UNIVERSIDADE SAGRADO CORAÇÃO

PAULO ROBERTO GIFALLI

ANALISE DE PERFORMANCE DE CLUSTER BASEADO EM RENDERIZAÇÃO DE ANIMAÇÃO

BAURU
2012

PAULO ROBERTO GIFALLI

ANALISE DE PERFORMANCE DE CLUSTER BASEADO EM RENDERIZAÇÃO DE ANIMAÇÃO

Trabalho de conclusão de curso, apresentado como requisito final para obtenção do titulo de Bacharel em Ciência da Computação, sob orientação do Prof. Dr. Kelton Costa.

BAURU
2012

Gifalli, Paulo Roberto

G457d

Analise de Performance de Cluster Baseado em Renderização de Animação/ Paulo Roberto Gifalli -- 2012.

61 f. : il.

Orientador: Prof. Dr. Kelton Costa

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Sagrado Coração - Bauru - SP

1. Cluster. 2. Renderização. 3. Poder computacional 4. Aquitetura Multicor. I. Costa, Kelton. II. Título.

PAULO ROBERTO GIFALLI

DISTRIBUIÇÃO DE CLUSTER PARA ANÁLISE DE PERFORMANCE

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Sociais Aplicadas como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, sob orientação do Professo Dr. Kelton A. Pontara Costa.

Banca examinadora:

Prof. Dr. Kelton A. Pontara Costa
Universidade do Sagrado Coração
Orientador

Prof. Ms. Marcio Cardim
Universidade do Sagrado Coração
Examinador

Prof. Dr. Dariel de Carvalho
Universidade do Sagrado Coração
Examinador

Bauru, 20 de junho de 2012.

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer aos meus pais Pedro Gifalli, Sandra S. Geraldo Gifalli e irmão Fabio G. Gifalli e aos avós, Clarinda Gifalli e Emygdio Gifalli, pela minha formação como cidadão, bem como pelo incentivo nos momentos mais difíceis de minha vida, em acreditarem no meu potencial.

Agradeço à minha esposa, Luciene F. Estefani Gifalli, por tudo que vivemos nesta última década juntos, ao amor e carinho ao longo de todos estes anos, e sobretudo ao seu companheirismo e apoio psicológico, para o desenvolvimento deste trabalho.

Nesta simples letra de música do cantor Alexandre Pires, resumo estes longos anos de agradecimento:

".....É por amor que os
Corações se tornam
Tão indivíveis
Só no amor que a gente
Entende e sente
O quanto é importante
É por amor que eu estou
Contigo e vivo a cada instante
Você me faz sorrir do mundo
E das dificuldades
É o meu porto mais seguro
Em minhas tempestades...."

(Alexandre Pires)

Agradeço à todos companheiros da estação desafio, em todos estes longos anos juntos, de forma a lapidar este material bruto, me proporcionando a cada dia novos desafios e superações tanto dentro quanto fora da fábrica.

"Não tinha como dar errado", esta era frase que sempre me movia a superar os obstáculos, que fui tendo no desenvolvimento deste trabalho academico.

Ao Prof. Dr. Kelton Costa, amigo e orientador do trabalho, que sempre acreditou em meu potencial e que depositou sua confiança em mim para desenvolvimento deste trabalho.

À todos os amigos da universidade, especialmente para Luciano, Leandro Brito, pelos momentos que convivemos durante os quatro anos de curso.

Uma menção especial ao meu grande amigo Luciano ("Insano") pelo apoio dado em que todos os anos de faculdade.

"Saiba meu grande amigo, que torço muito por você, nestas sua nova jornada, você é uma pessoa fantástica, espero lá na frente nos encontrar para falarmos dos grandes desafios superados".

Uma outra pessoa muito especial que vou levar comigo pela simplicidade é o grande companheiro de livros e auxílios, Leandro Brito, “saiba filhão que você é uma pessoa muito fantástica, continue sendo esta pessoa que é, que logo você irá muito longe, como já está indo, decolando em sua carreira”.

Obrigado meu Deus, por tudo que vem me proporcionando nesta vida, ilumine e abençoe a cada um citado neste trabalho, seja aonde eles estiverem!!!!!

RESUMO

Increasingly, current applications require high processing power, such as a game development, application of special effects to a scene in a movie. Based on this need for computational power, is studying some tools for image rendering. In this work, an evaluation of the performance rendering tools for pre-formatted images making use of clusters, consisting of computers that have multicore architectures. In the theoretical study done with the affairs belonging to the subject, we performed a case study using the Blender rendering tool. As operating system, Ubuntu Server x86 version 11, which possesses resources of parallelism, so you can see the division of an animation into a set of sub-tasks, thus making it an independent process, achieving process migration between nodes and making use of the cluster of nucle present in multicore computers.

Palavras-chave: Cluster. Renderização. Poder computacional. Arquitetura multicor.

ABSTRACT

Increasingly, current applications require high processing power, such as the game development, application of special effects to a scene in a movie. Based on this need for computational power, is studying some tools for image rendering. In this work, an evaluation of the performance rendering tools for pre-formatted images making use of clusters, consisting of computers have multicore architectures That. In the Theoretical study done with the affairs belonging to the subject, we Performed a case study using the Blender rendering tool. The operating system, Ubuntu Server x86 version 11, Which possesses resources of parallelism, so Can you see the division of an animation into a set of sub-tasks, Thus making it an independent process, process migration Achieving Between nodes and making use of the Nucleus present in cluster of multicore computers.

Keywords: Cluster. Rendering.Computational power. Multi-colored architecture.

LISTAS DE ILUSTRAÇÕES

LISTA DE SIGLAS

CESDIS (Center of Excellence in Space Data and Information Sciences)

HCA (Host Channel Adapter)

HPC (High Performance Computing)

IBTA (InfiniBand Trade Association)

LSU (Louisiana State University)

MPP´s (Massively Parallel Processors)

MIMD (Multiple Instruction Multiple Data)

MPI (Message Passing Interface)

NOWs (network of workstations)

PVM (Parallel Virual Machine)

QoS (Qualidade de Serviço)

SMP´s (Symmetric Multiprocessing)

SPMD (Single Program Multiple Data)

TCA (Target Channel Adapter)

SUMÁRIO

1 INTRODUÇÃO

Tradicionalmente, quando se menciona a tecnologia de cluster de computadores é inevitável a associação com centros de pesquisa realizando grandes quantidades de computação intensiva; porém, com a redução considerável dos custos dos computadores convencionais e a sua utilização em larga escala presencia-se a inserção desta tecnologia também nos ambientes corporativos. O uso desta tecnologia é, muitas vezes, empregada para solucionar o problema de necessidade cada vez maior dentro das organizações.

Foi desenvolvido um trabalho metodológico de modo a defender e difundir a idéia de ampliação do uso da tecnologia de clusters de computadores, estendendo-a pelos mais variados tipos e campos, onde os mesmos poderiam ser empregados, como por exemplo a renderização de imagem.

Segundo Manika (2000), diversas empresas de pequeno porte e universidades, com poucos recursos financeiros estão recorrendo a sistemas de cluster como alternativa. Isso permite que essas instituições possam aproveitar seus *hardwares* existentes para implementação desse sistema. Com relação às áreas de aplicações, os clusters podem ser usados em diversos ramos de atividades, como: na área de sensoriamento remoto onde necessitam de alto desempenho para processar imagens de satélites; outra área muito utilizada é a da engenharia genética, onde um cluster pode ser de grande utilidade no projeto Genoma; em exploração sísmica para determinação de local de poços petrolíferos, através de processamento rápido para simulação vetorial de uma região; na área de previsão de tempo, para realizar simulações climáticas de maneira rápida e com grande precisão, permitindo que os resultados das previsões sejam observada pelos pesquisadores com antecedência, antes da realização do fenômeno climático; uma outra importante área onde um cluster é utilizado é na segurança de reatores nucleares (monitorações críticas em tempo real), com intuito de analisar as condições dos reatores, através de simulações de operações, assim evitando possíveis acidentes.

Um cluster, ou aglomerado de computadores, é formado por um conjunto de computadores, que utiliza um tipo especial de sistema operacional classificado como sistema distribuído. Muitas vezes é construído a partir de computadores convencionais (personal computers), os quais são ligados em rede e comunicam-se através do sistema, trabalhando como se fossem uma única máquina de grande porte. Há diversos tipos de cluster. Um tipo famoso é o cluster da classe Beowulf, constituído por diversos nós escravos gerenciados por um só computador (PITANGA, 2003).

