IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Vítor Bruno Pacheco Pereira

fevereiro | 2016

Instituto Politécnico da Guarda
Escola Superior de Tecnologia e Gestão

Instalação e Configuração de um Cluster Beowulf

Projeto de Informática do curso Engenharia Informática

Vítor Bruno Pacheco Pereira
Nº1009088

19 de Fevereiro 2016

Instituto Politécnico da Guarda
Escola Superior de Tecnologia e Gestão

Instalação e Configuração de um Cluster Beowulf

Projeto de Informática do curso Engenharia Informática

Orientador: Professor Carlos Carreto

Vítor Bruno Pacheco Pereira
Nº1009088

19 de Fevereiro 2016

Agradecimentos

Os meus agradecimentos são para os que mais me ajudaram e contribuíram para que este projeto fosse possível.

Um agradecimento especial para o Sr. Professor Carlos Carreto por ter-me aceite como desenvolvedor deste projeto e por me orientar ao longo do seu progresso, sem a sua ajuda e orientação não seria possível.

Também quero agradecer à Sra. Professora Filipa Gaudêncio por me ajudar nas diversas adversidades que tive na realização deste mesmo projecto.

O meu último agradecimento vai para aqueles que me apoiaram e me ajudaram direta e indiretamente para que fosse possível e concretizável. A todos amigos, colegas e mesmo família que sempre me suportaram e acreditaram que tudo fosse possível, o meu muito obrigado.

Resumo

Este projeto teve como objetivo principal a criação de um Cluster Beowulf para ser utilizado pela instituição do Instituto Politécnico da Guarda.

Este projeto foi realizado com o intuito de aproveitar computadores obsoletos e pô-los a processar em paralelo, obtendo assim um poder considerável de processamento a baixo custo, fornecendo assim à Instituição de ensino um Cluster Beowulf.

Esta configuração utiliza várias máquinas ligadas no mesmo segmento de rede, com 12 nós, realizando uma computação concorrente, ou seja, aplicações ou outras tarefas, criando a ilusão de que o trabalho é feito por apenas uma máquina, chamada de "computador virtual".

Os testes realizados ao Cluster para comprovar o processamento paralelo foram o cálculo do valor de PI e análise dos resultados obtidos.

Palavras chave: Cluster, Beowulf, Linux, Processamento Paralelo, SSH, MPI.

Abstract

This project aims to create a Beowulf cluster that is used by the institution of the Polytechnic Institute of Guarda.

This project was carried out in order to take advantage of obsolete computers and put them to process in parallel, thus achieving a considerable processing power at low cost, thus providing the educational institution a Cluster Beowulf.

This configuration uses multiple machines connected to each all running on the same network that perform a concurrent computing, namely, applications or other tasks, creating the illusion that the work is done by just one machine, called "virtual computer".

The tests carried out to prove Cluster parallel processing were calculating the value of PI and analysis of the results.

Keywords: Cluster, Beowulf, Linux, Parallel Processing, SSH, MPI.

ÍNDICE

ÍNDICE DE FIGURAS

.

LISTA DE ABREVIATURAS

API	Application Programming Interface
DNS	Domain Name System
IPG	Instituto Politécnico da Guarda
ISP	Internet Service Provider
LAN	Local Area Network
MPI	Message Passing Interface
NASA	National Aeronautics and Space Administration
PC	Personal Computer
PVM	Parallel Virtual Machine
RAID	Redundant Array of Inexpensive Drives
SO	Sistema Operativo
TI	Tecnologias de Informação
VS	Virtual Server
MB	MegaByte
RAM	Random Access Memory
ARM	Advanced RISC Machine
MPICH	Message-Passing Interface Chameleon
QEMU	Quick EMUlator
CPU	Central Processing Unit
HDD	Dard Disk Drive
RPM	Red Hat Package Manager
AMD	Advanced Micro Devices inc.
DDR	Double Data Rate
VM	Virtual Machine
CAD	Computer Aided Design
HPCC	High Performance Computing Cluster
UPS	Unit Power Supply
PSU	Power Supply Unit
IP	Internet Protocol address
GW	Gateway
WAW	Wide Area Network
DHCP	Dynamic Host Configuration Protocol
TCP/IP	Transmission Control Protocol/ Internet Protocol
NAT	Network Address Translation
NFS	Network File System
VPN	Virtual Private Network
NTP	Network Time Protocol
SSH	Security Shell

1. INTRODUÇÃO

1.1. Enquadramento e motivação

Atualmente com o desenvolvimento cada vez mais rápido da tecnologia e da Informática encontram-se cada dia mais aplicações informáticas que requerem uma enorme capacidade de processamento, muito maior do que aquela que um computador pessoal comum pode fornecer.

Um supercomputador é uma máquina com grande capacidade de processamento e uma grande quantidade de memória. As suas aplicações são onde é requerida uma grande quantidade de processamento, como por exemplo em pesquisas militares, química, medicina e científica.

Estas máquinas são usadas em cálculos complexos e tarefas pesadas, como problemas nas áreas da mecânica, meteorologia, física quântica, modelagem molecular (computação nas estruturas e propriedades de compostos químicos, macromoléculas biológicas, polímeros e cristais) e simulações físicas, como simulação de aviões em túneis de vento, simulação da detonação de armas nucleares e investigação sobre a fusão nuclear. [1]

Cluster ou *Clustering* é o nome atribuído a um sistema com vários computadores, no qual o objectivo é distribuir o processamento das tarefas pelos computadores disponíveis no sistema, como um só se tratasse. Com esta implementação é possível obter resultados que até ali, só eram possíveis com supercomputadores.

Os vários computadores de um cluster designam-se por nós, todos estão ligados na mesma rede independentemente da topologia. A projecção dessa rede tem de ser construída tendo em conta o acréscimo ou retiro de máquinas, casa haja danos e ficam inoperacionais, contudo sem atrapalhar o seu funcionamento geral do *cluster*.

O SO que irá ser usados nas máquinas do *cluster* deverá ser ou Linux ou Windows, isto porque podem haver incompatibilidades em ferramentas ou mesmo no próprio funcionamento, para facilitar e evitar isso, deve-se usar o mesmo SO. [2]

A utilização de um *cluster* oferece alto desempenho a baixo custo, como centro de processamento, garante à instituição uma ferramenta viável e com a possibilidade de expansão de recursos sem a necessidade de grandes investimentos.

Uma instituição de ensino superior com este tipo de soluções é uma mais-valia, pois é no ensino superior, onde a maioria dos alunos começa a desenvolver seus primeiros projetos académicos e de iniciação científica.

As áreas que trabalham mais com computação de alto desempenho e soluções equiparadas são:

- A tecnologia da informação;
- A ciência da computação;
- As ciências ambientais;
- A engenharia;
- A física;
- A matemática e onde existam estudos/pesquisas envolvendo cálculos complexos.

No seguimento do que foi dito, refiro as principais motivações para abraçar a realização deste projecto que foram a possibilidade de trabalhar com *hardware* e *software* em conjunto. Depois também me incentivou a parte de ser feito numa rede LAN interligando as máquinas numa rede com ligação à internet.

E por último o facto de ser feito em Ubuntu e querer explorar e aprender um pouco mais de Linux, pois não possuía um conhecimento muito alargado neste sistema operativo.

Foi de facto muito desafiante poder testar os meus conhecimentos nestas áreas já descritas.

1.2. Local de desenvolvimento do projecto

Este projecto desenvolveu-se na íntegra no Instituto Politécnico da Guarda, no laboratório de sistemas operativos, foi o local que foi adoptado e disponível para instalar esta solução para a instituição.

1.3. Descrição do projecto

Desenvolvimento de um Cluster Beowulf. Fazendo uso de 12 computadores em paralelo controlados por 1 mestre para o cálculo de aplicações pesadas e processamento concorrente de elevado desempenho ou seja que utilizem *multicores*.

1.4. Objectivos

O projeto Beowulf IPG visa disponibilizar para a comunidade académica local um Cluster Beowulf (baseado em 12 PCs e no SO Ubuntu), para ser usado em estudos que necessitem de computação de alto desempenho.

Ou seja aplicações desenvolvidas em específico para processamento paralelo *multicores*, como exemplo o cálculo do valor de Pi que é o principal objectivo do projecto.

1.5. Estrutura do relatório

O documento inclui 5 capítulos, começando pela Introdução, onde foi feita uma pequena apresentação ao tema do projecto. Explica os conteúdos relacionados com o projecto, fez uma pequena introdução ao tema *clusters*, supercomputadores e processamento paralelo. Motivações que levaram à escolha deste projecto, o local da realização do mesmo. Uma breve descrição do projecto e os seus objectivos.

De seguida o segundo capítulo Trabalho Relacionado, apresenta uma breve descrição e apanhado sobre o tema, com informações detalhadas sobre os vários tipos existentes de *clusters*, onde se aplicam e os motivos para o seu uso em específico. Também vão ser apresentados exemplos de projectos conhecidos semelhantes ao realizado, detalhando o que vai ser igual ou diferente do projecto realizado.

O terceiro capítulo Desenvolvimento do Cluster Beowulf, onde será apresentado todo o tipo de trabalho realizado, desde a montagem, instalação e configuração. Este capítulo está dividido por várias secções onde serão detalhados o esquema de *hardware* e lógico do *cluster*. Em seguida vai ser detalhada toda a instalação e configuração feita no *cluster*.

No quarto capítulo serão apresentados os testes ao *cluster* desde o teste básico a verificar a sua operacionalidade até ao teste final do cálculo do valor de pi. No final serão apresentados os resultados do cálculo do valor de pi.

No quinto e último capítulo serão apresentadas as conclusões e elações deste trabalho realizado em final de curso.

2. TRABALHO RELACIONADO

Este capítulo irá conter o estudo prévio à realização do projecto, tal como a história dos *clusters* o porquê de serem usados e as tendências da actualidade.

Irão ser apresentados e descritos os vários tipos de *clusters* existentes, onde costumam ser usados, a possibilidade de serem combinados.

Uma descrição do processamento paralelo/distribuído e por fim o que é e como funciona o Cluster Beowulf que foi o usado no projecto.

2.1. História dos Clusters

O termo *cluster* foi o nome dado ao sistema que relaciona mais que dois computadores e que trabalha em conjunto para processar uma tarefa comum. Trabalhando de forma simultânea e dividindo as actividades de processamento entre elas.

É atribuído um nome a cada computador que pertence a um *cluster*, em teoria não existe limite máximo de nós a implementar num *cluster*. Contudo independentemente do seu tamanho, o *cluster* tem de ser visto como um computador só e não como vários agrupados de máquinas.

O *cluster* da figura 1 chama-se Chemnitzer Linux Cluster e pertence à Universidade Técnica de Chemnitz na Alemanha, mostra um *cluster* de computadores todos ligados na mesma rede.

Figura 1.Sala com um Cluster de múltiplos desktops (adaptado de [3])

As ligações entre os nós no *cluster* devem ser feitas com tecnologia de rede comum para facilitar a manutenção e reduzir os custos de implementação, como a Ethernet.

É uma solução bastante viável usando computadores pessoais e por vezes simples, combinando-os numa computação *cluster*, como diz o ditado a união faz a força. A solução conjunta de computadores soluciona muitas vezes desafios ou problemas que à partida não era possível solucionar a não ser recorrendo a supercomputadores ou a servidores de elevado desempenho o que acarreta maior despesa ou investimento. [3]

A ideia de interligar os Mainframes na década de 60 deu início à história dos *clusters*. Pensamento de vanguarda da poderosa IBM, empresa que já tinha grande parte do domínio da tecnologia nessa época e que pretendia obter uma solução ao mesmo tempo com computação paralela e que fosse favorável à comercialização.

Já em meados de 1980, algumas tendências surgiram, como microprocessadores de alto desempenho, o sistema de interligar máquinas começou verdadeiramente a ganhar força. Uma tendência em destaque é a necessidade de elevado poder processamento, como em aplicações científicas, pois normalmente o seu custo é muito elevado e de difícil acesso num único supercomputador.

Foi então que na década seguinte, dois pesquisadores da NASA (Donald Becker e Thomas Sterling) criaram o rascunho do que viria a ser o modelo de *cluster* dos próximos anos, ao qual iria contra o conceito de uso de máquinas potentes. O modelo inicial foi criado como protótipo com base em computadores pessoais e possuía 16 processadores DX4 agregados por dois canais Ethernet. O que garantiu um sucesso surpreendente e impressionante, com utilização na época apenas no âmbito acadêmico, pela NASA e em centros de pesquisa. Esse projeto foi denominado **Beowulf**. [4]

2.1.1. Razões para a utilização de um Cluster

Os *Clusters* são implementados quando temos tarefas críticas, não pode haver falhas nem paragens do serviço, ou quando os dados têm de ser processados o mais rapidamente possível, como é o caso dos Internet Service Providers (ISPs), serviços de e-mail, que precisam de um taxa de disponibilidade elevadíssima e até mesmo um balanceamento de carga equilibrado e escalável.

C*lusters* paralelos têm uma grande importância na indústria cinematográfica para a renderização de gráficos de altíssima qualidade e animações. Por exemplo o Titanic foi renderizado na plataforma de clustering dos laboratórios da Digital Domain.

Por sua vez os *clusters* Beowulf são usados na ciência, engenharia e finanças actuam em projetos de desdobramento de proteínas, dinâmica de fluídos, redes neurais, análise genética, estatística, economia, astrofísica entre outras. Pesquisadores, organizações e empresas estão a utilizar os *clusters* porque necessitam de

escalabilidade, gestão de recursos, disponibilidade ou processamento a nível supercomputacional a um preço disponível. [5]

2.1.2. Tendências e panorâmica actual dos Clusters

Embora os *clusters*, atualmente, já sejam bastante conhecidos e utilizados, a maior parte de suas aplicações são em empresas que lidam com pesquisas na *web* ou com análises que envolvam muitos cálculos simultâneos (normalmente, pesquisas científicas), não sendo, portanto, muito comuns no ambiente corporativo de empresas de média e grande dimensão que não sejam voltadas à TI, que ainda preferem investir os seus recursos na compra de equipamentos de grande dimensão e de qualidade reconhecida, de marcas como Dell, HP, IBM entre outras.

