho

Na primeira semana começamos por dividir as seguintes tarefas:

- Definição do *cluster* e do Beowulf e as suas vantagens e desvantagens - Andreia Almeida
- Definição do Kerrighed e OpenSSI e as suas vantagens e desvantagens- Beatriz Azevedo
- Definição MOSIX, HAProxy e as suas vantagens e desvantagens- Margarida Barros
- Definição RabbitMQ e as suas vantagens e desvantagens- João Silva
- Definição MPICH, LVS e as suas vantagens e desvantagens - José Fernandes
- Requisitos do sistema - Andreia Almeida

Na segunda semana, após a reunião com o docente decidimos que iriamos implementar o MOSIX e o Beowulf, para tal houve a seguinte divisão do trabalho:

- Implementação do MOSIX - Andreia Almeida, Beatriz Azevedo, Margarida Barros
- Implementação do Beowulf - João Silva, José Fernandes

Vale ressaltar que a Implementação tanto do MOSIX como do Beowulf inclui a arquitetura, a implementação em si e por fim, a verificação dos requisitos.

As semanas 3 e 4 foram focadas na implementação dos clusters e na resolução de problemas associadas à mesma

Na última semana, após termos as implementações feitas focamos nos mais na realização do relatório, mais precisamente:

- Introdução - José Fernandes
- Conclusão – José Fernandes
- Divisão do trabalho - Beatriz Azevedo
- Dificuldades Encontradas - Andreia Almeida, Beatriz Azevedo, Margarida Barros
- Junção do Relatório - Margarida Barros

8. Anexo

Diferentes Clusters

LVS – Linux Virtual Server

Linux Virtual Server (LVS) baseia-se num conjunto de componentes de software integrados entre si com o objetivo de distribuir uma determinada carga IP por um conjunto de servidores reais.[10]

O LVS tem como missão a criação de um servidor de alto desempenho e altamente disponível para Linux, usando para tal a tecnologia de *clustering*, que confere uma boa escalabilidade, confiabilidade e capacidade de manutenção.

A arquitetura do cluster de servidores é totalmente transparente para os utilizadores finais, uma vez que os mesmos interagem com o sistema de cluster como se fosse apenas um único servidor virtual de alto desempenho.

Vantagens	**Desvantagens**
• Maior aproveitamento dos recursos; • Ao reduzir a quantidade de hardware do servidor no centro de processamento de dados, os requisitos de refrigeração e energia diminuem; • Maior flexibilidade; • Facilidade na execução de backups, pois existe disponibilidade e recuperação asseguradas em caso de uma falha; • Migração dos servidores para um novo hardware de forma transparente; • É fácil mudar a plataforma da máquina virtual e aumentar o seu desempenho o que permite a existência de balanceamento de carga	• Vários 'hosts' partilham o mesmo ambiente físico quando se virtualiza um servidor e consequentemente durante períodos de alta utilização o desempenho pode reduzir; • A virtualização do Linux aumenta a complexidade do ambiente, caso o ambiente virtualizado estiver configurado incorretamente, pode acabar com problemas de desempenho e confiabilidade; • Com a virtualização, cada partição lógica ou ambiente virtual dentro do servidor físico poderá vir a ser comprometido;

[10] Anónimo, What is the Linux Virtual Server?,http://www.linuxvirtualserver.org/, Acedido em 6 dezembro.

RabbitMQ Cluster

O RabbitMQ Cluster é uma ferramenta para criar arquiteturas distribuídas. Os Clusters RabbitMQ cluster utiliza o mecanismo de comunicação entre processos Earlang nativo virutal machine para a comunicação entre nós, partilha a informação do estado e permite que as mensagens sejam publicadas e consumidas em todo o cluster.[11]

Vantagens
• Código aberto • Independente da plataforma e fornecedor • Ferramenta leve

HAProxy

O HAProxy, é um software de multiprocessamento, utilizado em algumas aplicações com orientação para servidores. O HAProxy consegue fazer o balanceamento de cargas de processamento para aplicações em HTTP e para o protocolo TCP. Para as aplicações que utilizem a internet e com vários requisitos simultâneos a solução HAProxy consegue fazer o balanceamento de cargas e disponibilidade.[12]

Vantagens
• Proxy de tráfego TCP e HTTP • Descarregar /iniciar conexões SSL • Normalizar tráfego HTTP • Balanceamento de carga

OpenSSI

O OpenSSI é uma solução aberta para clusters focada em ambientes LINUX. É uma opção SSI(*Single System Image)*, que significa que é um sistema

[11] Anónimo, Chapter 1. Foundational RabbitMQ, https://livebook.manning.com/book/rabbitmq-in-depth/chapter-1/21 , Acesso em 5 de dezembro.
[12] Anónimo, HAProxy, http://www.haproxy.org/, Acesso em 6 de dezembro.

que contém vários nós, mas que no ponto de vista do utilizador, este vê-o como um só computador.[13]

Vantagens[14]
• O alto desempenho; • A alta disponibilidade; • Possui recursos para balancear a carga;

Kerrighed

O Kerrighed é outra opção SSI aberta para clusters que tem como base o sistema operativo Linux. Esta solução destaca-se pelo uso do DSM *Distributed Shared Memory*, que consiste numa espécie de "Memória Compartilhada Distribuída", em que a memória de cada nó é somada aos restantes nós, formando assim um volume único à disposição de todo o cluster.[12]

Apresenta como características:

- Parte da memória pode ser disponibilizada para todos os clientes da aplicação;
- Pode-se migrar processos que usem fluxos (socket, pipe, fifo, char device, etc) sem que haja perda de desempenho de comunicação;
- Todos os discos do *cluster* são fundidos num só disco virtual, ficando numa espécie de RAID;
- É possível verificar e reiniciar processos a partir de qualquer nó do *cluster*;
- A interface *Thread Posix* consegue suportar diversas *threads* nos nós dos *clusters* o que torna a interface completa;
- O cluster pode se comportar como se fosse uma máquina com múltiplos processadores;
- As características customizáveis da imagem única do sistema, prevê que as características do SSI (memória compartilhada, escalonador global, migração de fluxos, etc) sejam ativadas ou não por base de processos.