Como cita Machado (2005) uma das ferramenta muito importante dentro do cluster é o sistema de paralelismo, é uma forma de computação em que vários cálculos são realizados simultaneamente, operando sob o princípio de que grande problemas geralmente podem ser divididos em problemas menores, que então são resolvidos concorrentemente (em paralelo). Existem diferentes formas de computação paralela: em bit, instrução, de dado ou de tarefa. A técnica de paralelismo já é empregada por vários anos, principalmente na computação de alto desempenho, mas recentemente o interesse no tema cresceu devido às limitações físicas que previnem o aumento de frequência de processamento. Com o aumento da preocupação do consumo de energia dos computadores, a computação paralela se tornou o paradigma dominante nas arquiteturas de computadores sob forma de processadores multinúcleo.

Computadores paralelos podem ser classificados de acordo com o nível em que o hardware suporta paralelismo. Computadores com multinúcleos ou multiprocesadores possuem múltiplos elementos de processamento em somente uma máquina, enquanto clusters, MPP e grades usam múltiplos computadores para trabalhar em uma única tarefa. Arquiteturas paralelas especializadas às vezes são usadas junto com processadores tradicionais, para acelerar tarefas específicas.

Programas de computador paralelos são mais difíceis de programar que sequenciais, pois a concorrência introduz diversas novas classes de defeitos potenciais, como a condição de corrida. A comunicação e a sincronização entre diferentes subtarefas é tipicamente uma das maiores barreiras para atingir grande desempenho em programas paralelos. O aumento da velocidade por resultado de paralelismo é dado pela lei de Amdahl.

O programa blender é multiplataforma, estando portanto disponível para diversos sistemas operacionais. O Blender implementa ferramentas similares às de outros programas proprietários, que incluem avançadas ferramentas de simulação, tais como: dinâmica de corpo rígido, dinâmica de corpo macio e dinâmica de fluidos, ferramentas de modelagem baseadas em modificadores, ferramentas de animação de personagens, um sistema de composição baseado em “nós” de texturas, cenas e imagens, e um editor de imagem e vídeo, com suporte a pós-produção. Inclui suporte a Python como linguagem de script, que pode ser usada tanto no Blender, quanto em seu motor de jogo. Atualmente, suporta 25 idiomas, incluindo o português brasileiro (JOOMLA, 2011).

Quando se fala em renderização de imagem fala-se em alto poder computacional, com base nesta informação, estes extremos propõe testes de capacidade computacional, por meio da renderização de imagens, hoje um campo bastante procurado, como por exemplo as empre-

sas automobilísticas, nos desenvolvimento dos seus mais variados modelos de carros, como também na área de animações de filmes, etc (ALECRIM, 2011).

1.1 OBJETIVO GERAL

Analisar a performance de processamento paralelo de um cluster na renderização de uma animação com a ferramenta blender.

1.2 OBJETIVOS ESPECÍFICOS

- Utilizar a ferramenta b*lender*
- Quantificar por meio de medições de tempo de processamento, quantidade de frames, utilização de recursos como processador, memória ram, quantidade de nós e núcleos, gerando assim, sua capacidade de processamento.

1.3 JUSTIFICATIVA

A construção de um cluster do tipo alto desempenho pode oferecer várias vantagens, em termos de utilização. Utilizam em seu sistema, computadores pessoais os quais em conjunto pode obter uma alta escalabiliade, onde apresenta a possibilidade de serem adicionados novos componentes (inclusão de novos computadores no *cluster*) à medida que aumenta a carga de trabalho para o mesmo; não dependem de licença, porque utilizam *softwares* livres, não havendo a preocupação de pagar a licença de uso e ficar dependendo da manutenção dos fabricantes e principalmente alto desempenho, pois tem a capacidade de solucionar problemas de grande complexidade através de processamento paralelo, o que faz com que se diminua o intervalo de tempo de processamento na resolução do problema.

Segundo Pitanga (2003), as tarefas que exigem grande poder computacional, até pouco tempo atrás, só podiam ser executadas por supercomputadores. No entanto, nos últimos anos, a *TI* (Tecnologia da Informação) evoluiu rapidamente, fazendo com que as organizações usufruíssem dessa evolução. Com o avanço tecnológico dos computadores pessoais e das estruturas de redes locais, cada vez mais velozes, surge a ideia de utilizar o poder individual destes computadores, conectando-os através de um meio físico de comunicação e dotados de softwares específicos para executarem aplicações paralelas e distribuídas, dando ao usuário transparência quanto ao seu funcionamento (BUYYA, 1999). Assim como Buyya (1999) cita, quando se utiliza dois ou mais computadores em conjunto para resolver um determinado problema, temos o que chamamos de *cluster*, que, do inglês, significa agrupamento, aglomerado ou concentração.

Assim como a tecnologia relacionada aos *clusters*, a indústria de processadores tem

evoluído em velocidade rápida nos últimos anos. A proliferação dos computadores e das tarefas que lhes são incumbidos, continua a pressionar a necessidade de processadores mais poderosos (WOODS, 2003). O Multicore consiste em colocar dois ou mais núcleos (*cores*) no interior de um único chip. O objetivo é possibilitar ao sistema executar várias operações em simultâneo e assim alcançar melhor desempenho para responder às necessidades dos dias de hoje. Assim, os processadores Multicore representam uma grande revolução na tecnologia computacional, sendo capazes de prover maior capacidade de processamento com um custo/benefício melhor do que processadores *singlecore* (PITANGA, 2003).

1.4 ESTRUTURA DO TRABALHO

Além do capítulo um, o trabalho está composto de mais cinco capítulos, conforme a descrição sumaria abaixo:

No capítulo 2 são observados, conceitos de *cluster*, é definido como é composto um Cluster como, tecnologia multicore, identificação de paralelismo, alta disponibilidade; no capítulo 3, Ferramenta de *Renderização* de Imagem, é descrito o conceito da renderização de imagem e sua origem; já no capítulo 4, Origem do *Frame* é apresentada a forma que constitui um *frame*, e seus vários formatos; para o capítulo 5, Metodologia, descreve-se os resultados obtidos, com as técnicas aplicadas e avaliação dos resultados colhidos; no capítulo 6, Considerações Finais, conclui - se a respeito do projeto e trabalhos futuros; identificando as Referências Bibliográficas consultadas e utilizadas no decorrer do desenvolvimento esta pesquisa.

2 CLUSTER

A resolução de diversos problemas científicos e comerciais complexos exige grande esforço computacional. Computadores de grande porte têm sido utilizados com frequência nesse tipo de aplicação (YOKOKURA, 2007).

Para alcançar grande poder de processamento, uma abordagem que fornece alto desempenho computacional e processamento paralelo. Dentre as abordagens utilizadas para processamento de alto desempenho, podem ser destacadas a utilização de supercomputadores *clusters* de computadores (PITANGA, 2003).

A determinação de propriedades físicas dos materiais com uso das ferramentas da análise de imagens (GONZALES, 1993).

Segundo Yokokura (2007), os supercomputadores são extremamente caros, bem como os chamados *MPP´s* (*Massively Parallel Processos*), multicomputadores construídos com milhares de processadores comerciais conectados por uma rede proprietária de alta velocidade, uma alternativa com custo muito menor e que resulta em bons resultados, ou seja, garante um bom poder de processamento são os *clusters* de computadores, com um custo muito baixo e alto desempenho, os *clusters* têm se tornado a cada dia uma alternativa aos *MPP´s* (Massively Parallel Processors) supercomputadores.

Segundo Perry (2005), os clusters são representantes da Arquitetura *MIMD* (Multiple Instruction Multiple Data) são arquiteturas caracterizadas pela execução simultânea de múltiplos fluxos de instruções. Essa capacidade deve-se ao fato de que são construídas a partir de vários processadores operando de forma cooperativa ou concorrente, na execução de um ou vários aplicativos. Essa definição deixa margem para que várias topologias de máquinas paralelas e de redes de computadores sejam enquadradas como *MIMD*. A diferenciação entre as diversas topologias *MIMD* é feita pelo tipo de organização da memória principal, memória cache e rede de interconexão, com memória distribuída, onde também incluímos as *MPP´s* (Massively Parallel Processors), enquanto nas Arquiteturas *MIMD* com memória compartilhada temos as *SMP´s* (Symmetric Multiprocessing), todos representantes de arquiteturas paralelas.