Porém, com o advento da virtualização, é provável que esse panorama, aos poucos, comece a mudar, uma vez que o *cluster* se torna extensível, não sendo mais necessário um *cluster* para cada serviço. Um único *cluster* pode, dessa forma, hospedar vários ou até mesmo todos os serviços necessários à entidade, em sistemas operativos diferentes, contudo respeitando o aumento do processamento e disponibilidade da tecnologia.

Um outro fator positivo para o aumento na utilização dos *clusters* é o custo para as empresas, tendo em conta que os computadores pessoais estão cada vez mais baratos, por outro lado os computadores profissionais têm preços mais elevados, o que faz com que as empresas adquiram PCs mais baratos em vez de fazerem grandes investimentos num supercomputador que poderia até ser subutilizado.

Por outro lado, a computação distribuída ainda é vista com algum receio por muitos profissionais das TI, por não conhecerem bem os seus benefícios, ou então por não terem o conhecimento teórico necessário para perceber as suas vantagens.

Um outro aspecto negativo é o trabalho inicial que tem de ser feito para montagem dum *cluster* podendo este posteriormente ser usado por uma entidade. Por vezes é mais simples encontrar uma solução já existente, ou seja, um servidor já montado por uma marca fiável com redundância já instalada, para as necessidades do cliente. Num cluster é preciso lidar com vários equipamentos ao mesmo tempo para solucionar as necessidades do cliente, além de tempo é necessário conhecimentos na área de *cluster computing*.

Portanto, embora a tecnologia de *clustering* seja bastante benéfica em vários pontos para qualquer corporação, o seu crescimento/alastramento tende a ser lento, seja pela credibilidade que algumas marcas de equipamentos conseguiram (com justiça) obter junto aos seus clientes em relação aos seus equipamentos de elevada computação, seja pela falta de conhecimento e/ou tempo da equipa de TI da entidade para propor (ou montar) um *cluster* ou ainda, pela falta de informação da entidade como um todo sobre esta tecnologia. [6]

2.2. Tipos de Cluster

Existem muitas aplicações tais como sistemas meteorológicos, ferramentas de mapeamento genético, simuladores geotérmicos, programas de renderização de imagens tridimensionais e tantos outros que para obterem resultados bons resultados recorrem às soluções de *clustering*.

Com a evolução desta área há a possibilidade duma infraestrutura tecnológica remota ofereça essa solução, sendo um serviço de *Cloud Computing*.

Na maior parte dos casos que foram referidos a solução *cluster* é bastante viável, desde que o seu tipo seja bem escolhido.

Existem vários tipos de *clusters*, mas os principais são: cluster de alto desempenho, cluster de alta disponibilidade e cluster de balanceamento de carga.

2.2.1. Cluster de Alto Desempenho (High Performance Computing Cluster)

Este tipo de *clusters* são usados em soluções de exigência máxima onde é requerido o máximo poder computacional disponível, realizando as operações no menor tempo possível, como é o caso de pesquisas cientificas, onde necessitam de analisar grandes quantidades de dados ou efectuar cálculos complexos.

O objectivo destes *clusters* é dar resultados satisfatórios em tempo aceitável e isso engloba centenas de milhares de *gigaflops,* ou seja, milhares de operações de cálculo por segundo. [3]

2.2.2. Cluster de Alta Disponibilidade (High Availability (HA) ou Failover Computing Cluster)

Este tipo de *clusters* o principal foco é ter sempre a aplicação activa, não sendo aceitável que o sistema deixe de funcionar, contudo se não houver volta a dar que a paragem seja a menor e que o sistema continue operacional mesmo que perca a sua performance. É o caso de soluções com objectivos críticos que exigem disponibilidade de serviço de 99,999% durante o ano.

Para conseguir solucionar esta exigência e garantir tal operabilidade, estes *clusters* de alta disponibilidade têm ferramentas de monitorização que identificam nós defeituosos ou com falhas na ligação, redundância de sistemas e computadores reserva para substituição imediata, caso haja máquinas com problemas e precisem de ser trocadas e o curso a geradores de energia caso haja cortes de corrente temporários.

É tolerável que o sistema em casos excepcionais perca performance, mas inadmissível que o sistema pare por completo podem contribuir para prejuízos para as empresas. [3]

A figura 2 demonstra um *cluster* de alta disponibilidade na qual os clientes da parte superior da imagem têm acesso aos dados da parte inferior, tem este sistema caminhos redundantes caso haja uma falha, não quebrando assim a disponibilidade de acesso aos dados.

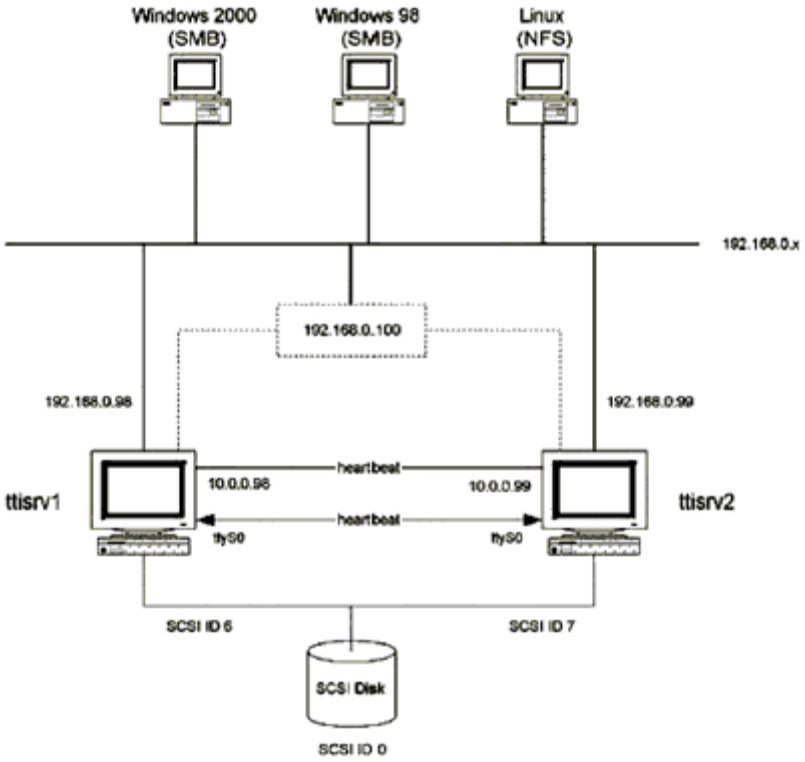

Figura 2. Cluster de Alta Disponibilidade (Adaptado de [5])

Quando mais se precisa dos computadores por vezes à a tendência deles falharem, quem trabalha como administrador de sistemas críticos é normal receber chamadas mesmo de madrugada a informar que o seu sistema ao qual está encarregado para de funcionar ou esteja com problemas, não havendo outra solução a não ser ver o que se passa e compor o problema.

A nossa grande dependência dos computadores leva a que necessitamos de soluções de alta disponibilidade, pois muitas empresas oferecem serviços computacionais como é o caso do *e-business*, notícias, *websites*, base de dados entre outros cujo funcionamento necessita de ser sempre activo.

Esta solução tem o objectivo de ter sempre os seus serviços activos, para isso replica os seus serviços, até mesmo servidores, conseguindo assim uma redundância em *hardware* e *software*.

Com redundância em *software* é possível monitorizar os serviços e caso haja uma falha em algum, sejam redirecionados os trabalhos para um outro activo. A nível de *hardware* costumam-se ser usados sistemas RAID para ter cópia dos dados, fontes de alimentação e placas redundantes, bem como no sistema de ligações de rede tendo caminhos alternativos. [5]

2.2.3. Cluster para balanceamento de carga (Load Balancing (LB))

Neste tipo de *clusters* as tarefas são distribuídas o mais uniformemente possível entre os vários nós que compõem o *cluster*, o principal objectivo é que máquina sirva um pedido e não haja divisão de tarefas.

Um exemplo, um grande *website* na internet recebe mil visitas por segundo e tem um *cluster* com 20 nós a servir os pedidos que chegam, como a solução é de balanceamento de carga, os pedidos são distribuídos de forma equilibrada pelos nós do servidor, ou seja, cada nó atende a 50 pedidos que chegam a cada segundo.

Para tal, não basta dividir as tarefas de igual modo pelos nós, pois pode haver atrasos em certos pedidos e o equilíbrio perde-se, há a necessidade de mecanismos de monitorização da distribuição e quando haja desequilíbrio, possa ser corrigido reencaminhando tarefas para máquinas com mais disponibilidade.

Clusters com este método de trabalho são bastante em várias aplicações que estão disponíveis na internet, pois é uma solução que oferece resposta a um aumento espontâneo do número de pedidos garantindo assim o bom funcionamento do sistema em si. [3]

Os sistemas não trabalham juntos no mesmo processo, mas reencaminham as requisições de forma independente assim que chegam baseados numa escala e num algoritmo próprio.

Esta solução é muito utilizada em serviços de *e-commerce* e ISPs pois há a necessidade de equilibrar os pedidos há medida que chegam aos servidores em tempo real. Um *cluster* deste tipo para que tenha escalabilidade tem que garantir que haja uma total ocupação/uso das suas máquinas. [5]

Há aspectos que devem ser salientados na implementação de um *cluster* de balanceamento de carga, tais como:

- ❖ O algoritmo usado para o balanceamento de carga, tendo em conta o balanceamento entre os servidores e quando o cliente faz a requisição para o endereço virtual (VS), todo o processo de escolha do servidor e resposta do servidor deve ocorrer de modo transparente e imperceptível para o utilizador como se não existisse o balanceamento.
- ❖ Criar um método para verificar se os servidores estão activos e em pleno funcionamento, é vital para que a comunicação não seja redirecionada para um servidor que acabou de ter uma falha.
- ❖ Ter um método para que um cliente tenha acesso ao mesmo servidor quando o desejar.

❖ Balanceamento de carga é mais que um simples redireccionamento do tráfego dos clientes para outros servidores. A sua implementação correta, consiste na verificação permanente da comunicação, verificação dos servidores e redundância.

Todos estes itens são fundamentais para que a escalabilidade do volume de tráfego das redes seja suportada e para que não haja estrangulamento de tráfego ou um ponto único com falha. [5]

Os algoritmos que fazem o balanceamento de carga são indispensáveis e têm a grande importância neste método e contribuem para a optimização e perfeito funcionamento dos *clusters*, as três principais técnicas de divisão de requesições:

❖ **Least Connections**: Esta técnica redireciona os pedidos para o servidor baseado com menos ligações. Por exemplo, se o servidor 1 está controlando atualmente 50 requisições/ligações, e o servidor 2 controla 25, o próximo pedido será automaticamente enviado para o servidor 2, desde que no momento o servidor tenha um número inferior de ligações estabelecidas.

❖ **Round Robin**: Este método usa a técnica de direcionar sempre as requisições para o próximo servidor disponível de uma forma circular. Por exemplo, as conexões de entrada são dirigidas para o servidor 1, depois servidor 2 e por fim servidor 3, depois recomeça no servidor 1.

❖ **Weighted Fair**: Esta técnica faz a divisão de requisições mediante a performance do servidor. Por exemplo, se o servidor 1 é quatro vezes mais rápido no atendimento aos pedidos do que o servidor 2, o administrador vai colocar uma maior carga de trabalho para o servidor 1 do que o servidor 2. [5]

2.2.4. Combinação dos tipos de clusters ou Cluster combinado de Alta Disponibilidade e Balanceamento de Carga

É importante realçar que uma solução final de *cluster* não precisa ser em exclusivo de apenas um tipo. Conforme a necessidade, podem-se combinar características diferentes dos vários tipos no intuito de atender plenamente à solução desejada.

Um bom exemplo para esta solução combinada, pode ser uma loja online que faça uso de um *cluster* de alta disponibilidade para colocar o seu site 24 horas por dia disponível e ao memso tempo ter um outro sistema de balanceamento de carga para dar conta do aumento substancial de pedidos no site por causa de uma promoção. [3]

Algumas características desta plataforma combinada:

- ❖ Redireccionamento dos pedidos de nós com falha para nós reserva;
- ❖ Melhoria na qualidade dos níveis de serviço para as aplicações típicas de rede;
- ❖ Disponibilizar uma arquitetura de *framework* altamente escalável.

A figura 3 demonstra o esquema de um *cluster* combinado destes dois tipos, no qual chegam os pedidos pela internet, seguindo para o VS onde é feito o balanceamento nos atendimentos e de seguida passa para o cluster com alta disponibilidade com caminhos redundantes na rede até chegar aos dados inicialmente pretendidos.

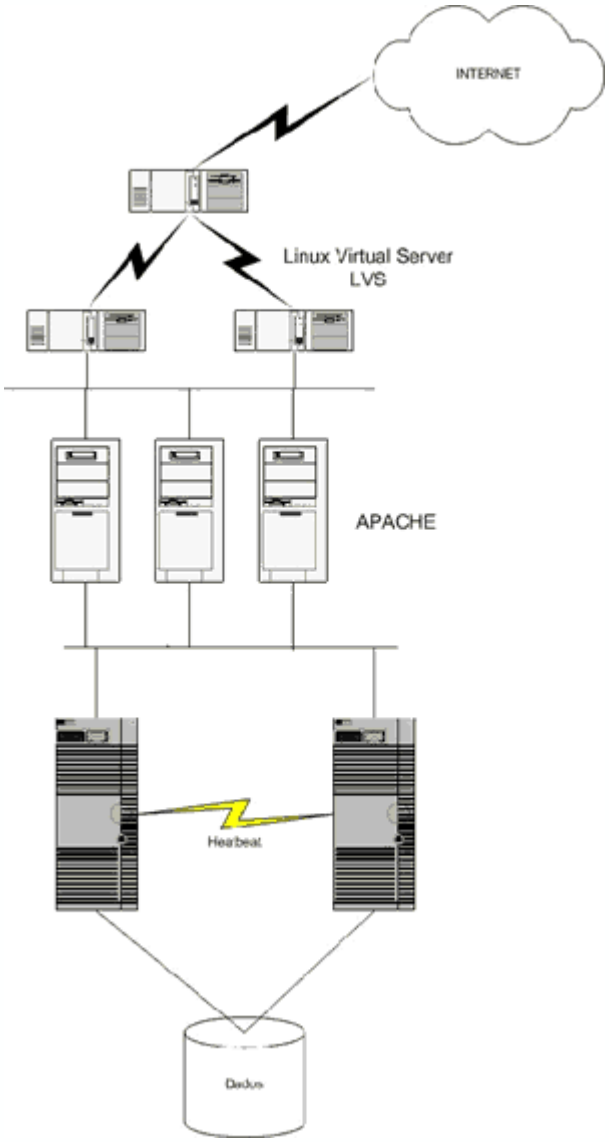

Figura 3. Solução combinada HA + LB (Adaptado de [5])

2.2.5. Processamento distribuído ou paralelo

Com este modelo de computação em *cluster* passamos a ter uma maior disponibilidade e performance para as aplicações em grandes tarefas computacionais.