[13] Emerson Alecrim, Cluster: conceito e características, https://www.infowester.com/cluster.php , Acesso em 4 de dezembro.
[14] Anónimo, Tipos de Cluster: parte 2 - MOSIX, OpenSSI e Kerrighed, http://becoolonit.blogspot.com/2015/11/tipos-de-cluster-parte-2-mosix-openssi.html , Acesso em 4 de dezembro.

Vantagens[15]
• O alto desempenho de aplicações; • A alta disponibilidade do cluster; • A eficiência na administração de recursos; • O alto poder de customização do sistema operacional; • A facilidade de uso.

[15] Anónimo, What is Kerrighed ?, http://www.kerrighed.org/wiki/index.php/Main_Page , Acesso em 4 de dezembro.

9. Bibliografia

- Emerson Alecrim, Cluster: conceito e características, https://www.infowester.com/cluster.php , Acesso em 4 de dezembro.
- Anónimo, Tipos de Cluster: parte 2 - MOSIX, OpenSSI e Kerrighed, http://becoolonit.blogspot.com/2015/11/tipos-de-cluster-parte-2-mosix-openssi.html , Acesso em 4 de dezembro.
- Dalvan Jair Griebler , Vinicius Casali , Claudio Schepke, Avaliação de Ferramentas de Gerenciamento de Clusters , https://d1wqtxts1xzle7.cloudfront.net/26445336/errc08-with-cover-page-v2.pdf?Expires=1641584850&Signature=LfiNTR-6Vt927xhs4roSiLBwhSgxcRSFli2IPzJsTw6xUTkcBe1e~8LsnCT7HpkKQ6BsPRPdzese4caQBc86uDV8oF57FsqU3oXYow-Jl9d5EpHV0pi6cn3Y9yjOBPq297vyOdl4nYyAa6CcwZvhRFF0e8oe56ZyznSqhIAvJW-uXP28r9CsIGv-TXzA-Xekz2UK3pkBTS8KhXzKRnr8ZWsQr3AFhpA0MElYUT7cOkkG4ToB~~Z1H2wwdhAsj9F~KYzfrOn0YBm2DKNJuezlZAlv~9BJD4Oa8E5JgUPJvNhHAb24e9uOYfkJWGSCGNG3QIdnCBg5YCSTphfv8GYN6Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA , Acesso em 4 de dezembro.
- Anónimo, What is Kerrighed ?, http://www.kerrighed.org/wiki/index.php/Main_Page , Acesso em 4 de dezembro.
- Aaron Nordhoff, O que é um cluster? Uma visão geral do clustering na nuvem, https://www.capitalone.com/tech/cloud/what-is-a-cluster/, Acesso em 5 de dezembro.
- Emerson Alecrim, Cluster: conceito e características, https://www.infowester.com/cluster.php, Acesso em 5 de dezembro.

- Anónimo, Chapter 1. Foundational RabbitMQ, https://livebook.manning.com/book/rabbitmq-in-depth/chapter-1/21 , Acesso em 5 de dezembro.
- Anónimo, Aglomerado Beowulf, https://pt.wikipedia.org/wiki/Aglomerado_Beowulf#Como_implementar, Acesso em 5 de dezembro.
- Anónimo, MPICH Overview, https://www.mpich.org/about/overview/, Acesso em 5 de dezembro.
- Andre, Introdução ao Processamento Paralelo e ao Uso de Clusters de Workstacions em Sistemas GNU/LINUX ,https://listman.redhat.com/archives/fedora-users-br/2006-December/pdfXbBDApNiw1.pdf, Acesso em 5 de dezembro.
- Serrano Pereira, Building a Simple Beowolf cluster with Ubuntu, https://www-users.cs.york.ac.uk/mjf5/pi_cluster/src/Building_a_simple_Beowulf_cluster.html#_building_a_virtual_beowulf_cluster, Acesso em 5 de dezembro.
- Taisy Silva Weber, Tolerância a falhas: conceitos e exemplos, http://www.inf.ufrgs.br/~taisy/disciplinas/textos/ConceitosDependabilidade.PDF , Acesso em 5 de Dezembro
- Matheus Souza Fonseca, Projeto de um balanceador de carga de servidores confiável para sistemas Linux , https://fga.unb.br/articles/0000/5580/TCC1_MatheusFonseca.pdf , Acesso em 5 de Dezembro
- Anónimo, About MOSIX, https://mosix.cs.huji.ac.il/txt_about.html, Acesso em 6 de dezembro.
- Anónimo, HAProxy, http://www.haproxy.org/, Acesso em 6 de dezembro.
- Sourabh, Install Linux Kernel 5.1 On Ubuntu, https://sourcedigit.com/24190-install-linux-kernel-5-1-on-ubuntu/, Acesso em 12 de dezembro

- Nayyar, MOSIX OS- Creating Cluster in MOSIX OS, https://www.youtube.com/watch?v=u3F0YqpjeoA, Acesso em 15 de dezembro.

- Salgado, Mosix Grupo 4 Sistemas Operativos 2, https://www.youtube.com/watch?v=lVB2kURBvUo, Acesso em 15 de dezembro.