Segundo dados da Conferência Internacional de Supercomputação realizada em Junho de 2010, dentre os 500 computadores mais poderosos do mundo, 304 (60.8 % do total) são *clusters* de computadores. O *cluster* mais potente do mundo ocupa a 1ª posição na lista dos 500 computadores mais poderosos do mundo. Trata-se do Roadrunner Cluster, localizado no *USA*.

São 13.140 processadores, em 6.562 nós dual Power com 3.200 MHz (1.02 PetaFlops), atingindo um pico de processamento de 1.02 quatrilhões por segundo, com seus 80 TB de memórias. O cluster possui sistema operacional Linux e rede de interconexão *Myrinet*, ocupando 288 *hacks* e ocupa uma área de cerca de 557,41 $m^{2,}$ com um peso de 226,796 Kg. (DIGNOW, 2011).

O cluster foi uma iniciativa do governo americano com a *IBM*, com finalidades de utilizar na segurança nacional dos *USA*, testar material nuclear e sistemas de armas nucleares, previsão de tempo e estudo do universo (ACIDIT, 2011).

Segundo o líder da divisão de computação de alta performance do laboratório de Los Alamos, John Morrison, a curto prazo de tempo, é montar um mapeamento do *HIV* e tenta achar a cura.

Segundo Yokokura (2007), o cluster de computadores são uma alternativa aos supercomputadores, para prover alta performance e alta disponibilidade, e é particularmente atrativo para aplicações de servidores. No final de 1993, Donald Becker e Thomas Sterling, ambos cientistas da *CESDIS* (Center of Excellence in Space Data and Information Sciences) da *NASA*, iniciaram um esboço de um sistema de processamento distribuído construído a partir de hardware convencional como uma medida de combate aos custos dos supercomputadores, onde no início de 1994, nasce o primeiro *cluster* e o projeto Beowulf. O protótipo inicial era um cluster de 16 processadores *DX4* ligados por dois canais *Ethernet* acoplados. A máquina foi um sucesso instantâneo, e esta ideia rapidamente se espalhou pelos meios acadêmicos, pela *NASA* e outras comunidades de pesquisa (DIGNOW, 2011). A utilização de clusters tem sido uma alternativa cada vez mais viável para suprir a demanda do crescente número de aplicações que exigem alto desempenho computacional. Têm sido largamente utilizados no meio acadêmico e comercial para a execução de aplicações ditas de “execução críticas", geralmente, relacionadas a sistemas de tempo real ou que manipulam uma grande quantidade de dados (PITANGA, 2004).

Portanto é possível definir que *cluster* é formado com grupos de computadores interconectados trabalhando de forma conjunta como um único computador. Os computadores do *cluster* se comportam como sistemas que podem, opcionalmente, funcionar de forma independente do *cluster*. Cada computador do *cluster* é tipicamente referenciado como um nó (STALLINGS, 2000). Nessa arquitetura, normalmente, existe um nó mestre que gerencia ou divide as tarefas entre os demais nós, chamados de escravos (PITANGA, 2004).

Figura1 - Cluster.
Fonte: Vieira (2012).

Na Figura 1, mostra um cluster de grande porte, onde dentro de uma sala climatizada, tem-se vários *racks* onde faz toda a massa de processamento, neste caso em específico é utilizado para provedores de internet que precisam estar disponíveis 7 dias por semana 24 horas por dia e 365 dias por ano. Também são usados na ciência, finanças e engenharia. Atuam em diversos projetos de manipulação de proteínas, processos da física, redes neurais, analise do genoma humano.

Independente do Sistema Operacional usado, é preciso usar um software que permita a montagem do *cluster* em si. Esse *software* vai ser responsável, entre outras coisas, pela distribuição do processamento. Esse é um ponto crucial na montagem de um *cluster*. É preciso que o *software* trabalhe de forma que erros e defeitos sejam detectados, oferecendo meios de providenciar reparos, mas sem interromper as atividades do *cluster*. Obviamente, esse tipo de necessidade pode ser controlado através de um equipamento específico, ou seja, não depende apenas do software (PITANGA, 2004).

Pitanga (2003) afirma que para um sistema ser considerado um *cluster*, é necessário que tenha pelo menos dois computadores. Evidentemente, quanto maior o número de computadores no *cluster*, maiores os custos de implementação e manutenção. Isso não se deve apenas ao preço dos computadores, mas também aos equipamentos para comunicação e manutenção do sistema (*switches*, cabos, *hubs*, *nobreaks*, etc). Mas ainda assim, os custos costumam ser menores do que a aquisição/manutenção de supercomputadores e algumas vezes o processamento é até mais eficiente.

2.1 COMPONENTES DE UM CLUSTER

Segundo Yokokura (2007) os principais componentes de um *cluster* de computadores são:

• Nós – são os computadores que fazem parte do *cluster*. Estes podem ser computadores de vários tipos, desde computadores pessoais, com somente uma unidade de processamento, até os computadores multiprocessados (YOKOKURA, 2007).

• Sistema operacional – O sistema operacional mais utilizado em se tratando de *cluster* é o *Linux*, o qual possui código fonte aberto, permitindo assim realizar alterações no código fonte como acontece no caso do [1]openMosix (BUYYA, 1999).

• Rede Local – este componente do *cluster* influi diretamente no seu desempenho. A placa de rede, o *switch* e o cabeamento são os que possuem a função de transmitir e receber os pacotes entre os nós que compõe o *cluster* (YOKOKURA, 2007).

• Protocolos – conjunto de regras e procedimentos para se efetuar uma comunicação de dados rápida e confiável entre os nós do cluster. O protocolo mais utilizado para implementação de um *cluster* é o TCP/IP, apesar de não ser otimizado para tarefas deste tipo, mas oferece um menor custo de implantação (YOKOKURA, 2007).

• Sistema de arquivos – um dos sistemas de arquivos mais conhecidos e utilizados é o [2]*NFS* (Network File System), mas existem diversas bibliotecas de interfaces a sistemas de arquivos paralelos para redes de estações. Um exemplo típico é o [3]DFSA (Direct File System Access), utilizado pelo sistema openMosix. (BUYYA, 1999).

• Ferramentas de comunicação – é necessário que um cluster possua ambientes de programação que ofereçam meios para desenvolver programas paralelos, como é o caso do *MPI* (Message Passing Interface) (YOKOKURA, 2007).

2.2 TECNOLOGIA MULTICORE

Antes de tudo, é preciso compreender quais são as principais partes de um processador.

Até pouco tempo, os processadores da Intel eram formados por dois componentes básicos (INTEL CORPORATIVO, 2011).

Segundo Intem Corporativa (2011), o núcleo (em inglês, *Core*): lugar onde as instruções são executadas. É no núcleo que ocorre as execuções de instruções, cálculos, a no

[1]OpenMosix é uma extensão ao núcleo Linux para clustering em single system image, que possibilita a conversão de uma rede clássica de computadores desktop num super-computador para aplicações Linux.
[2] O NTFS (New Technology File System) é o sistema de arquivos padrão para o Windows NT e seus derivados.
[3] DFSA é um biblioteca padrão do OpenMosix de interface de sistema de arquivos.

cache, é uma memória de acesso rápido localizada dentro do processador, ao invés do processador ir fazer a busca de dados na memória principal o tempo todo, ele possui uma memória interna mais próxima a ele, onde os dados são armazenados temporariamente, no caso do processador sempre procura por um dado dentro da sua memória [4]*cache* antes de ir buscar o dado na memória principal.

Por esta razão, o *cache* também tem influência no desempenho do processador.

Uma analogia bastante interessante feita em relação ao significado de se adicionar um novo núcleo de processamento a um processador é descrita a seguir: “adicionar um novo núcleo assemelha-se a abrir uma nova pista em uma estrada para aliviar o trânsito: os carros não precisam dirigir mais rápido para chegarem mais cedo ao seu destino, eles apenas não são atrasados tanto pelo gargalo de poucas pistas e congestionamentos (PATTERSON, 2009).

Os processadores de múltiplos núcleos como mostra na Figura 2, podem trabalhar em um ambiente multitarefa (PATTERSON, 2009).

Nos sistemas que possuem somente um núcleo, as funções de multitarefa podem ultrapassar a capacidade da *CPU* (Central Processing Unit – Unidade Central de processamento), o que implica em queda no desempenho enquanto as operações aguardam para serem processadas. Em sistemas de múltiplos núcleos, como cada núcleo tem seu próprio *cache,* o sistema operacional dispõe de recursos suficientes para lidar com o processamento intensivo de tarefas executadas em paralelo. Assim, melhora-se a eficiência do sistema e o desempenho dos aplicativos em computadores que executam vários aplicativos em simultâneo (PATTERSON, 2009).