Uma grande tarefa computacional pode ser dividida em pequenas tarefas que são distribuídas pelos nós do *cluster*, como se fosse um supercomputador massivamente paralelo. É comum associar este tipo de *cluster* ao projeto Beowulf da NASA. Estes *clusters* são usados para computação científica ou análises financeiras, tarefas típicas com exigência de alto poder de processamento. [5]

2.2.6. Cluster Beowulf

Um dos mais notáveis avanços tecnológicos dos dias de hoje, tem sido o crescimento da performance dos PCs, na verdade é que o mercado dos computadores pessoais é maior que o mercado de *workstations*, tornando assim o preço de um PC mais acessível em relação às estações de trabalho tendo muitas vezes um desempenho não muito inferior.

A ideia do *cluster* Beowulf era conseguir uma elevada taxa de processamento para satisfazer diversas áreas científicas com o objetivo de construírem sistemas computacionais poderosos e economicamente viáveis.

Claro que a evolução constante do desempenho dos processadores tem colaborado e muito na aproximação entre PCs e *workstations*, a diminuição dos custos das tecnologias de rede e dos próprios processadores bem como os SOs de código aberto e gratuitos, como é o exemplo do GNU/Linux, influenciam na escolha desta nova filosofia de processamento com alto desempenho.

Uma das características chave de um *cluster* Beowulf, é o *software* utilizado, que é de elevado desempenho e gratuito na maior parte das ferramentas, como por exemplo em GNU/Linux e FreeBSD contêm ferramentas e compatibilidade no processamento paralelo, como é o caso das API's, MPI e PVM. Com isto é possível tornar o SO Linux capaz de correr e suportar aplicações paralelas. [5]

2.2.7. Funcionamento do Cluster Beowulf

Este sistema é dividido num nó controlador chamado de *front-end* (nó mestre), cuja função é controlar o *cluster*, monitorizando e distribuindo tarefas.

Atua também como servidor de ficheiros e faz a ligação dos utilizadores ao *cluster*. Os restantes nós são conhecidos como clientes ou *back-ends* (nós escravos), e são exclusivamente dedicados ao processamento de tarefas enviadas pelo nó controlador, e não existe a necessidade de teclados e monitores, e podem até não ter

discos rígidos tendo *boot* remoto, podendo ser acedidos via *login* remoto (telnet ou ssh). [5]

A figura 4 apresenta o esquema dum *cluster* Beowulf, o tipo que foi usado no desenvolvimento deste projecto, podemos ver o nó controlador (mestre) e os restantes nós escravos que complementam o *cluster*.

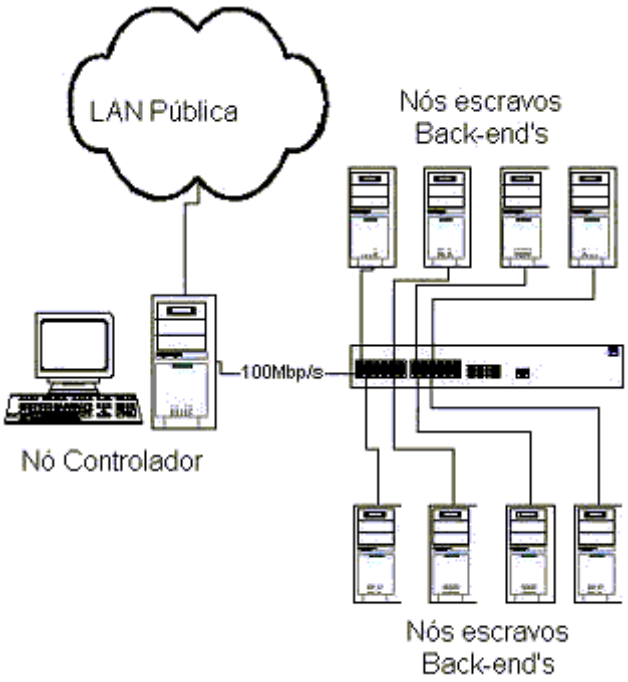

Figura 4. Cluster Beowulf (Adaptado de [5])

O Beowulf foi um projeto bem-sucedido, tendo em conta a opção usada pelos seus criadores em usar *hardware* popular e *software* aberto foi uma aposta ganha pois é possível aplica-lo, replicá-lo e modifica-lo, a prova disso é a elevada quantidade de sistemas construídos com base na arquitectura Beowulf em muitas Universidades, empresas Americanas, Europeias e até mesmo soluções caseiras. Para além do experimento bem-sucedido, foi obtido um sistema de uso prático que continua sendo aperfeiçoado constantemente a cada dia [5]

2.3. Exemplos de Projectos conhecidos

2.3.1. RPiCluster

Este *cluster* foi desenvolvido na "Boise State University" nos EUA, foi um *cluster* realizado recentemente em 2013, com recurso ao *Raspberry Pi*, omo é visível na figura 5.

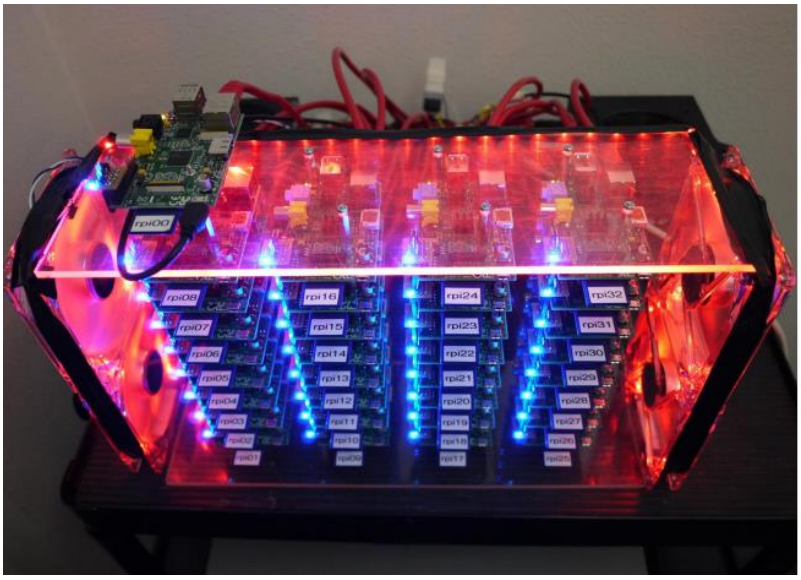

Figura 5. RPiCluster (Adaptado de [7])

Hardware utilizado:

- 32 RPi (*700 MHz core CPU, 512 MB RAM*);
- 1 *Switch* 48 portas, 10/100 Mbps.

Arquitectura:

- Foi usado o Arch Linux ARM (baseado no Ubuntu 11.10 de 32bits).

Software:

- Biblioteca MPICH (transforma um conjunto de máquinas num *cluster*)
- QEMU (Emulador de arquitectura e virtualização), usado para configurar os nós do *cluster* com uma máquina que não era do *cluster*.

O objectivo principal do projecto foi calcular o valor de Pi, com um programa desenvolvido pelo autor do projecto usando o método de 'Monte Carlo', e comparar os resultados com dois *clusters* de maior desempenho também existentes na Universidade.

Os resultados foram bastante bons, pois em relação ao tamanho do *cluster* e à quantidade de processamento os resultados foram bastante bons. [7]

No projecto desenvolvido as configurações e instalações das bibliotecas de processamento paralelo vão ser as mesmas ou praticamente as mesmas, unicamente com outro *hardware* e programas já existentes para fazer o teste do *cluster* calculando também o valor de Pi.

2.3.2. Cluster Beowulf GNU/Linux - Mandriva

Um *cluster* desenvolvido na Universidade Federal de Ouro Preto no Brasil, sobre a plataforma GNU/Linux utilizando a distribuição *Mandriva*.

Hardware utilizado:

- PC1 (*CPU Intel Core 2 Quad, HDD 240GB*) ;
- PC2 (*CPU Intel Core 2 Duo, HDD 240GB*) ;
- PC3 (*AMD X2, HDD 240GB*);
- 1 *Switch D-link* com 16 portas e velocidade de 1Gbps;

Arquitectura:

- Foi usando o Linux – Mandriva de 64bits.

Software:

- Pacotes RPM (*Red Hat Package Manager*): conjunto de programas que permite aos utilizadores executar comandos em máquinas remotas, executar *login* remoto, copiar ou trocar arquivos entre máquinas.
- Pacote PVM (*Parallel Virtual Machine*): habilita os computadores a tornarem-se um *cluster*, com bibliotecas de funções necessárias ao funcionamento de programas paralelos.
- Programa Kdevelop, permite programar em várias linguagens C, C++, Fortran, Perl entre outras. Foi usado para compilar e desenvolver o programa de teste que cálcula o valor de Pi.

O objectivo deste projecto foi a construção do *cluster*, instalação do *software* necessário, configurações necessárias para funcionar em PVM. O projecto documentado será muito semelhante a este desenvolvido, com a excepção de a distribuição usada ser o Ubuntu e não o Mandriva.

No final da configuração foi usado um programa de *chat* para verificar se os escravos estavam operacionais e se todas as configurações estavam funcionais. E por último o teste com o programa desenvolvido pelo autor do projecto. [8]

2.3.3. Cluster Virtal Ayanami

Este *cluster* foi desenvolvido Universidade Federal Rural de Pernambuco, no Brasil. O principal objectivo desta construção em *cluster* foi aproveitar o poder de processamento dos PCs que são usados em CAD (Desenho Assistido por Computador) na instituição.
Foram feitos 2 testes neste projecto, o primeiro avaliando a performance com configurações no *hardware* nativo e o segundo teste com virtualização da VM usando 80% da RAM dos PCs do *cluster*. [9]

Hardware utilizado:

- 8 PC (*CPU AMD X2 3.0GHz, RAM 4 GB DDR III*);
- 1 *Swtich Gigabit Ethernet*

Arquitectura:

- Linux - Ubuntu 11.10 (base);
- Linux - Rocks Cluster 5.4.3 (SO usado na virtualização).

Software:

- VirtualBox 4.1.18 (Software usada para virtualização da VM);
- HPCC (conjunto de ferramentas para avaliação do desempenho).

As semelhanças neste projecto com o desenvolvido são poucas unicamente o SO é igual, o objectivo deles também é diferente do meu.
Foi interessante observar a possibilidade de aproveitar um sistema já existente e virtualizar uma 'Virtual Machine' ou seja um *cluster* virtual e ver como era o seu desempenho, e comprovou-se que é 84,7% inferior. [9]

3. DESENVOLVIMENTO DO CLUSTER BEOWULF@IPG

3.1. Arquitectura do *hardware* do *cluster*

Como demonstra a figura 6 esta foi a arquitectura de *hardware* usada no *cluster* do projecto. Com esta ordem sendo o C1 o computador mestre e os restantes os escravos do *cluster*.

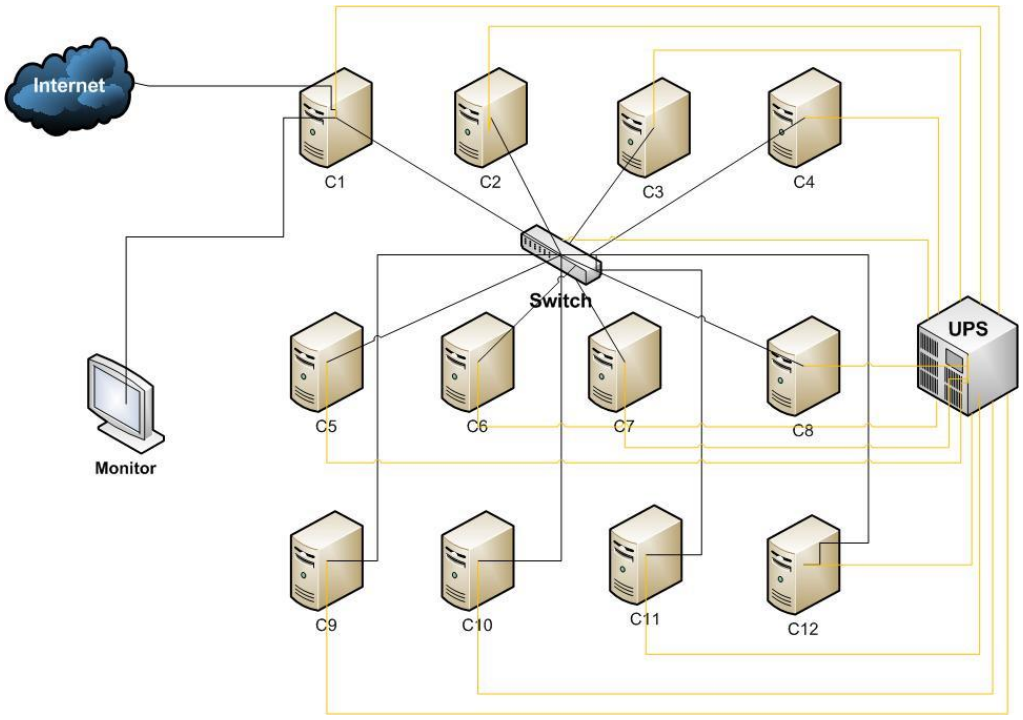

Figura 6. Diagrama da arquitectura do cluster.