[4] Cache - É um dispositivo de acesso rápido, interno a um sistema, que serve de intermediário entre um operador de um processo e o dispositivo de armazenamento ao qual esse operador acede

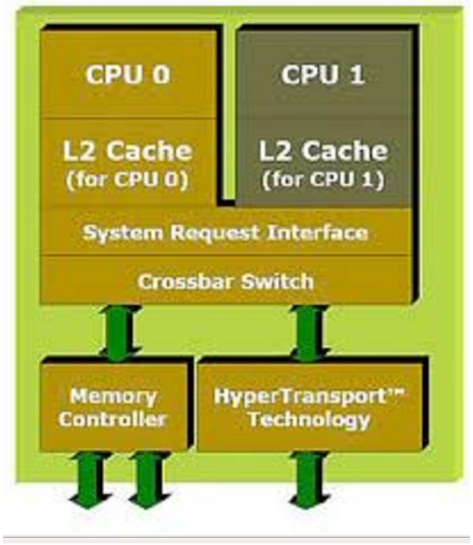

Figura 2 – Processador Multicore.
Fonte: O QUE É... (c2012).

2.3 IDENTIFICAÇÃO DE PARALELISMO

Como cita Machado (2005), os computadores multicore estão cada vez mais disponíveis no mercado. Para se obter vantagem de múltiplos processadores, os programadores e/ou compiladores precisam identificar operações que possam ser executadas em paralelo.

Segundo Buyya (2004), um método para identificar o paralelismo é desenhar o grafo de dependência de dados. A Figura 3 representa esses tipos de grafos, onde os nós desses grafos representam tarefas, e as setas de um nó para outro significa que uma tarefa precisa ser completada antes que a outra tenha início.

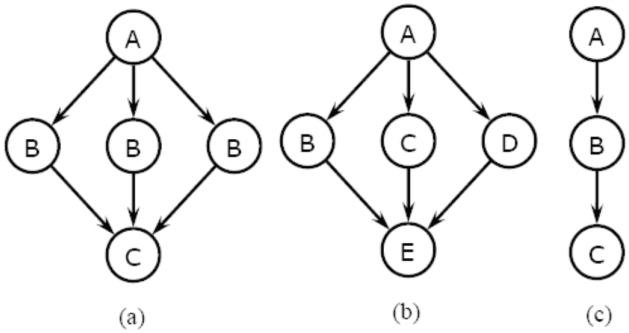

Figura3 - Grafos de dependência de dados.
Fonte: Machado (2005).

Como explica Machado (2005), a Figura 3 tem-se o chamado paralelismo de dados, em que tarefas independentes aplicam a mesma operação a diferentes elementos de dados.

2.4 GRID COMPUTING

Segundo Foster (2003) define [5]*Grid* como uma infraestrutura emergente que irá mudar fundamentalmente a forma como pensamos e utilizamos os computadores. A palavra *Grid* é utilizada como uma analogia as grades energia, que provem acesso pervasivo a eletricidade, da mesma forma que o computador e outros pequenos avanços, proporcionaram um dramático impacto nas capacidades humanas e na sociedade.

Foster (2003) afirmando que muitos acreditam que, por permitir que todos os componentes de nossa infraestrutura de tecnologia da informação – capacidades computacionais, bancos de dados, sensores e pessoas – sejam compartilhados de uma forma flexível através de ferramentas colaborativas, *Grid* terá um efeito de transformação similar, permitindo o surgimento de novas classes de aplicação.

O impacto previsto por Foster (2003), com o surgimento dos *Grids* não se concretizou comercialmente. Por outro lado, a tecnologia desenvolvida para os *Grids* evoluiu e vem sendo utilizada nos meios acadêmicos (COULOURIS, 2008).

2.5 ALTA DISPONIBILIDADE

Segundo Coulouris (2008), o *cluster* de alta disponibilidade é normalmente construído com a intenção de fornecer um ambiente seguro contra falhas utilizando-se da redundância de componentes (sejam de [6]*hardware*, [7]*software*, serviços de rede ou de interconectividade ou interoperabilidade). Em outras palavras, fornecer um ambiente computacional onde a falha de um ou mais componentes não irá afetar significativamente a disponibilidade do ambiente de computação ou aplicações que estejam sendo usadas.

2.6 ALTO PODER COMPUTACIONAL

Como cita Dollimore (2008), o *cluster* tem um alto desempenho de computação é pro-

[5] Grid Computing é um modelo computacional capaz de alcançar uma alta taxa de processamento dividindo as tarefas entre diversas máquinas, podendo ser em rede local ou rede de longa distância, que formam uma máquina virtual. Esses processos podem ser executados no momento em que as máquinas não estão sendo utilizadas pelo usuário, assim evitando o desperdício de processamento da máquina utilizada.

[6] Hardware é a parte física do computador.

[7] Software - A definição mais simples para software é um segmento de comandos executados, manipulados, redirecionados, modificados ou seguidos gerando a alteração de uma informação (dado) ou evento. Todo procedimento mostrado pela execução do conjunto de instruções em computadores, também é denominado software.

jetado para fornecer maior poder de computação para a solução de um problema e tradicionalmente está relacionado com aplicações científicas, de simulação ou de manipulação de imagens.

O usuário interage com um nó específico para iniciar ou escalonar uma atividade que deverá ser executada. A aplicação, juntamente com as funções internas do cluster, irá determinar como a atividade será dividida e enviada para cada elemento que compõe o ambiente computacional, buscando extrair uma maior vantagem dos recursos disponíveis (DOLLIMORE, 2008).

2.7 BALANCEAMENTO DE CARGA

O *cluster* de balanceamento de carga é usado para fornecer uma interface simplificada para um conjunto de recursos que podem aumentar ou diminuir com o passar do tempo e conforme a necessidade por processamento. Neste tipo de *cluster*, estão implícitos os conceitos da alta disponibilidade (com a redundância de componentes) e de alto desempenho de computação (com a distribuição das tarefas completas pelos vários componentes replicados ao nó central, montando uma espécie de quebra cabeça (CASAVANT, 1988; SINGHAL, 1992).

2.8 ALGORITMOS PARALELOS

Segundo Foster (2005), a metodologia de desenvolvimento de algoritmos paralelos como mostrada na Figura 4 o modelo tarefa/canal descrito por Ian, esse modelo facilita o desenvolvimento de programas paralelos eficientes.

Considera-se a seguir questão de saber que abstrações são apropriadas e úteis num modelo de programação paralela. Claramente, são necessários mecanismos que favoreçam a discussão explícita da concorrência e da localidade e que facilitem o desenvolvimento de programas escaláveis e modulares. São também necessárias abstrações simples de trabalhar que se ajustem ao modelo arquitetônico do multi computador (YOKOKURA, 2007).

A Figura 4 representa o modelo tarefa/canal proposto por Machado (2005), através de um conjunto de tarefas (círculos) que interagem umas com as outras enviando mensagens através de canais (setas). Assim, cada tarefa é um programa, sua memória local e um conjunto de portas de E/S (entrada e saída). Um canal é uma fila de mensagens que conecta a porta de saída de uma tarefa com a porta de entrada de outra tarefa. Nesse modelo, a transmissão de uma mensagem e a recepção de outra pode ser feita concorrentemente.

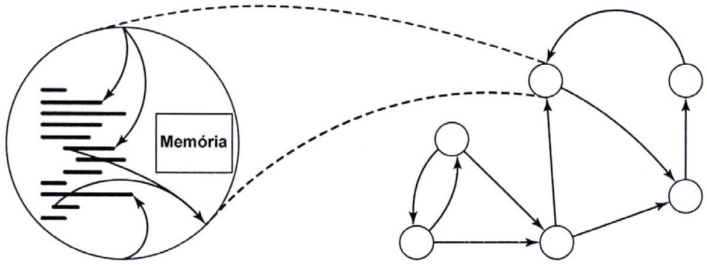

Figura 4 - Modelo tarefa/canal.
Fonte: Machado (2005).

Um modelo de programação simples. A Figura 4 mostra um estado instantâneo de uma computação e uma imagem detalhada de uma tarefa simples. A computação compreende um conjunto de tarefas (representadas por círculos) ligados por canais (setas). Uma tarefa esconde um programa e memória local e define um conjunto de portas que definem o seu interface com o meio ambiente. Um canal é uma fila de mensagens na qual um emissor pode colocar mensagens e de onde um receptor pode retirar mensagens, "bloqueando" se não existirem mensagens disponíveis (MACHADO, 2005).