O *hardware* utilizado no *cluster*:

- 12 PC que vão de C1 até C12 (*1GB RAM, 40GB HDD, CPU Single Core 1533 Mhz*) - C1 sendo o mestre e os restantes escravos.
- 1 *Switch Enterasys* 24 portas – Interliga os 12 nós do *cluster* na mesma rede.
- 1 UPS - fornece energia ao *cluster* nomeadamente aos 12 PCs e *Switch* caso haja uma quebra de energia temporária.
- 1 Monitor Samsung – serve para visualizar a interface de cada PC e poder interagir com ele.

Arquitectura do *cluster*:

- Foi usado o Linux – Ubuntu 12.04 LTS de 32bits em todos os PCs.

***Software*:**

- Vim editor – usado para editar e fazer configurações em ficheiros Linux.
- Biblioteca OpenMPI - transforma um conjunto de máquinas num *cluster*.
- OpenSSH para se fazer uma comunicação criptografada.
- Putty – software usado para fazer a ligação aos PC's do *cluster* através do meu próprio PC para visualizar/editar/gerir as configurações.

A figura 7.apresenta o *cluster* montado com os seus 12 computadores obsoletos da instituição, reaproveitados assim neste projecto com a criação deste *cluster* Beowulf.

Figura 7. Forma final do Cluster.

3.2. Configuração do Cluster

Nesta secção vão descreve-se todos os passos desde o início do trabalho no *cluster*, até à finalização das configurações.

Os primeiros passos foram verificar se todos os computadores estavam funcionais, sendo que 4 deles não funcionavam correctamente 2 deles com problemas na PSU (fonte de alimentação) na qual se trocou por uma funcional, outro estava com problemas numa das memórias a qual foi também substituída por uma funcional o último era problema no disco não arrancava o sistema, foi também substituído.

Depois de ter todos os computadores funcionais prosseguiu-se à instalação do sistema operativo Ubuntu em cada máquina do *cluster*.

3.2.1. Instalação do SO Ubuntu

Começou-se por instalar SO Ubuntu 12.04 LTS nas 12 máquinas que foram usadas no *cluster*. A instalação realizada começa com o menu da figura 8.

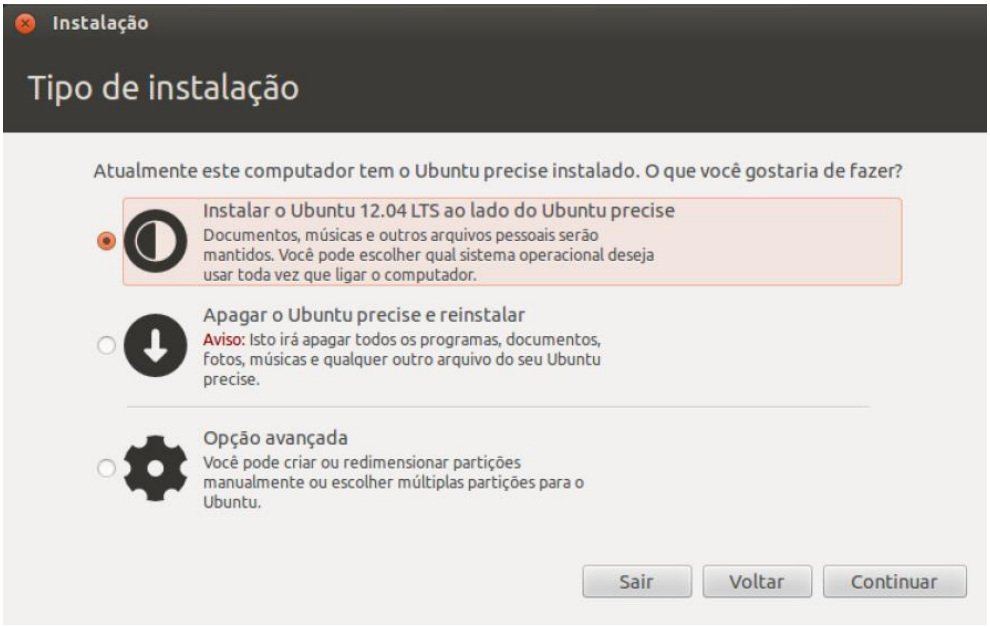

Figura 8. Instalação do Ubuntu.

Depois de concluída a instalação do SO Ubuntu, actualizou-se o sistema com os seguintes comandos:

#sudo apt-get update
#sudo apt-get upgrade

3.2.2. Configuração dos IPs estáticos

Começando a configuração dos IPs estáticos nas duas placas de rede do PC C1 que será o computador mestre como apresenta a figura 9.

Endereço IP - de forma genérica, é uma identificação de um dispositivo (computador, impressora, etc) em uma rede local ou pública. Cada computador na internet possui um IP (Internet Protocol) único, que é o meio em que as máquinas usam para se comunicarem na Internet.

Mask - a máscara de sub rede, também conhecida como *subnet mask* ou *netmask*, é um número de 32 bits usado num IP para separar a parte correspondente à rede pública, à sub rede e aos *hosts*.

Gateway (GW) - no geral é um computador ou router que fica entre duas redes. A tradução da palavra é "portal", sendo exatamente assim que um *gateway* funciona. Fazendo a ligação de uma rede local a outra rede geralmente a WAN (Internet).

Na placa *Eth0* terá a ligação à internet com o IP 192.168.15.30 *mask* 255.255.255.240 (/20) e o *gateway* (GW) 192.168.15.254.

Na outra placa de rede *Eth1* que será a placa ligada à rede *cluster* terá um endereço dentro da rede local 10.10.20.1 com a *mask* 255.255.255.0 (/24).

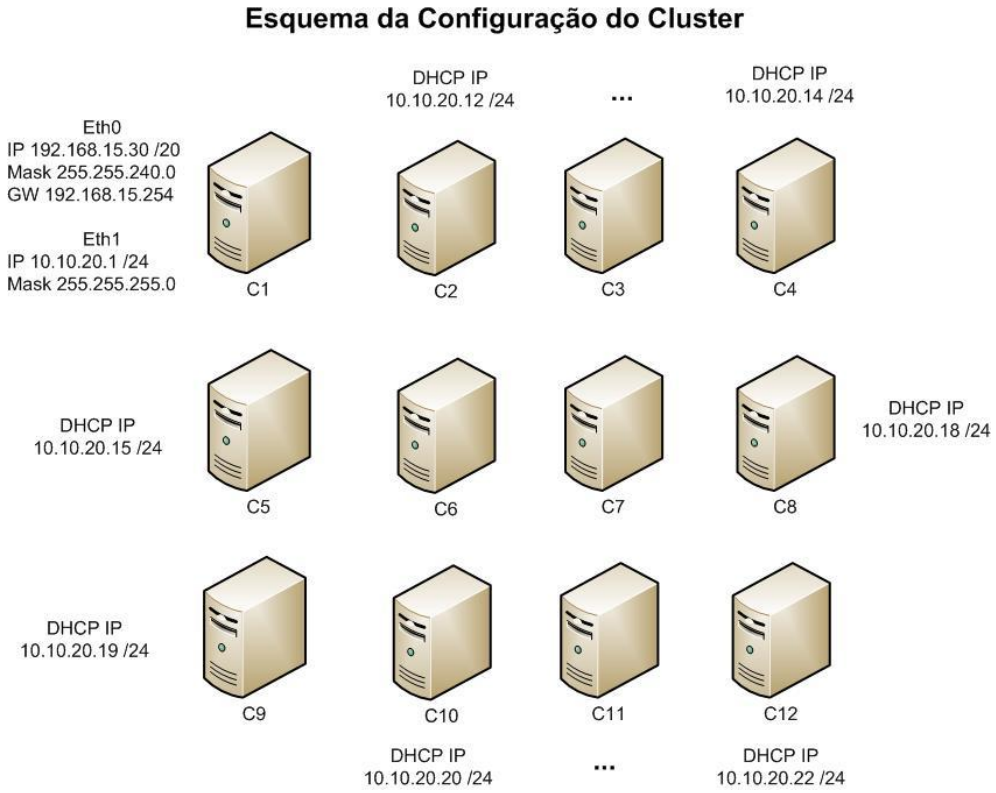

Figura 9. Esquema de endereçamento de IPs

Aceder às interfaces de rede através do editor vim com o comando:

#sudo vim /etc/network/interfaces

Depois adicionar no PC Mestre C1 na placa Eth0 o IP da rede WAN com o respectivo GW da rede e na *Eth1* o IP da rede *cluster*, como mostra a figura 10.

```
# The loopback network interface

auto lo
iface lo inet loopback

#the primary network interface
#WAN
#auto eth0
#iface eth0 inet dhcp

#ETH0
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
address 192.168.15.30
netmask 255.255.240.0
#network 192.168.0.0
#broadcast 192.168.15.255
gateway 192.168.15.254

#LAN (Vai servir de DHCP para os clientes)
auto eth1
iface eth1 inet static
address 10.10.20.1
netmask 255.255.255.0
```

Figura 10. Configuração das interfaces Eth0 e Eth1

Os restantes endereços serão atribuídos por DHCP função configurada a seguir no PC mestre para atribuir de forma dinâmica endereços aos escravos, computadores restantes, dando assim a vantagem na ocorrência de uma falha de um PC possa ser substituído e logo que o seja receberá um novo endereço pelo protocolo DHCP configurado.

3.2.3. Configuração Protocolo DHCP

DHCP - (Protocolo de configuração dinâmica de *host*), é um protocolo de serviço TCP/IP que oferece configuração dinâmica de computadores, com endereços IP, máscara de sub-rede e *default gateway*.

De seguida efectuou-se a configuração do protocolo DHCP no C1 computador mestre com os seguintes passos:

1. Instalar o DHCP-Server através do comando:
   #sudo apt-get install isc-dhcp-server

2. Depois editar a interface (Eth1) que vai distribuir os endereços DHCP através da edição do script dhcp-server com comando:
   #sudo vim /etc/default/isc-dhcp-server

Como demonstra a figura 11, com este serviço os PC's escravos ficam confinados logo que são iniciados com um endereço IP, ficando com o primeiro IP disponível inferior da *pool* definida, no próximo incrementa +1.

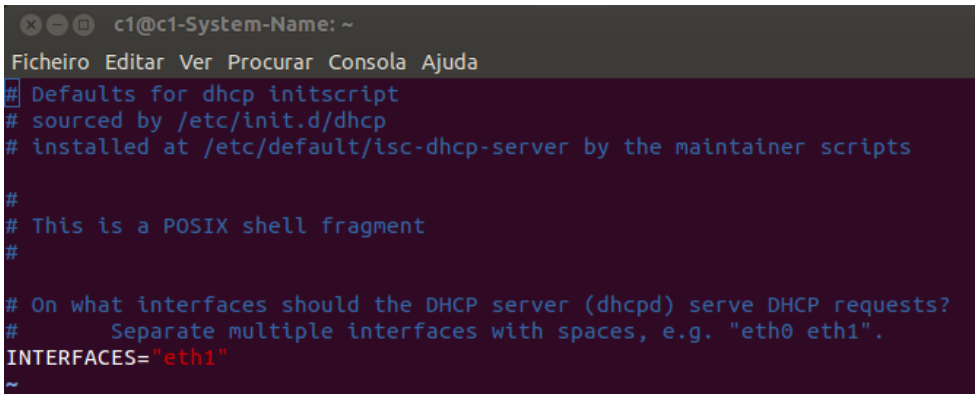

Figura 11. Editar a Interface que distribui DHCP

3. O passo seguinte foi definir a variação de endereços do protocolo DHCP com o comando:
   #sudo vim /etc/dhcp/dhcpd.conf

Alterando o ficheiro dhcpd.conf mudando o domain-name-servers para '8.8.8.8', depois mudar a variação de IP, começando no X:X.X.11 e terminando no X.X.X.30, ficando o PC mestre a funcionar como *router* com o IP fixo 10.10.20.1 como mostra a figura 12.

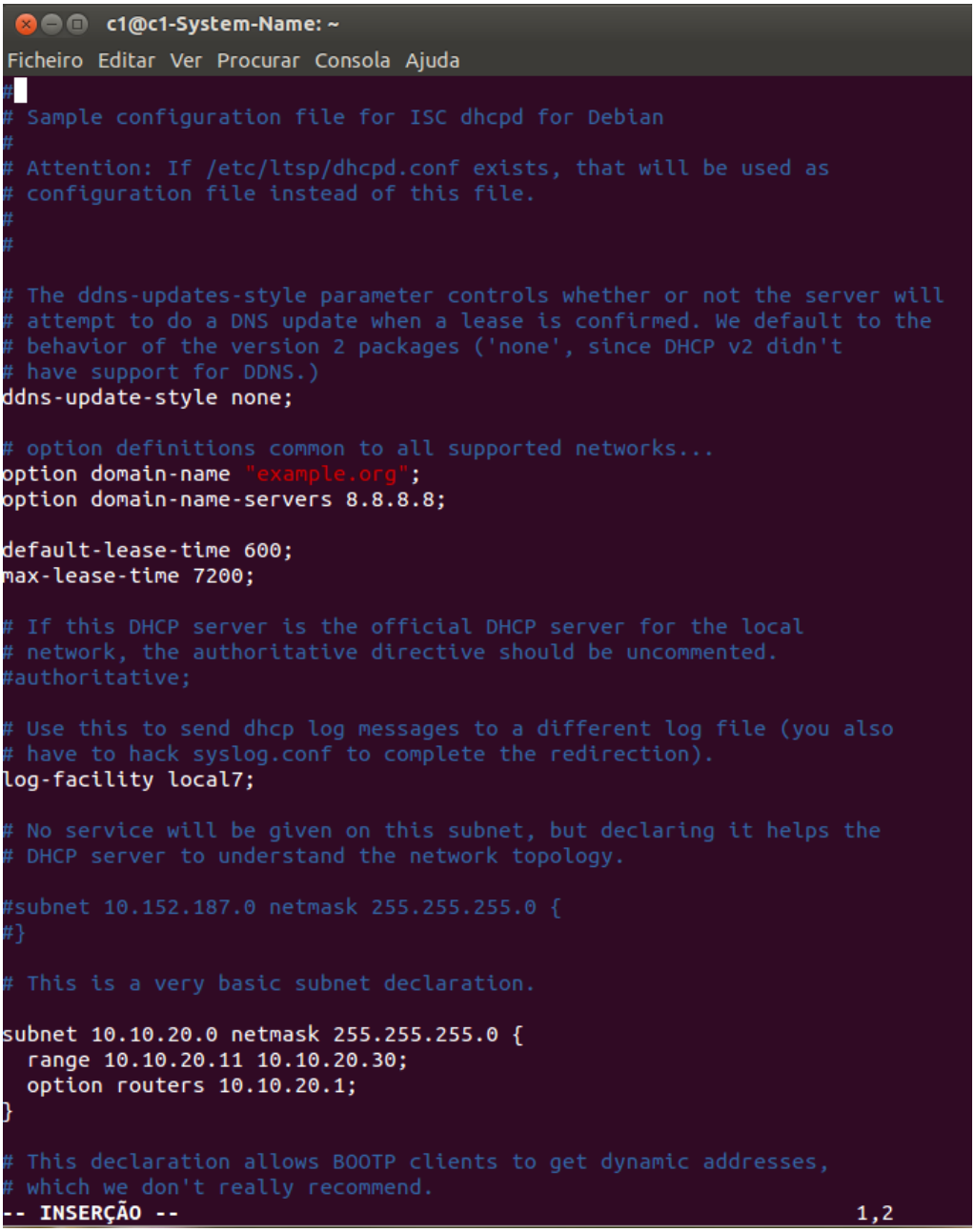

Figura 12. Edição do ficheiro dhcpd.conf

4. Por fim reiniciamos o serviço DHCP para ficar operacional com o comando: #sudo /etc/init.d/isc-dhcp-server restart

3.2.4. Configuração do IP Forward

IP Forwarding - é necessário quando se trabalha com duas interfaces Ethernet, quando precisamos configurar o sistema como um *router* ou *gateway*, também quando o sistema precisa funcionar como servidor VPN, e para partilhar a ligação de internet.

Continuando a configuração no computador mestre C1, com os seguintes etapas:

1. Vamos aceder ao serviço IP Forward para isso foi usado o comando:
   #sudo vim /etc/sysctl.conf

2. De seguida com o ficheiro aberto habilitamos o serviço com a alteração do valor de '0' para '1' como demonstra a figura 13.

```
# Aqui (Enabling IP Forwarding)
net.ipv4.ip_forward=1
```

Figura 13. Habilitando o serviço IP Forward

NAT - também conhecido como *masquerading* é uma técnica que consiste em reescrever, fazendo uso de uma tabela *hash*, os endereços IP de origem de um pacote que passam por um *router* ou *firewall* de maneira a que um computador de uma rede interna tenha acesso ao exterior.

3. Vamos aceder ao ficheiro de NAT para alterar as tabelas de NAT adicionando 2 novas linhas para registar os endereços de todos os outros computadores do *cluster* que usem o serviço para sair da rede local privada do *cluster* para a LAN e Internet. Foi usado o comando para aceder ao ficheiro rc.local:
   #sudo vim /etc/rc.local

4. Aqui dentro do ficheiro como vemos na figura 14, adicionamos as duas linhas a branco e no final usamos o comando do vim para guardar e sair:
   :wq (write and quit)

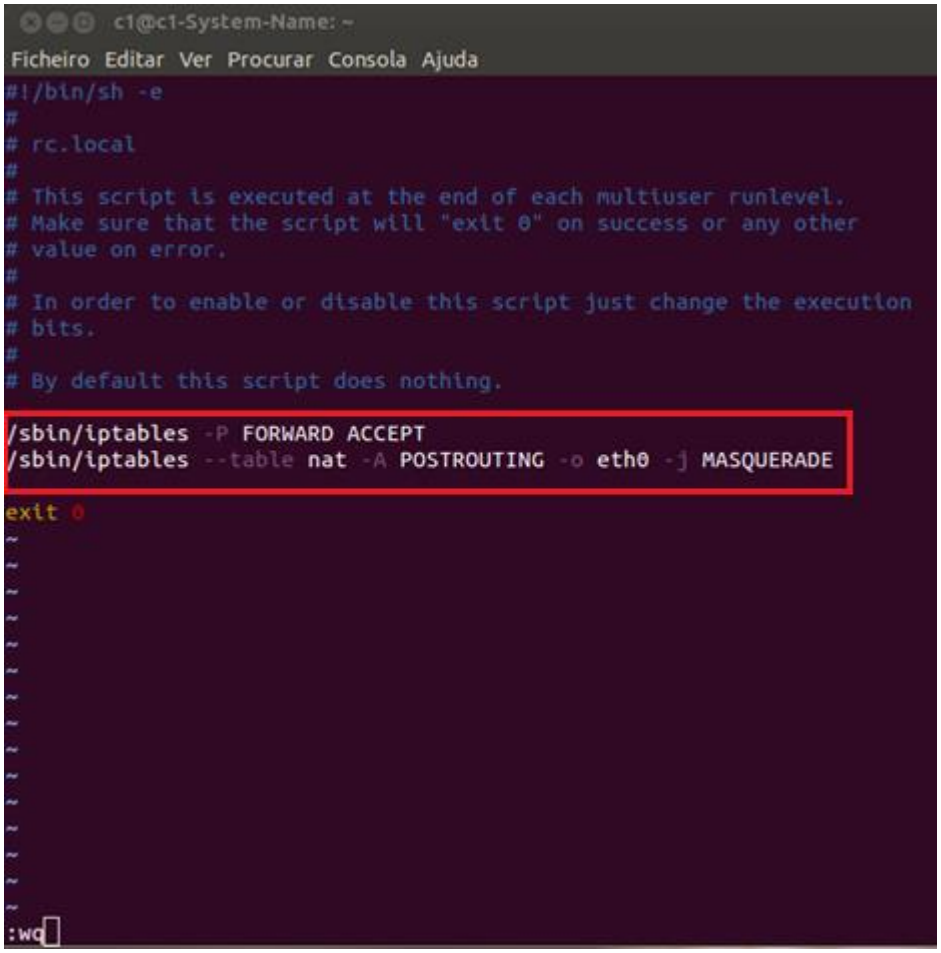

Figura 14. Adicionadas as linhas a branco.

5. Em seguida foi habilitado o serviço IP Forward com o comando:
   #sudo sysctl –w net.ipv4.ip_forward=1

6. Continuamos com a activação das tabelas de NAT com o seguinte comando:
   #sudo iptables --tables nat –A POSTROUTING –o eth0 –j MASQUERADE

7. Por fim fazemos o *restart* à rede no computador C1 (mestre) para colocar tudo operacional com o seguinte comando:
   #sudo /etc/init.d/networking restart

3.2.5. Instalar e Configurar NFS em todos os nós

NFS – é um sistema de ficheiros distribuídos desenvolvido inicialmente pela Sun Microsystems Inc. a fim de partilhar ficheiros e diretórios entre computadores ligados na mesma rede, formando assim um diretório virtual.

Foi feita a instalação e configuração em 3 passos:

- **Passo 1:**

No mestre (C1) foi instalado o nfs-server:

```
# apt-get install nfs-kernel-server nfs-common
```

Em seguida nos nós escravos instalou-se o nfs:

```
# apt-get install nfs-common
```

- **Passo 2:**

No nó mestre (C1) foi criado o directório '/dados' que foi partilhado na rede e usado para albergar todos os ficheiros que o *cluster* usa:

```
# mkdir /dados
```

De seguida foi editado o ficheiro 'exports' para exportar o directório '/dados' e para que fique disponível para montagem a partir dos restantes nós, acrescentado a linha circundada na figura 15:

```
# vim /etc/exports
```

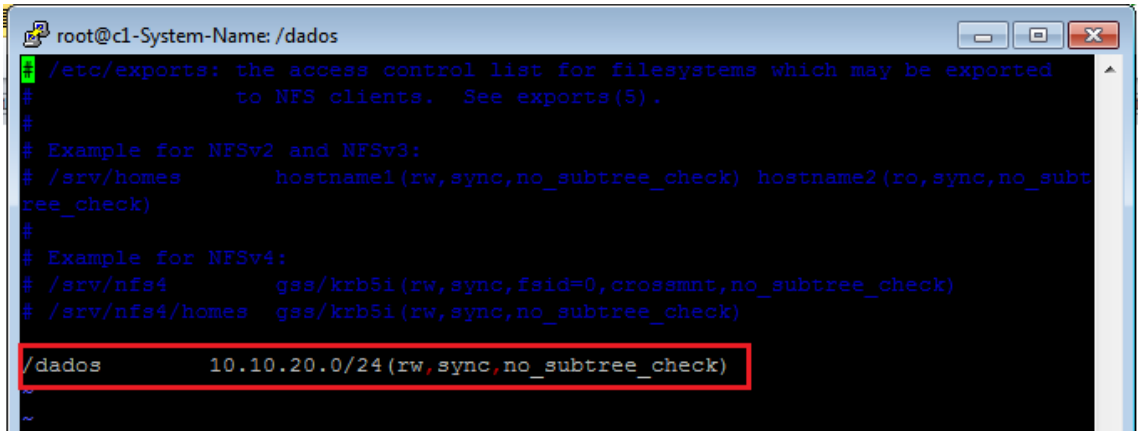

Figura 15. Edição ficheiro exports.

Por último foi reiniciado o serviço NFS no mestre para que as configurações surtam efeito:

```
# /etc/init.d/nfs-kernel-server restart
```

- **Passo 3:**

Para montar o diretório '/dados' nos nós escravos, precisámos criá-lo em cada nó escravo:

```
# mkdir /dados
```

De seguida foi editado o ficheiro '/etc/fstab' para que o diretório seja montado automaticamente durante o arranque de cada nó escravo com a seguinte linha assinalada na figura 16:

```
# gedit /etc/fstab
```

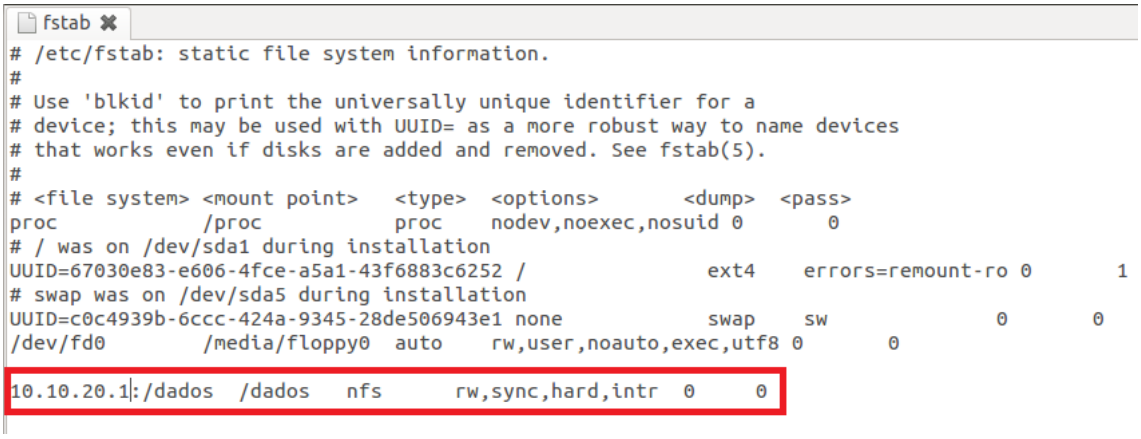

Figura 16. Edição do ficheiro fstab.

Em seguida foi executado o comando *mount* que fará a releitura do ficheiro *fstab* em todos os nós montando assim o diretório '/dados' imediatamente:

```
# mount –a
```

Verificar se o directório foi realmente bem montado com o comando:

```
# df –h
```

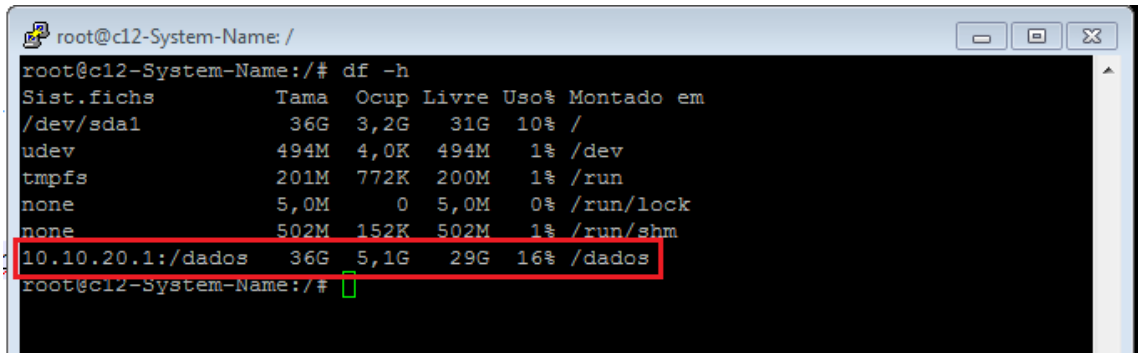

Figura 17. Resultado do mount.

Realizar testes de leitura do diretório '/dados' de todos os nós escravos com o comando:

```
# ls -la /dados/
```

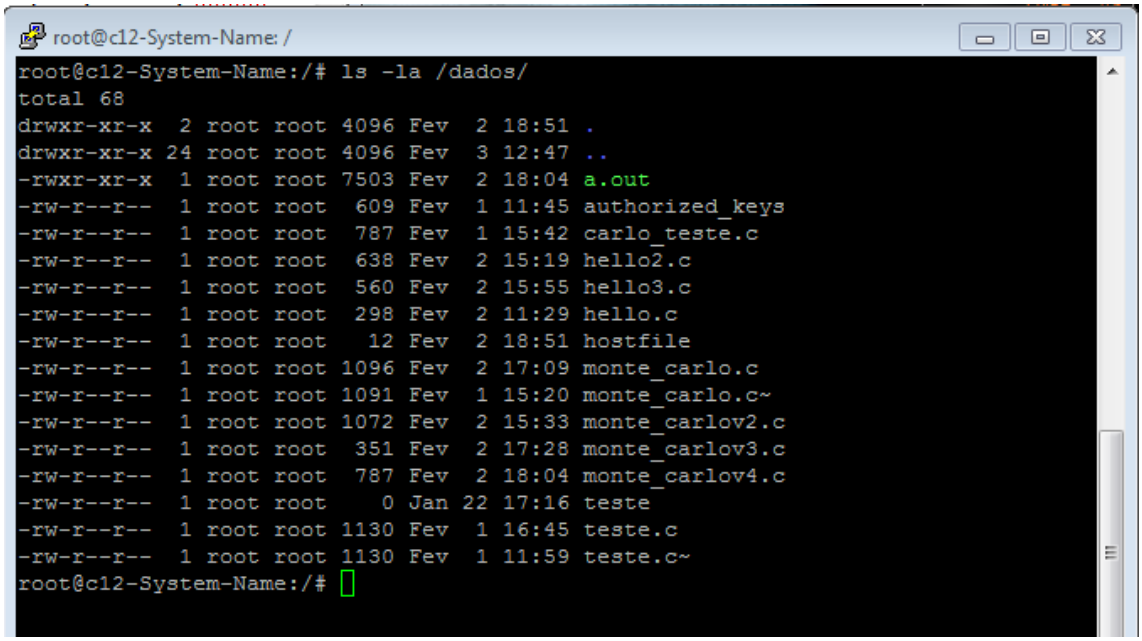

Figura 18. Listagem de /dados em C12.