Uma computação paralela compreende uma ou mais tarefas. As tarefas executam concorrentemente. O número de tarefas pode variar durante a execução do programa (BUYYA, 1999).

Uma tarefa esconde um programa sequencial e uma memória local. (É, com efeito, uma máquina virtual.). Adicionalmente um conjunto de portas de entrada e portas de saída definem o seu interface com o ambiente (MACHADO, 2004).

Uma tarefa pode efetuar quatro ações básicas, para além de ler e escrever na sua memória local enviar mensagens pelas suas portas de saída, receber mensagens, pelas suas portas de entrada, criar novas tarefas e terminar (BUYYA, 1999).

A operação de envio é [8]assíncrona: completa-se imediatamente. A operação de recepção é [9]síncrona: leva à execução da tarefa ficar bloqueada, até que esteja disponível uma mensagem (MACHADO, 2005).

Pares de portas de entrada e porta de saída, canais, podem ser ligados por filas de mensagens chamadas canais. Os canais podem ser criados e eliminados e as referências aos canais

[8] Assíncrona - género mais comum de comunicação, seja por cabo seja por modem. Cada carácter vem entre bits de início e de fim, e a temporização entre os caracteres pode ser desigual, onde o seu oposto é a transmissão síncrona usada na comunicação com alguns mainframes (grandes equipamentos informáticos) ou computadores.»

[9] Síncrono, nesse caso, se referem ao comportamento do emissor das mensagens. Se o emissor aguarda uma resposta ou confirmação antes de transmitir novas informações, é um protolo síncrono. O HTTP é síncrono, o SNMP é assíncrono. Você consegue implementar aplicações assíncronas usando HTTP com Javascript e XML, o famoso AJAX (Asynchronous Javascript with XML).

(portas) podem ser incluídas nas mensagens, por isso a conectividade pode variar dinamicamente (BUYYA, 1999).

As tarefas podem ser atribuídas a processadores físicos de várias maneiras; a correspondência utilizada não afeta a semântica de um programa. Em particular, múltiplas tarefas podem ser alojadas a um mesmo processador. Podemos imaginar uma única tarefa a ser alojada a múltiplos processadores, mas essa possibilidade não é considerada aqui (MACHADO, 2004).

A noção de tarefa fornece um mecanismo para a falar acerca da localidade: os dados contidos na memória local, estão perto; os outros dados são remotos. A abstração canal fornece um mecanismo para indicar que a computação numa tarefa necessita dos dados de uma outra tarefa para poder prosseguir. (Isto é designado por dependência de dados). O seguinte exemplo, simples, ilustra algumas destas características (BUYYA, 1999).

O método de Foster (2003) é dividido em quatro etapas: Partição dos dados e do processamento com a criação das tarefas primitivas, estabelecimento do padrão de comunicação ente as tarefas, aglomeração de tarefas primitivas e mapeamento das tarefas nos processadores disponíveis.

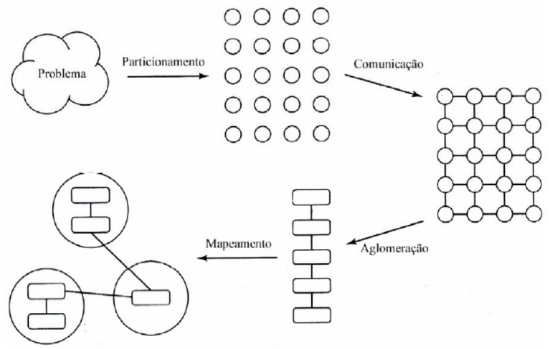

Figura 5 – Metodologia de Foster para projetos de algoritmo paralelos.
Fonte: Machado (2005).

Partição é o processo responsável pela divisão da computação e os dados em partes buscando descobrir o máximo de paralelismo possível. Já na comunicação é onde ocorre a identificação do padrão de comunicação entre as tarefas. Nesses dois processos iniciais o objetivo é identificar o máximo de paralelismo possível sem se preocupar com a plataforma em que o programa vai executar. Na aglomeração ocorre a combinação de tarefas primitivas para mapeá-las em processadores físicos de forma a reduzir a sobrecarga paralela, enquanto no mapeamento vai acontecer a designação das tarefas para os processadores (MACHADO, 2005).

2.9 ALGORITMOS DE COMPARTILHAMENTO DE RECURSOS

Para conseguir um melhor desempenho, este modelo de cluster é sempre verificado através de algoritmos de Balanceamento Dinâmico de Carga e algoritmo de Anunciação de Memória. O objetivo desses algoritmos é responder dinamicamente às variações da utilização dos recursos pelos diversos nós, garantindo sempre a escalabilidade (BUYYA, 1998).

O algoritmo de balanceamento de carga é utilizado quando um programa que está sendo executado em um nó específico está por alguma razão muito sobrecarregado, esse algoritmo irá agir com a finalidade de fazer uma migração desta atividade ou parte dela para um nó com mais recursos disponíveis, o sistema, consequentemente, ganhará em desempenho (PITANGA, 2003).

O novo algoritmo implementado no OpenMosix um *software* capaz de fazer a distribuição das tarefas, é de certa forma, bastante interessante pelo fato de tentar conciliar as diferenças baseado em princípios da economia e análise de competitividade. Para o algoritmo chegar a alguma conclusão, os recursos de comunicação são medidos em termos de largura de banda, memória em termos de espaço e CPU em termos de ciclos. A ideia principal é fazer a conversão do uso total desses diversos recursos heterogêneos. Assim, os processos são então transferidos para o local que possuir um menor custo. Esse algoritmo é executado de forma completamente independente em todos os nós do *cluster* (FERRETO, 2002).

Existe, também, um algoritmo que é usado com a finalidade de se evitar um total esgotamento da memória. Este algoritmo de anunciação de memória entra em ação quando um determinado nó do cluster começa a fazer uma paginação. Nesse caso específico, este algoritmo tem a preferência sobre os demais, mesmo que essa migração do processo possa causar um desbalanceamento da carga (PITANGA, 2003).

2.10 MIGRAÇÃO DE DADOS

A migração de tarefas entre os nós, processos de modo preemptivo, onde os processos são divididos em duas partes, como mostrado na Figura 5: a que é transferida para ser executada em outro nó, denominada de contexto do usuário (remote). A outra parte possui todas as descrições dos recursos dos quais o processo está utilizando ou está alocado para ele, bem como a pilha do sistema que controla a execução do programa. Essa parte é conhecida como contexto do sistema (*deputy*). É bom ressaltar que o processo remote pode ser transferido tan-

tas vezes que se fizer necessário, para qualquer nó do cluster, mas o *deputy* nunca pode ser removido (FERRETO, 2002).

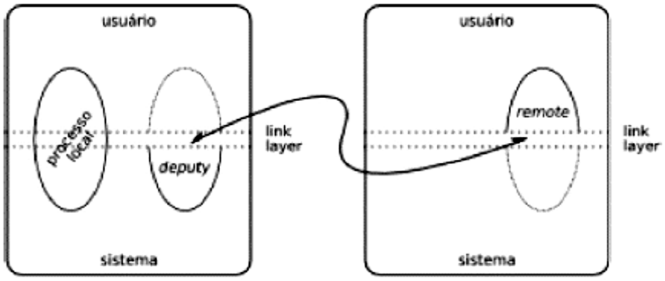

Figura 6 – Fluxo migração de processos.
Fonte: Pitanga (2003).

Ainda sobre a Figura 6, é possível visualizar que existem dois contextos de execução para um processo: o modo usuário e o modo sistema. Como a interface dos dois modos é muito bem definida e conhecida, é possível, de um modo simples, interceptar toda e qualquer interação que existe entre esses dois modos e encaminhá-la para outros nós da rede. Este processo é realizado pela camada de adaptação, também conhecida como *link layer*, a qual contém um canal especial de comunicação para que exista essa interação dos dois contextos do sistema operacional (PITANGA, 2003).

Segundo Perry (2005), o processo de migração de processos pode ser feito de duas formas: automático ou manual. Quando é da forma manual, o administrador faz a indicação de qual processo deverá migrar para determinado nó, podendo ainda indicar a capacidade máxima que um determinado nó deve tolerar antes de iniciar o processo de migração para outro nó do conjunto. Porém, existem muitos processos que podem migrar para outros nós, porém, isso não significa que os mesmos terão benefícios com esta migração.