3.2.6. Sincronização da data e hora

Em seguida acertou-se a data e hora em todos os nós do *cluster* usando o servidor de NTP 'a.ntp.br' com o seguinte comando:

```
# apt-get install ntpdate
# ntpdate a.ntp.br
```

O resultado da hora sincronizada ao segundo com este protocolo pode ser confirmado com o comando:

```
# date
```

3.2.7. Instalar OpenSSH

OpenSSH - Open Secure Shell é uma coleção de programas de computador que oferecem comunicação em sessões criptografadas numa rede de computadores usando o protocolo SSH. Ele foi criado usando código aberto alternativo ao código proprietário da *suíte de softwares Secure Shell* oferecido pela SSH *Communications Security*. [10]

Esta configuração foi feita em 2 passos:

- **Passo 1:**

Começamos por instalar em todos os nós do *cluster* o openssh-server. É necessário para que o nó mestre controle os escravos. Para isso foi usado o seguinte comando:

```
#sudo apt-get install openssh-server
```

Depois de instalado foram geradas as chaves no mestre para serem usadas na comunicação do *cluster* entre o nó mestre e os escravos:

```
# ssh-keygen -t dsa
```

De seguida foram copiadas as chaves para o directório comum no *cluster* '/dados':

```
# cp /root/.ssh/authorized_keys /dados
```

- **Passo 2:**

Em cada nó escravo foi criado o directório '.ssh' e copiadas as chaves geradas anteriormente e guardadas em '/dados' no mestre:

```
# mkdir /root/.ssh
# cp /dados/authorized_keys /root/.ssh/
```

3.2.8. Instalação e Configuração OpenMPI

Ao instalar estas bibliotecas em todos os nós teremos um *cluster* senão eram simples PCs ligados na mesma rede. Assim podemos depois usar aplicações que façam uso da computação paralela ou seja em *cluster*.

1. Usando este comando para instalar a *OpenMPI* em todos os PCs do *cluster*:
   ```
   # apt-get install openmpi-bin libopenmpi-dev
   ```

A configuração da *OpenMPI* no computador mestre é necessária para que a máquina mestre reconheça as máquinas escravas onde vai correr os programas para realizar a computação paralela.

2. Para isso é necessário criar o ficheiro que armazene essa configuração chamar-se-á '*hostfile'*:
   ```
   # cd /dados
   # vim hostfile
   ```

3. No ficheiro deverá conter os seguintes dados:

Localhosts slots = 11

C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12

Depois de feito isto o *cluster* ficou configurado. Pode se testar com um programa simples '*Hello World*' devolvendo a posição de cada nó total de nós existente no *cluster*:

Programa hello.c

```
#include "mpi.h"
#include <stdio.h>

int main( argc, argv )
int argc;
char **argv;
{
       int rank, size;
       MPI_Init( &argc, &argv );
       MPI_Comm_rank( MPI_COMM_WORLD, &rank );
       MPI_Comm_size( MPI_COMM_WORLD, &size );
       printf( "Hello world! I'm %d of %d\n",rank, size );
       MPI_Finalize();
       return 0;
}
```

4. Para compilar o programa foi usado o seguinte comando dentro da pasta comum ao cluster '/dados':
   # mpicc hello.c

5. Foi usado o seguinte comando para executar o programa no *cluster,* com '*np'* sedo o parâmetro de nós escravos a ser passado '*--hosfile'* o parâmetro de entrada com o nome do ficheiro contendo as máquinas escravas do cluster.
   # mpirun -np 6 --hostfile hostfile ./a.out

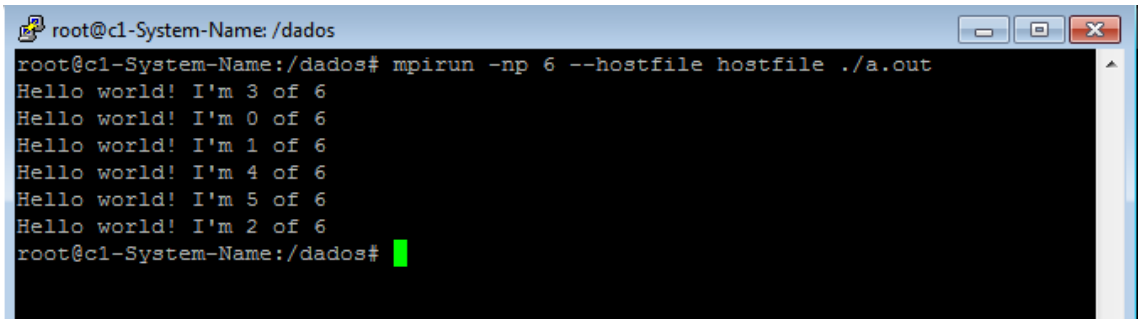

Figura 19. Resultado do programa hello.c

Com esta configuração realizada o *cluster* ficou pronto para ser testado com o programa em C calculando o valor de Pi usando o método Monte Carlo.

3.3. Programação paralela para o Cluster

As aplicações são vistas como um conjunto de programas que são executados de forma independente em diferentes processadores de diferentes computadores. A semântica da aplicação é mantida por troca de informações entre os vários programas.

A sincronização e o modo de funcionamento da aplicação é da responsabilidade do programador. No entanto, o programador não quer desperdiçar muito tempo com os aspectos relacionados com a comunicação propriamente dita.

Existem diferentes bibliotecas que implementam os pormenores da comunicação. Essas bibliotecas permitem executar programas remotamente, monitorizando o seu estado, e trocar informação entre os diferentes programas, sem que o programador precise de saber explicitamente como isso é conseguido. [11]

3.3.1. Message Passing Interface (MPI)

O que não é o MPI:

- O MPI não é um modelo revolucionário de programar máquinas paralelas. Pelo contrário, ele é um modelo de programação paralela baseado na troca de mensagens que pretendeu recolher as melhores funcionalidades dos sistemas existentes, aperfeiçoá-las e torná-las um *standard*.
- O MPI não é uma linguagem de programação. E um conjunto de rotinas (biblioteca) definido inicialmente para ser usado em programas C ou Fortran.
- O MPI não é a implementação é apenas a especificação.

Principais objectivos:

- Aumentar a portabilidade dos programas;
- Aumentar e melhorar a funcionalidade;
- Conseguir implementações eficientes numa vasta gama de arquitecturas;
- Suportar ambientes heterogéneos.

Iniciar e Terminar o ambiente de execução do MPI:

- MPI_Init() inicia o ambiente de execução do MPI.
- MPI_Finalize() termina o ambiente de execução do MPI.
- Todas as funções MPI retornam 0 se Ok, valor positivo se erro.

Estrutura Base de um Programa MPI:

```
// incluir a biblioteca de funções MPI
#include <mpi.h>
...
main(int argc, char **argv) {
...
// nenhuma chamada a funções MPI antes deste ponto
MPI_Init(&argc, &argv);
...
MPI_Finalize();
// nenhuma chamada a funções MPI depois deste ponto
...
}
```

3.3.2. Comunicadores

Uma aplicação MPI vê o seu ambiente de execução paralelo como um conjunto de grupos de processos.

O comunicador é a estrutura de dados MPI que abstrai o conceito de grupo e define quais os processos que podem trocar mensagens entre si. Todas as funções de comunicação têm um argumento relativo ao comunicador.

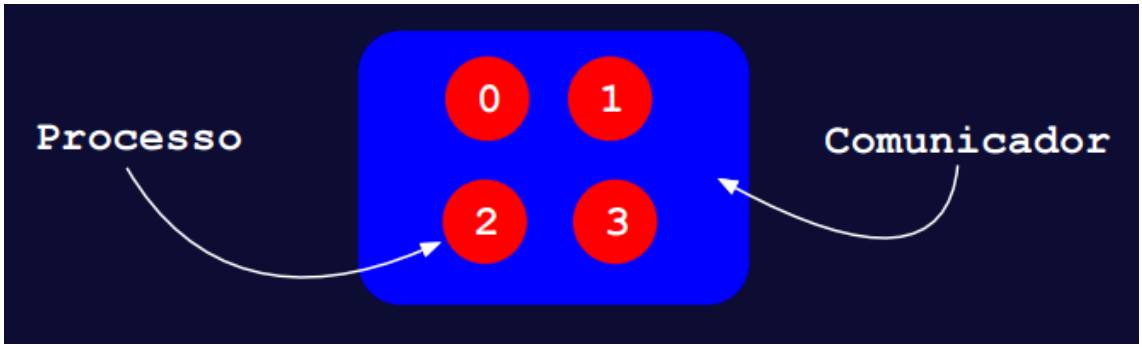

Figura 20. Esquema de Comunicador. (Adaptado de [11]).

Por defeito, o ambiente de execução do MPI define um comunicador universal (MPI COMM WORLD) que engloba todos os processos em execução.

Todos os processos possuem um identificador único (*rank*) que determina a sua posição (de 0 a N-1) no comunicador. Se um processo pertencer a mais do que um comunicador ele pode ter *rankings* diferentes em cada um deles. [11]

Informação Relativa a um Comunicador:

- `MPI_Comm_rank(MPI_Comm comm, int *rank)`

MPI_Comm_rank() devolve em *rank* a posição do processo corrente no comunicador *comm*.

- `MPI_Comm_size(MPI_Comm comm, int *size)`

MPI_Comm_size() devolve em *size* o total de processos no comunicador *comm*.

3.3.3. Mensagens MPI

Na sua essência, as mensagens não são nada mais do que pacotes de informação trocados entre processos. Para efectuar uma troca de mensagens, o ambiente de execução necessita de conhecer no mínimo a seguinte informação:

- Processo que envia;
- Processo que recebe;
- Localização dos dados na origem;
- Localização dos dados no destino;
- Tamanho dos dados;
- Tipo dos dados.

O tipo dos dados é um dos itens mais relevantes nas mensagens MPI. Daí, uma mensagem MPI ser normalmente designada como uma sequência de tipo de dados. [11] A figura 21 apresenta os vários tipos disponíveis e usados nas mensagens MPI.

MPI	C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_PACKED	

Figura 21. Tipos de dados básicos (Adaptado de [11]).

3.3.4. Mensagens colectivas (Reduce)

```
MPI_Reduce(void *sendbuf, void* recvbuf, int count, MPI_Datatype
datatype, MPI_Op op, int root, MPI_Comm comm)
```

- `MPI_Reduce()` permite realizar operações globais de resumo fazendo chegar mensagens de todos os processos a um único processo no comunicador.
- `sendbuf` é o endereço inicial dos dados a enviar.
- `recvbuf` é o endereço onde devem ser colocados os dados recebidos (só é importante para o processo *root*).
- `count` é o numero de elementos do tipo *datatype* a enviar.
- `datatype` é o tipo de dados a enviar.
- `op` é a operação de redução a aplicar aos dados recebidos.
- `root` é a posição do processo, no comunicador *comm*, que recebe e resume os dados.
- `comm` é o comunicador dos processos envolvidos na comunicação

O objectivo da função é recolher os dados dos nós do *cluster* e devolve-los ao *root* (PC mestre) da maneira que o programador o desejar, como pode ser visto na figura 22.

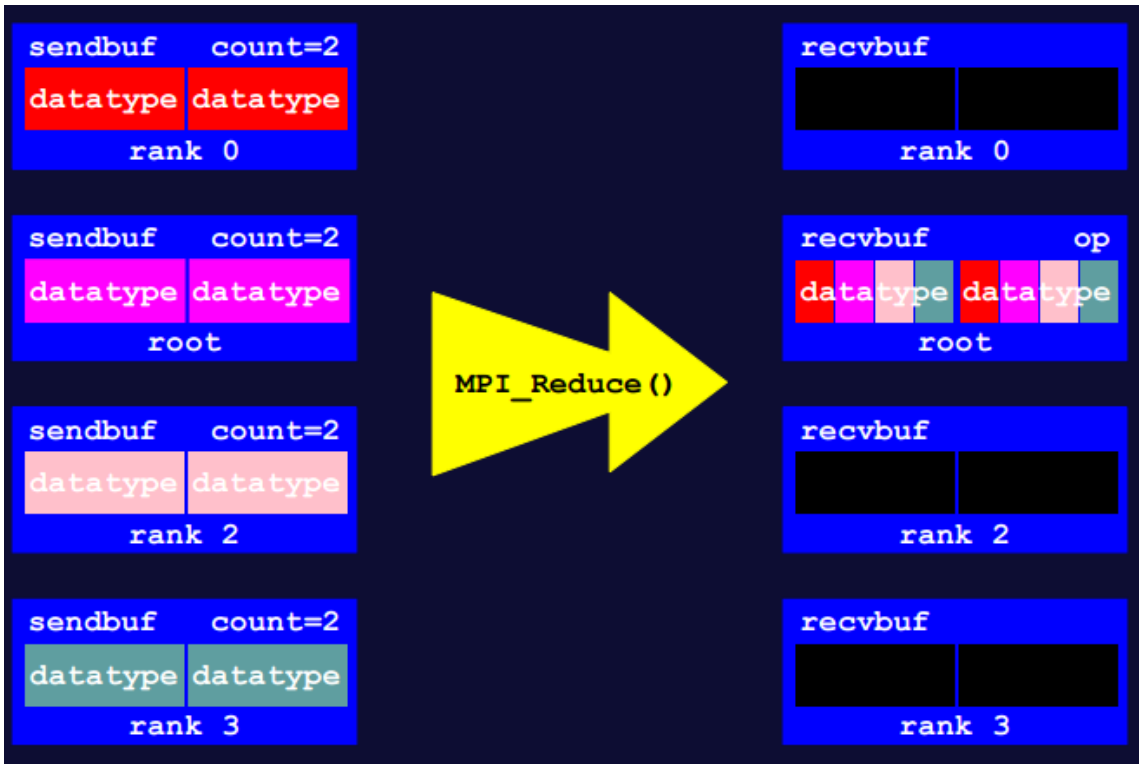

Figura 22. Esquema de funcionamento da função MPI_Reduce() (Adaptado de [11]).