Quando se tem um conjunto de computadores que devem aparecer como um único recurso (ou seja, como um único sistema em cluster), é preciso usar uma camada de [10]*middleware* denominada *SSI*, que deve estar localizada entre o sistema operacional e as aplicações de usuário (PITANGA, 2004).

Na camada que compõe o *SSI* existem basicamente duas subcamadas como pode ser visto na Figura 7:

A. Infraestrutura de imagem única;

B. Infraestrutura de disponibilidade.

[10] Middleware ou mediador, no campo da computação distribuída, é um programa de computador que faz a mediação entre software e demais aplicações.

C. primeira subcamada deve estar presente em todos os sistemas operacionais para criar a abstração de unificação dos recursos de todas as máquinas em uma única máquina.

A segunda subcamada tem por objetivo oferecer serviços ao cluster como recuperação e tolerância a falhas para todos os nós do cluster (PITANGA, 2003).

Figura 7- Camada de *SSI* em um nó cluster.
Fonte: Pitanga (2003).

Algumas vantagens do uso de *SSI* estão listadas abaixo, como cita (PITANGA, 2003).

- Transparência na execução de aplicações;
- Não necessidade de localização física dos nós;
- Robustez de configuração devido à menor necessidade de intervenção de operadores;
- Permite modelo de processo com espaço globalizado para balanceamento de carga.

2.11 ASPECTOS DA INTERLIGAÇÃO EM SOFTWARE

A interligação de *clusters* recai fortemente no âmbito do *software* usado para permitir que o sistema funcione realmente como um único sistema interligando vários computadores distintos. Quando tem-se um sistema de memória distribuída, no qual cada processador possui sua própria memória local, é necessário definir um modo de interação entre os processos em andamento. Para sistemas em *cluster*, o sistema de troca de mensagens é largamente usado para esse fim, onde as mensagens requerem os dados a serem copiados que serão executados em *threads* locais (BARROSO, 2003).

Para que o ambiente de troca de mensagens seja empregado em *clusters*, é preciso levar em conta alguns aspectos, como:

- Fraco acoplamento entre computadores;
- Possível diferença na largura de banda de interconexão de nós;
- Atrasos na comunicação de nós;
- Diferenças entre hardwares, sistemas operacionais e linguagens dos nós

interligadas.

Os ambientes de programação mais importantes e independentes das características dos computadores utilizados são o [11]*PVM* (Parallel Virtual Machine, Máquina Virtual Paralela) e o [12]*MPI* (Message Passing Interface, Interface de Passagem de Mensagens) (BODEN, 1995).

Para uma maior abstração dos computadores em cluster, um pacote de software *SSI* (Single System Image, Imagem Única do Sistema) pode ser utilizado, funcionando como uma camada entre o sistema operacional e as aplicações (BARROSO, 2003).

A interligação de *clusters* recai fortemente no âmbito do *software* usado para permitir que o sistema funcione realmente como um único sistema interligando vários computadores distintos (BARROSO, 2003).

Quando tem-se um sistema de memória distribuída, no qual cada processador possui sua própria memória local, é necessário definir um modo de interação entre os processos em andamento. Para sistemas em *cluster*, o sistema de troca de mensagens é largamente usado para esse fim, onde as mensagens requerem os dados a serem copiados que serão executados em [13]*threads* locais (BODEN, 1995).

Segundo Boden (1995), para que o ambiente de troca de mensagens seja empregado em *clusters*, é preciso levar em conta alguns aspectos, como:

- Fraco acoplamento entre computadores;
- Possível diferença na largura de banda de interconexão de nós;
- Atrasos na comunicação de nós;
- Diferenças entre *hardwares*, sistemas operacionais e linguagens dos nós interligadas.

Os ambientes de programação mais importantes e independentes das características dos computadores utilizados são o *PVM* (Parallel Virtual Machine, Máquina Virtual Paralela) e o *MPI* (Message Passing Interface, Interface de Passagem de Mensagens) (BODEN, 1995).

Boden (1995), ainda descreve que para uma maior abstração dos computadores em *cluster*, um pacote de *software SSI* (Single System Imagem, Imagem Única do Sistema) pode ser utilizado, funcionando como uma camada entre o sistema operacional e as aplicações.

---

[11] PVM é a abreviação de Parallel Virtual Machine. Este é um pacote de software que permite que uma rede heterogênea de computadores de todos os tipos seja programada como se fosse apenas uma única "Máquina Paralela Virtual"

[12] Message Passing Interface (MPI) é um padrão para comunicação de dados em computação paralela. Existem várias modalidades de computação paralela, e dependendo do problema que se está tentando resolver, pode ser necessário passar informações entre os vários processadores ou nodos de um cluster, e o MPI oferece uma infraestrutura para essa tarefa.

[13] Thereads - é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentemente.

2.12 CONTROLE DE SEQUÊNCIA E CONTROLE DE ERROS

Segundo Ross (2008) os protocolos de propósito geral (por exemplo, *TCP/IP*) necessitam de mecanismos que garantam uma transmissão confiável e possibilidades de roteamento, criando um grande [14]*overhead* na análise dos pacotes na pilha de protocolos. Dessa forma, efetua-se apenas um controle de sequencia simples e nenhum controle de erros. A quebra na sequencia dos pacotes poderá ocorrer quando: não houver um tratamento da interrupção do *hardware* de rede a tempo e o limite do *buffer* do *hardware* de rede for atingido. Testes práticos revelaram que somente ocorrem perdas de pacotes quando houver muita carga de processamento em disco ou falta de memória, o que não acontece em ambientes de computação paralela. Entretanto, como o sistema operacional *Linux* é considerado não determinístico, é razoável considerar a probabilidade de ocorrência dessas perdas (ROSS, 2008).

A implementação do controle de sequencia é alcançada através de uma simples verificação do número do *fragmento* do pacote. Caso ocorra uma quebra de sequencia, a [15]*softirq* de recepção do *cluster* na máquina receptora envia um pacote de controle para a máquina transmissora informando que certo pacote não foi processado e que deve ser retransmitido. Esse pacote sempre será atendido na máquina transmissora, pois o ambiente utilizado é [16]*Full-Duplex* e o [17]*buffer* do *hardware* de rede do transmissor sempre conseguirá armazenar esse pacote. (YOKOKURA, 2007)

Uma das grandes preocupações na implementação de um programa que será executado em um *cluster* de computadores é garantir que nenhum computador ficará sobrecarregado, executando uma tarefa maior que os outros computadores do *cluster*. Um dos problemas envolvidos nisso é o fato de não existir uma maneira de se fazer o acompanhamento do andamento do programa durante a sua execução. Uma informação muito importante que poderia ser passada é o tamanho dos dados passados entre os computadores do *cluster*. Se houvesse essa informação, seria instantânea a detecção de que um determinado computador no *cluster* está sobrecarregado, pois se ele estiver recebendo muito mais dados a serem

[14] Overhead - é geralmente considerado qualquer processamento ou armazenamento em excesso, seja de tempo de computação, de memória, de largura de banda ou qualquer outro recurso que seja requerido para ser utilizado ou gasto para executar uma determinada tarefa. Como consequência pode piorar o desempenho do aparelho que sofreu o overhead.
[15] Softirq – É um temporizador dentro do Linux, onde gerencia quando e qual o tempo de processamento de cada pacote.
[16] Full Duplex é um sistema de comunicação composto por dois interlocutores que podem comunicar entre si em ambas direcções. Diz-se, portanto, bidireccional. Note-se, contudo, que um sistema composto por mais de dois interlocutores, ainda que suporte bidireccionalidade entre cada um deles, não se diz duplex.
[17] Buffer - uma região de memória temporária utilizada para escrita e leitura de dados.

processados que os outros computadores do cluster estará realizando um processamento muito maior, ou se receber uma quantidade muito pequena de dados estará realizando um processamento muito menor. (ROSS, 2008)

2.13 TROCA DE MENSAGEM

O paradigma de troca de mensagens tem sido tradicionalmente empregado em sistemas fracamente acoplados, representados pelas arquiteturas baseadas em memória distribuída (*clusters*), em que cada processador tem acesso somente à sua memória local e dá-se por meio de uma rede de interconexão (CLARKE et al., 2001).