Podem ser feitas operações aritméticas em passagem de mensagens caso o programador assim o deseje, as operações são apresentadas abaixo na figura 23:

Operação	Significado
MPI_MAX	máximo
MPI_MIN	mínimo
MPI_SUM	soma
MPI_PROD	produto
MPI_LAND	e lógico
MPI_BAND	e dos bits
MPI_LOR	ou lógico
MPI_BOR	ou dos bits
MPI_LXOR	ou exclusivo lógico
MPI_BXOR	ou exclusivo dos bits

Figura 23. Operações da Função MPI_Reduce (Adaptado de [11]).

4. TESTES E RESULTADOS

4.1. Cálculo do valor de PI pelo método de Monte Carlo

Para testar o bom funcionamento do *cluster* foi usado um programa de computação paralela (adaptado de [12]) que implementa o método de Monte Carlo para calcular o valor de Pi. Este método que é explicado a seguir, permite calcular o valor aproximado de Pi através de um elevado número de cálculos. A implementação em computação paralela do método permite distribuir os cálculos pelos vários processadores do cluster permitindo assim diminuir drasticamente o tempo de execução do método.

A fórmula que o método usa é a demonstrada no esquema da figura 24, a ideia é a seguinte:

- Gerar N pontos aleatórios (x, y).
- Para cada ponto (x, y) verificar se $x * x + y * y < 1$ e em função disso incrementar in ou out.
- Calcular o valor aproximado de π como $4 * in/(in + out)$.

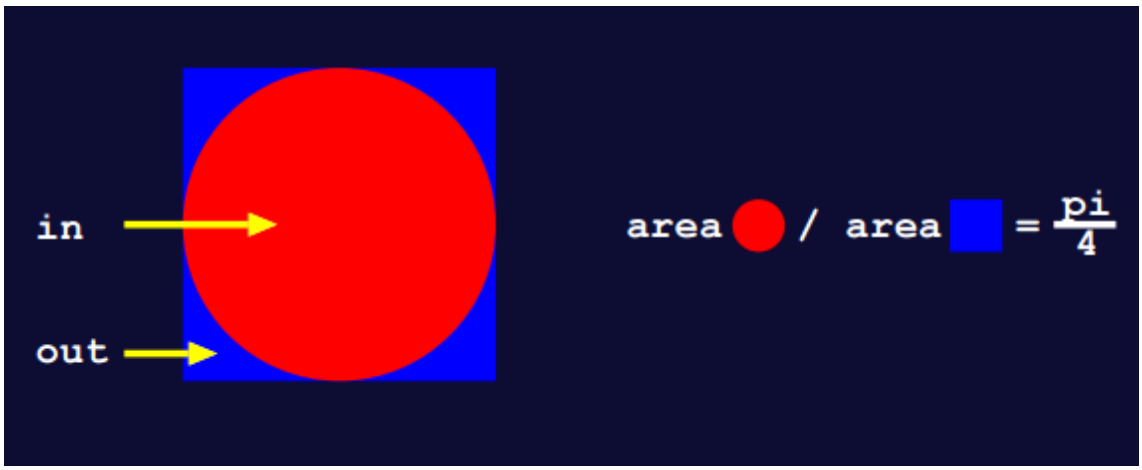

Figura 24. Esquema da fórmula de cálculo (Adaptado de [11]).

Para se conseguir obter o valor de Pi por aproximação este método considera um círculo de raio =1 dentro dum quadrado de lado =2, em seguida são gerados pontos aleatórios dentro do quadrado, os pontos que ficam dentro do círculo são contados e o número de pontos total é guardado.

Com a seguinte fórmula que é a usada pelo método de Monte Carlo conseguimos obter o valor de Pi por aproximação, quantos mais pontos forem mapeados maior será a

aproximação, daí a vantagem de correr o programa num *cluster* para obter o resultado deste cálculo:

$$\pi = 4 * \frac{Pontos\ círculo}{Pontos\ totais}$$

A figura 25 mostra como o método funciona, contendo já uma soma de pontos para demonstração.

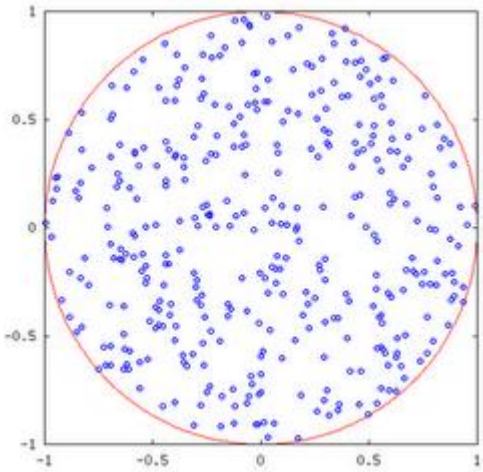

Figura 25. Distribuição aleatória de pontos (adaptado de [12])

Depois de executada a fórmula podemos ver a convergência do valor de Pi pelo seguinte gráfico, ao qual o valor de Pi estabiliza entre 200 e ≈300 pontos para o seu valor 3.1415926 representado pela linha recta vermelha no gráfico da figura 26. [12]

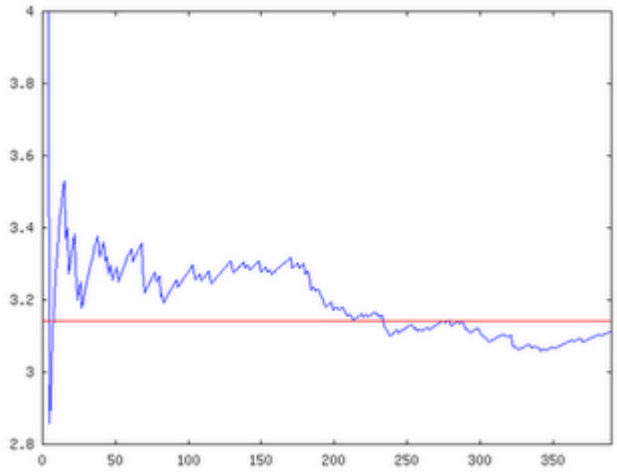

Figura 26. Convergência para o valor de Pi (adaptado de [12])

4.2. Algoritmo e código para o cálculo do valor de Pi:

O algoritmo usado para a implementação do método descrito na secção anterior, foi o seguinte:

```
Início
      Definição das variáveis
      Definição dos nPontos para aproximação
      Divisão do nPontos pelo nEscravos = meusPontos

      Lançamento da tarefa pelos escravos

      Fazer cálculo de somas até meusPontos ser = 0
            Gerar pontos random dentro do quadrado
            Fazer soma parcial dos pontos dentro do círculo
      Fim cálculo

      Recolha das somas parciais dos escravos e enviá-las somadas para
o Mestre

      Mestre divide somasEscravos pelo nPontos obtendo valorPi
      Escreve valor de Pi
Fim
```

O código do programa para este algoritmo é apresentado a seguir e foi desenvolvido em linguagem C com recurso à biblioteca de computação paralela MPI [14]. O programa calcula o valor de Pi por aproximação usando o método de Monte Carlo, como explicado na secção anterior, distribuindo os cálculos a realizar pelo conjunto de nós do *cluster*, paralelizando assim o método.

O código do programa a baixo:

```
/*incluir as bibliotecas base e de funções MPI*/
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
```

1

```
int main(int argc, char *argv[]) {
/*Definição de variáveis*/
  int myPos,nNodes; //pos_actual | num_totalNodes
  long int npts = 1e9; //num de pontos mapeados 1000.000.000
  long int i,mynpts;
  double Pi,sum,mysum;
  double Xmin,Xmax,X;
```

2

```
/*inicio das chamadas de MPI*/
  MPI_Init(&argc,&argv); //iniciar MPI
  MPI_Comm_size(MPI_COMM_WORLD,&nNodes); //dá o total de nós
  MPI_Comm_rank(MPI_COMM_WORLD,&myPos); //dá a pos. do nó.
```

3

```
/*Divisão das tarefas*/
  if (myPos == 0) {
    mynpts = npts - (nNodes-1)*(npts/nNodes);
  } else {
    mynpts = npts/nNodes; /*divisão do npts pelos nós do Cluster
paralelizando assim o cálculo*/
  }

/*Inicialização das variáveis*/
  mysum = 0.0;
  Xmin = 0.0;
  Xmax = 1.0;

  srand(myPos); //lançamento da semente aleatória pelos nós

/*Somatório parcial dos pontos*/
  for (i=0; i<mynpts; i++) {
    X = (double) rand()/RAND_MAX*(Xmax-Xmin) + Xmin;
    mysum += 4.0/(1.0 + X*X);
  }

/*Envio do resultado pela Função MPI_Reduce para root = 0*/
  MPI_Reduce(&mysum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

  /*Apresentação ~PI, recolha das somas parciais pela MPI_Reduce*/
  if (myPos == 0) {
    Pi = sum/npts;
    printf("PI calculated with %ld points = %f \n",npts,Pi);
  }

/*Fim do programa*/
  MPI_Finalize();
}
```

4 5 6 7 8 9 10

Explicação das secções identificadas no código:

1) Inclusão das bibliotecas base e as de MPI;
2) Definição das variáveis globais do programa;
3) Inicio das chamadas de funções de **MPI, MPI_Init** começa por iniciar o ambiente de execução **MPI, MPI_Comm_size** regista o número total de nós do *cluster* em **nNodes** e **MPI_Comm_rank** regista a posição em que nó está no total de nós em **myPos**;
4) Aqui faz se a divisão da tarefa, ou seja, dividem-se os pontos pelo número de nós disponíveis do *cluster*, guardando a divisão em **mynpts**;
5) Declaração de variáveis locais;
6) Lançamento da tarefa aleatoriamente pelos nós do *cluster*;

7) Usando a fórmula do método de Monte Carlo, gera pontos aleatórios dentro do quadrado e soma-os, guardando a soma parcial dos pontos em **mysum**;
8) A função **MPI_Reduce** recolhe no final os resultados dos nós escravos, juntando as somas, devolvendo-as ao nó *root*, ou seja, ao mestre na variável **sum**;
9) No mestre apresenta o valor de Pi já aproximado;
10) Termina o ambiente de execução MPI.

Executando o código no cluster:

1. Compilou-se o código do programa com o nome monte_carlo.c que está na pasta '/dados' partilhada com o seguinte comando:
   # mpicc monte_carlo.c

2. Depois de compilado corri o programa no *cluster* onde passamos o número de nós que queremos usar na flag 'np' o resultado obtido está visível na figura 27:
   #time mpirun -np 1 --hostfile hostfile ./a.out

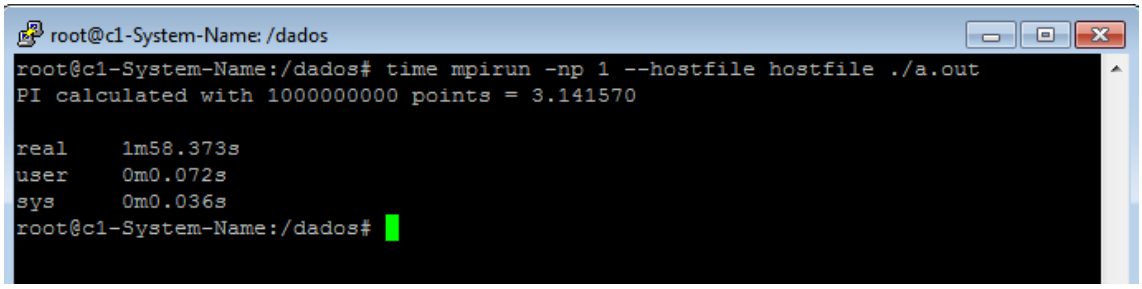

Figura 27. Resultado do cálculo com 1 nó.

3. Subindo o número de nós para 2, o resultado foi uma redução no tempo de cálculo para cerca de metade como se pode observar na figura 28:
   #time mpirun -np 2 --hostfile hostfile ./a.out

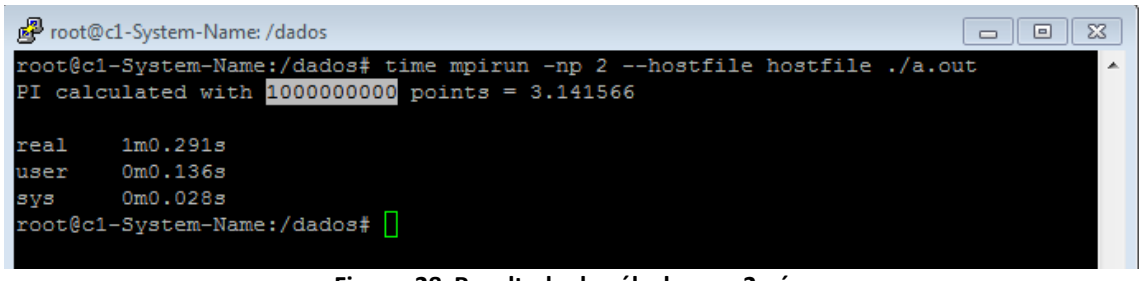

Figura 28. Resultado do cálculo com 2 nós.