Conceitualmente, a ideia de troca de mensagens é totalmente independente de *hardware*, sistema operacional, linguagens de programação e bibliotecas. Quanto ao desenvolvimento de um programa paralelo por troca de mensagens, o programador deve distribuir os dados explicitamente entre os processadores. Visto que os espaços de endereçamento dos processos que compõem o programa paralelo são distintos, concebeu-se a abstração de mensagem, que pode ser enviada de um processo a outro por um canal de comunicação. Tal conceito é denotado no programa através de primitivas do tipo *send (*envia*)* e *receive* (recebe*)*, as quais supõem um processo que pretende enviar (*send*) uma mensagem a outro, que espera recebê-la (*receive*). Entre os processos comunicantes diz-se que existe um canal de comunicação.

Na prática, os programadores dispõem de bibliotecas de comunicação com primitivas à semelhança de *send* e *receive*. As bibliotecas de comunicação que obtiveram maior aceitação foram, inevitavelmente, [18]*MPI* (CLARKE et al.,2001) e [19]*PVM* (GEIST et al., 2001), ambas com suporte às linguagens de programação C e *Fortran* (BEOWULF, 2009).

MPI (*Message Passing Interface*) "Mensagem passando em Interface" é, na verdade, um padrão estabelecido pelo *MPI*, especificando a sintaxe e semântica de um conjunto de funções de comunicação que, em um dado momento, julgou-se atender às necessidades de aplicações paralelas típicas. Existem diversas implementações do padrão *MPI*, para as mais variadas arquiteturas (CLARKE et al.,2001).

---

[18] MPI - É um padrão de troca de mensagens portátil que facilita o desenvolvimento de aplicações paralelas.

[19] MPV - é um padrão de troca de mensagens portátil que facilita o desenvolvimento de aplicações paralelas.

O padrão *MPI* (*Message Passing Interface*) “Mensagem passando em Interface” incorporou os modelos de comunicação por troca de mensagens habitualmente descritos na literatura, prevendo tanto a comunicação síncrona quanto a assíncrona, permitindo, em qualquer dos casos, o uso de primitivas do tipo *send* e *receive* bloqueantes ou não-bloqueantes. Além disso, *MPI* (*Message Passing Interface*) “Mensagem passando em Interface” destaca-se pelo excelente suporte à comunicação coletiva, que facilita o desenvolvimento de aplicações que necessitam de efetuar frequentes operações sobre matrizes. (BEOWULF, 2009).

No modelo de programação, [20]*MPI* (*Message Passing Interface*) “Mensagem passando em Interface” segue a filosofia [21]*SPMD* (*Single Program Multiple Data*), de modo que os processos componentes do programa paralelo apresentam exatamente o mesmo código; adicionalmente, não é permitida a criação ou destruição de processos durante a execução da aplicação. Em outras palavras, o modelo de processos é estático (CLARKE et al., 2001).

Outra biblioteca de comunicação notoriamente difundida na área de Processamento Paralelo e Distribuído é *PVM* (*Parallel Virual Machine*) (CLARKE et al., 2001).

Diferentemente de *MPI* (*Message Passing Interface*), *PVM* (*Parallel Virual Machine*) não é um padrão formal, embora as implementações mais recentes estejam efetivamente convergindo para uma padronização (CLARKE et al., 2001) *PVM* (*Parallel Virual Machine*) é mais do que uma biblioteca de comunicação. Trata-se de um *software* básico com suporte próprio ao gerenciamento de processos e centrado no conceito de máquina virtual. O usuário do ambiente *PVM* (*Parallel Virual Machine*) pode adicionar máquinas físicas de uma rede local ou *cluster* — à máquina virtual *PVM* (*Parallel Virual Machine*)Após proceder às devidas configurações da máquina virtual, pode-se disparar processos que fazem uso das primitivas da biblioteca de comunicação *PVM* (*Parallel Virual Machine*) (BEOWULF, 2009).

A grande diferença entre *PVM* (*Parallel Virual Machine*). e *MPI* (*Message Passing Interface*) está no modelo de processos. *PVM* suporta o modelo de programação *MPMD* (*Multiple Program Multiple Data*), possibilitando a criação e destruição de processos em tempo de execução da aplicação paralela. Em contrapartida, os serviços de comunicação

[20] MPI - é um padrão para comunicação de dados em computação paralela. Existem várias modalidades de computação paralela, e dependendo do problema que se está tentando resolver, pode ser necessário passar informações entre os vários processadores ou nodos de um cluster, e o MPI oferece uma infraestrutura para essa tarefa.

[21] SPMD (único processo, dados múltiplos, ou único programa, dados múltiplos) é uma técnica empregada para atingir o paralelismo , é uma subcategoria de MIMD . As tarefas são divididas e executadas simultaneamente em vários processadores com entrada diferente a fim de obter resultados mais rápidos.

ponto a ponto e comunicação coletiva são ruins, quando comparados com os da biblioteca *MPI*. Os grupos de processos *PVM* são dinâmicos e abertos, ao passo que em *MPI* os grupos são eminentemente estáticos (CLARKE et al.,2001).

Desde o projeto inicial, em 1989, *PVM* sempre favoreceu mecanismos de interoperabilidade, ensejando a construção de uma máquina virtual única, possivelmente composta por computadores de diferentes arquiteturas. Neste contexto, os *daemons* de comunicação *PVM* exercem um papel fundamental. Ao contrário, o padrão *MPI* procurou promover um equilíbrio entre facilidade de uso e desempenho para aplicações paralelas (CLARKE et al.,2001).

As tendências atuais apontam para a união das bibliotecas *MPI* e *PVM*.

Recentemente, foi proposto o padrão *MPI*-2, agregando ao original o suporte ao modelo *MPMD*, além de primitivas para o compartilhamento de memória, mesmo em ambientes fisicamente distribuídos. De modo análogo, *PVM* também está evoluindo na direção de *MPI*, com a introdução de melhores serviços de comunicação coletiva. Sendo assim, há correntes que advogam a junção de ambas as bibliotecas em um ambiente de programação único e abrangente em termos de aplicações e serviços fornecidos (CLARKE et al.,2001)d.

2.14 BIBLIOTECA MPI

A biblioteca de *Message Passing* foi desenvolvida para ambientes de memória distribuída, máquinas paralelas massivas, *network of workstations* e redes heterogêneas. *MPI* define um conjunto de rotinas para facilitar a comunicação (troca de dados e sincronização) entre processos paralelos. A biblioteca *MPI* é portável para qualquer arquitetura, tem aproximadamente 125 funções para programação e ferramentas para se analisar a performance (MICHAEL, 2004).

A biblioteca *MPI* possui rotinas para programas em Fortran 77 e *ANSI C*, portanto pode ser usada também para Fortran 90 e C++. Os programas são compilados e linkados com a biblioteca *MPI*. Todo paralelismo é explícito, ou seja, o programador é responsável por identificar o paralelismo e implementar o algoritmo utilizando chamadas aos comandos da biblioteca *MPI* (CLARKE, 1999).

O *MPI* básico contém seis funções básicas indispensáveis para o uso do programador, permitindo escrever um vasto número de programas em *MPI*. O *MPI* avançado contém cerca de 125 funções adicionais que acrescentam às funções básicas a flexibilidade (permitindo tipos diferentes de dados), *robustez* (modo de comunicação *non-blocking*), eficiência (modo

ready), modularidade através de grupos e comunicadores (*group* e *communicator*) e conveniência (comunicações coletivas, topologias). A seguir, algumas características do padrão *MPI*: (MICHAEL, 2004).

Eficiência - Foi cuidadosamente projetado para executar eficientemente em máquinas diferentes. Especifica somente o funcionamento lógico das operações. Deixa em aberto a implementação. Os desenvolvedores otimizam o código usando características específicas de cada máquina (CLARKE, 1999).

Facilidade - Define uma interface não muito diferente dos padrões *PVM*, *NX*, Express, P4, etc. e acrescenta algumas extensões que permitem maior flexibilidade.

Portabilidade - é compatível para sistemas de memória distribuída, *NOW*s (*network of workstations*) e uma combinação deles (MICHAEL, 2004).

Transparência - Permite que um programa seja executado em sistemas heterogêneos sem mudanças significativas (MICHAEL, 2004).

Segurança - Provê uma interface de comunicação confiável. O usuário não precisa preocupar-se com falhas na comunicação (MICHAEL, 2004).

Escalabilidade - O *MPI* suporta escalabilidade sob diversas formas, por exemplo: uma aplicação pode criar subgrupos de processos que permitem operações de comunicação coletiva para melhorar o alcance dos processos (CLARKE, 1999).