4. Com 4 nós:
   #time mpirun -np 4 --hostfile hostfile ./a.out

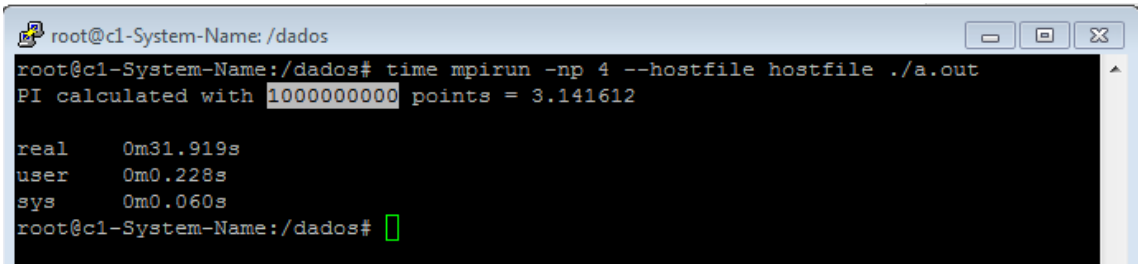

Figura 29. Resultado do cálculo com 4 nós.

5. Com o 6 nós escravos:
   #time mpirun -np 6 --hostfile hostfile ./a.out

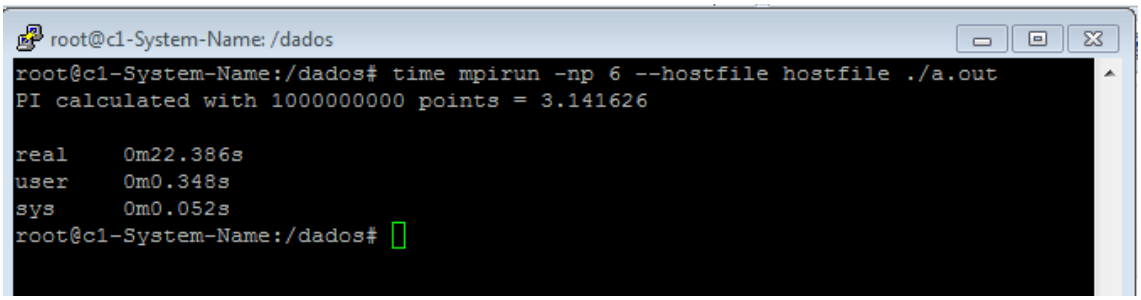

Figura 30. Resultado do cálculo com 6 nós.

4.3. Análise dos Resultados:

À medida que duplicamos o número de *cores*/nós a realizar o cálculo o tempo é reduzido sensivelmente para metade do tempo anterior, podemos concluir que o método de Monte Carlo é ideal para ser executado em processamento paralelo pois escala em rapidez (reduzindo o tempo de cálculo) consoante é duplicado o número de máquinas a realizar o teste, como se pode observar pelo gráfico da figura 31.

Embora se possa constatar pelos resultados apresentados no gráfico da figura 31 que o aumento do número de nós usados no cálculo, diminui o tempo total do mesmo, verificasse que a partir de 4 nós o tempo tende a estabilizar e não se obtém uma melhoria significativa. Uma razão provável para a estabilização dos tempos de execução do programa será o aumento de tráfico de dados pelo barramento (relacionado com a comunicação entre processos) que ocorre quando aumentamos o número de nós usados no processamento.

Seria também expectável, seguindo esta leitura que se tivéssemos um *cluster* com muitos mais nós e fossem todos usados para calcular este programa o tempo de cálculo poderia inclusive a ser maior ao qual era só com um nó pela mesma razão, ou seja, o entupimento do barramento com dados relacionados com o processamento paralelo.

Os resultados foram bastante bons pois como se esperava este método de Monte Carlo precisa de muitos pontos para conseguir devolver o valor de Pi com precisão, correndo num *cluster* obtêm-se os resultados que buscávamos com menor tempo testando assim o *cluster* e cumprindo o objectivo deste projecto.

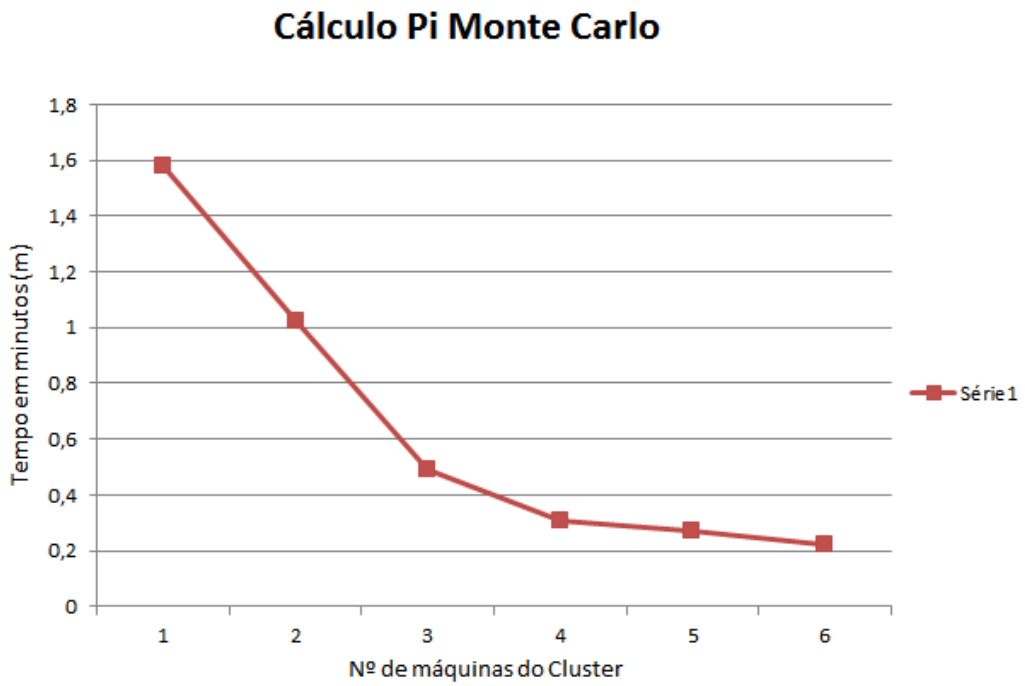

Figura 31. Gráfico de análise dos resultados.

5. CONCLUSÕES

A realização deste projecto envolveu vários desafios desde a criação deste relatório em a forma de tutorial, foi tido em cuidado não o tornar demasiado cheio de informações mas sim, o foco nas partes realmente importantes, mas também não o tornar demasiado superficial o que o tornaria muito incompleto, a dificuldade foi torná-lo o máximo equilibrado.

O estudo sobre o tema foi exaustivo, foi difícil encontrar a informação que fosse compatível e realmente útil para o auxílio na realização deste projecto. O principal recurso usado foi a internet, ou seja tudo o que pude usar foram tutoriais, artigos publicados, apresentações, páginas web com conteúdos pertinentes, fóruns e vídeos.

Quanto à montagem em si do *cluster* já estava previamente montado, unicamente foi solucionado com ajuda do orientador o problema de onde o colocar para a realização do projecto, foi fornecida a sala com os requisitos, com espaço para o *cluster* e com ligação à internet. Alguns computadores já não estavam funcionais mas foram consertados, por mim até à instalação do SO, tornando alguns a avariar e não sendo usados depois.

Na etapa do desenvolvimento do projecto em si, foram surgindo novas dificuldades devido aos baixos conhecimentos em Linux e pouca prática da minha parte com o SO, tornando-se cada passo da configuração uma nova procura/pesquisa. Outra das dificuldades foi a demora no processo de instalação/configuração pois os computadores são já antigos e algo lentos para os padrões actuais tornando assim o trabalho muito mais demorado e chato.

Contudo a maior parte das dificuldades foram superadas com o meu esforço, com a ajuda do fiel orientador e da Professora Filipa Gaudêncio que me auxiliou em alguns comandos Linux e ligação do *cluster* à internet.

Por fim concluo que a realização do projecto foi positiva e muito enriquecedora, pois aprendi imenso em Linux e adquiri alguns conhecimentos na área da programação paralela e distribuída bem como na área do *cluster computing*.

BIBLIOGRAFIA

[1] "Supercomputadores," [Online]. Available: https://pt.wikipedia.org/wiki/Supercomputador. [Acedido em 03 11 2015].

[2] "O que é cluster e como funciona?," [Online]. Available: http://faqinformatica.com/o-que-e-cluster-e-como-funciona/. [Acedido em 03 11 2015].

[3] "Cluster," 2013. [Online]. Available: http://www.infowester.com/cluster.php. [Acedido em 19 11 2014].

[4] "Blog Rede Host," [Online]. Available: http://blog.redehost.com.br/cloud-server-2/cluster-e-seus-conceitos.html. [Acedido em 20 11 2014].

[5] M. Pitanga, "computacao-em-cluster," 30 05 2003. [Online]. Available: http://www.clubedohardware.com.br/artigos/computacao-em-cluster/153. [Acedido em 14 10 2015].

[6] A. B. P. d. Barros, "Computação em Cluster," 2008. [Online]. Available: http://www.ice.edu.br/TNX/encontrocomputacao/artigos-internos/prof_andersown_computacao_em_cluster.pdf. [Acedido em 26 Junho 2015].

[7] "Creating a Raspberry Pi-Based Beowulf Cluster," 22 05 2013. [Online]. Available: http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf. [Acedido em 29 10 2015].

[8] D. Tonidandel, "MANUAL DE MONTAGEM DE UM CLUSTER BEOWULF," 12 2008. [Online]. Available: http://www.em.ufop.br/cecau/monografias/2008/DANNY%20TONIDANTEL.pdf. [Acedido em 13 10 2015].

[9] A. B. M. A. A. A. David Beserra, "Um Cluster Virtual Automatizado," [Online]. Available: http://www.imago.ufpr.br/csbc2012/anais_csbc/eventos/e-science/artigos/E-SCIENCE%20-%20Um%20Cluster%20Virtual%20Automatizado%20para%20Reuso%20de%20Recursos%20Ociosos%20em%20um%20Laboratorio%20Universitario%20de%20Uso%20Geral.pdf. [Acedido em 21 10 2015].

[10] "OpenSSH," [Online]. Available: https://pt.wikipedia.org/wiki/OpenSSH. [Acedido em 06 11 2015].

[11] R. Rocha, "Programação Paralela e Distribuída," 2009. [Online]. Available: https://www.dcc.fc.up.pt/~ricroc/aulas/0910/ppd/apontamentos/mpi.pdf. [Acedido em 04 02 2016].

[12] "Center Of High Performance Computing," [Online]. Available: http://chpc.wustl.edu/mpi-c.html. [Acedido em 02 12 2015].

[13] "Monte Carlo Pi," Tiago Charters de Azevedo, [Online]. Available: http://www.diale.org/montecarlo_pi.html. [Acedido em 10 11 2015].

[14] A. N. L. a. M. S. University, “mpi.h,” 10 01 2003. [Online]. Available: http://web.mit.edu/16.225/software/include/mpi.h. [Acedido em 02 12 2015].

[15] “Cluster Beowulf Utilização,” 14 10 2014. [Online]. Available: http://cluster.ft.unicamp.br/wiki/doku.php. [Acedido em 17 06 2014].

[16] J. L. C. Neto, “Cluster Beowulf,” [Online]. Available: http://www.vivaolinux.com.br/artigo/Cluster-Beowulf?pagina=1. [Acedido em 28 10 2014].

[17] L. Woodman, “Setting up a Beowulf Cluster Using Open MPI on Linux,” 02 12 2009. [Online]. Available: http://techtinkering.com/2009/12/02/setting-up-a-beowulf-cluster-using-open-mpi-on-linux/. [Acedido em 20 11 2014].

[18] H. V. Bacellar, “Cluster: Computação de Alto Desempenho,” [Online]. Available: http://www.ic.unicamp.br/~ducatte/mo401/1s2010/T2/107077-t2.pdf. [Acedido em 12 10 2015].

[19] J. E. d. Silva, “Historia-do-GNU-Linux-1965-assim-tudo-comecou,” 02 01 2006. [Online]. Available: http://www.vivaolinux.com.br/artigo/Historia-do-GNU-Linux-1965-assim-tudo-comecou/. [Acedido em 18 10 2015].

[20] “O que é DHCP e para que serve?,” [Online]. Available: http://www.palpitedigital.com/o-que-e-dhcp-e-para-que-serve/. [Acedido em 16 10 2015].

[21] “A (very) Brief Introduction to MPI: C example,” Washington University in St. Louis, 2012. [Online]. Available: http://chpc.wustl.edu/mpi-c.html. [Acedido em 29 11 2015].

ANEXOS

Programa em C usado para o cálculo de Pi:

```
/*incluir as bibliotecas base e de funções MPI*/
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

int main(int argc, char *argv[]) {
/*Definição de variáveis*/
  int myPos,nNodes; //pos_actual | num_totalNodes
  long int npts = 1e9; //num de pontos mapeados 1000000000
  long int i,mynpts;
  double Pi,sum,mysum;
  double Xmin,Xmax,X;

/*inicio das chamadas de MPI*/
  MPI_Init(&argc,&argv); //iniciar MPI
  MPI_Comm_size(MPI_COMM_WORLD,&nNodes); //dá o total de nós
  MPI_Comm_rank(MPI_COMM_WORLD,&myPos); //dá a pos. do nó.

/*Divisão das tarefas*/
  if (myPos == 0) {
    mynpts = npts - (nNodes-1)*(npts/nNodes);
  } else {
    mynpts = npts/nNodes; /*divisão do npts pelos nós do Cluster
paralelizando assim o cálculo*/
  }

/*Inicialização das variáveis*/
  mysum = 0.0;
  Xmin = 0.0;
  Xmax = 1.0;

  srand(myPos); //lançamento da semente aleatória pelos nós

/*Somatório parcial dos pontos*/
  for (i=0; i<mynpts; i++) {
    X = (double) rand()/RAND_MAX*(Xmax-Xmin) + Xmin;
    mysum += 4.0/(1.0 + X*X);
  }

/*Envio do resultado pela Função MPI_Reduce para root = 0*/
  MPI_Reduce(&mysum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  /*Apresentação ~PI, recolha das somas parciais pela MPI_Reduce*/
  if (myPos == 0) {
    Pi = sum/npts;
    printf("PI calculated with %ld points = %f \n",npts,Pi);
  }
/*Fim do programa*/
  MPI_Finalize();
}
```