2.15 OPENMOSIX

O projeto *Mosix* - Multicomputer Operating System IX - é um sistema operacional distribuído, desenvolvido originalmente pelos estudantes do professor Amnom Barak, na Universidade Hebrew em Jerusalém, Israel. Foi utilizado nos anos 80 pela força área americana para a construção de um *cluster* de computadores *PDP*11/45. O projeto foi desenvolvido sete fases, para diferentes versões de *UNIX* e arquiteturas de computadores. A primeira versão para *PC* foi desenvolvida para o *BSD/OS*. A última versão foi para o sistema operacional *Linux* em plataforma Intel O *OpenMosix* é uma extensão do projeto *Mosix*, baseado no *GPLv2*, iniciado em 10 de fevereiro de 2002, coordenado pelo Ph.D Moshe Bar, para manter os privilégios desta solução *Linux* para *cluster* disponível com *software* de código aberto (OPENMOSIX, 2005).

Segundo Ferreto (2002), este agrupamento de máquinas *Linux* é o que poderia classificar de verdadeiro sistema de imagem simples ([22]*SSI - Single System Imagem*), pois já é clara que a ideia de que não se têm um *cluster* verdadeiro enquanto não existir um *SSI*. Pode-se ter como referência os primeiros *clusters SSI* como o *IBM SysPlex* e o *cluster DEC*. Em um *cluster DEC,* quando dado um comando *telnet* para um endereço no *cluster* e essa chamada será atendida por qualquer nó do *cluster*, e o usuário não precisa se preocupar com qual nó que irá atender esta chamada, e qualquer programa iniciado pelo usuário será executado no nó que possuir maior disponibilidade de recursos para atender ao programa.

O *OpenMosix* é uma extensão do núcleo do sistema operacional *Linux,* que faz com que um *cluster* de computadores se comporte como um grande e único supercomputador através da utilização de migração *preemptiva* de processos e balanceamento dinâmico de carga. A implementação da Migração *Preemptiva* de processos é capaz de migrar qualquer processo do usuário, em qualquer instante e para qualquer nó disponível de maneira transparente. Para atingir um melhor desempenho este é controlado por Algoritmos de Balanceamento Dinâmico de Carga e de prevenção contra falta de memória. Estes algoritmos são projetados para responder dinamicamente as variações da utilização dos recursos nos diversos nós. Isto garante que o cluster se comporte muito bem, seja numa configuração com poucas ou com muitas máquinas, propiciando uma maior escalabilidade (OPENMOSIX, 2005).

Segundo *OpenMosix* 2005, o programa que roda em uma máquina consumir muitos recursos, o sistema varre toda rede e procura uma máquina mais "disponível no momento" em termos de memória e *CPU,* e desloca seu "programa" ou parte dele para ser executado remotamente. Com isso, o sistema ganha desempenho.

2.16 TRANSMISSÃO

Como cita Kurose (2006) na transmissão, o protocolo cria [23]*frames Ethernet* utilizando as informações contidas na estrutura de controle. A mensagem é colocada no pacote *Ethernet* a partir da região de memória do processo usuário. Caso a mensagem seja maior que a área de dados do pacote *Ethernet* com os cabeçalhos, há a necessidade da geração de fragmentos. Ao ter um pacote montado, efetua-se a transmissão do mesmo por meio da fila de transmissão do

[22] SSI - é um conjunto de máquinas que parece ser um único sistema. O conceito é muitas vezes considerado sinônimo de que de um sistema operacional distribuído, mas uma única imagem podem ser apresentadas para fins mais limitados, apenas programação de trabalho , por exemplo, que pode ser conseguido por meio de uma camada adicional de software sobre convencionais imagens do sistema operacional de funcionamento em cada nó.

[23] Frames Ethernet são "envelopes" para os pacotes TCP/IP.

dispositivo de rede *Ethernet*, especificado no sistema de rede do *Linux* ([24]*pfifo_fast_queue*). Os pacotes dessa fila são transmitidos pela rotina do *driver* de rede, até que a fila se esvazie ou até que informe uma sobrecarga. Caso ocorra uma sobrecarga, a transmissão dos pacotes é interrompida.

Quando o *hardware* de rede se encontrar apto a realizar uma nova transmissão, este informa ao *driver*, que libera a fila para novo processamento e marca a *softirq* de transmissão do *Linux* para continuar o processamento da fila interrompida. (PERRY, 2005)

Existem no sistema operacional *Linux* desde o [25]*kernel* 2.3.43, podem ser consideradas rotinas de sistema que executam em nível de interrupção. O objetivo principal delas é efetuar a grande parte das tarefas de um tratador de interrupção (PERRY, 2005).

2.17 REDES DE CONEXÃO

Como os *clusters* são compostos por um conjunto de computadores interligados, um dos gargalos mais sérios para o seu desempenho é a rede que interconecta esses computadores. Nesta seção, são apresentados três modelos de redes de alto desempenho bastante utilizados em *clusters*, a Myrinet, Gigabit Ethernet e Infiniband (PERRY, 2005).

2.18 ETHERNET E TCP/IP

Como cita Perry (2005), *ethernet* vem sendo a principal tecnologia de redes locais desde o início dos anos oitenta, e é usada por todas as empresas, que tenham uma infraestrutura mínima de rede. Embora os seus protocolos básicos tem mudado pouco, novas opções como o [26]*Fast Ethernet* (100Mbps) e o [27]*Gigabit Ethernet* (1 *Gbps*), aliados aos baixos preços dos equipamentos tem garantido uma sobrevida fenomenal ao *Ethernet*.

Segundo Perry (2005), o nome oficial do padrão *Ethernet* é *IEEE* 802.3, padrão este que foi publicado em 1985 com o título: *IEEE* 802.3, já o suplemento 802.3i-1990 descreve a utilização do 802.3 em banda base a 10Mbps com par trançado categoria 3 ou superior (10Base-T), o qual é implementado pelo controlador 802.3 usado.

---

[24] Pfifo_fast_queue – é um conjunto de três bandas movimentando pacotes dentro da rede do Linux.
[25] Kernel – Nucleo central do sistema operacional do Linux.
[26] Fast Ethernet é um termo para vários padrões da Ethernet que levam o tráfego de dados à taxa nominal de 100 Mbit/s, contra a taxa de transmissão de 10 Mbit/s da Ethernet original.
[27] Gigabit Ethernet - é o termo que descreve várias tecnologias para transmissão de quadros em uma rede a uma velocidade de Gigabit por segundo definido no padrão IEEE 802.3-2005.

Sistemas 10 Base-T como o nosso funcionam muito bem na grande maioria das redes, mas é recomendada a utilização de [28]switches ou [29]switching-hub, para isolar a rede10Base-T das outras de maior velocidade (MIERS, 2003).

O *TCP/IP* é uma suíte de protocolos que permite que computadores de todos os tipos, rodando sistemas operacionais totalmente distintos comuniquem entre si. Surgiu como uma pesquisa financiada pelo governo de redes por chaveamento de pacotes e acabou se tornando a maior estrutura tecnológica já criada pela humanidade, envolvendo o mundo inteiro. É um sistema aberto no sentido de que seus padrões são divulgados e podem ser implementados livres de qualquer ônus. Estas facilidades levaram ao surgimento da Internet (PERRY 2005).

O *TCP/IP* é uma suíte de protocolos de quatro camadas apenas: enlace, rede, transporte e aplicação. O controlador ethernet; seu driver e o protocolo *ARP* formam a camada de enlace no nosso caso (PERRY, 2005).

A ampla aceitação e utilização da *Ethernet* e do *TCP/IP*, levaram algumas empresas a implementar estes protocolos em chips de baixo custo para aplicações embutidas, devido ao surgimento de tais chips a aplicação se tornou viável (MIERS, 2003).

3 FERRAMENTAS DE RENDERIZAÇÃO

Sabe-se que a origem da palavra renderizar é originária da palavra inglesa render sendo um termo bastante usado na computação gráfica para fazer menção ao convertimento

[28] Switch - é um dispositivo utilizado em redes de computadores para reencaminhar módulos (pacotes) entre os diversos computadores da rede.
[29] Switching-hub - é uma evolução do *hub*. Em termos de tecnologia e, dependendo do tipo de rede onde é usado, o *hub-switch* pode ser considerado mais avançado.

de um tipo de arquivo para outro, ou até mesmo a "tradução" de uma linguagem para outra (GONZALEZ, 1993).

Nesse processo de renderização de uma determinada cena é preciso fazer definição de muitas "variáveis", por exemplo os objetos presentes na cena, suas texturas, sua cor, reflexão, transparência e fazer a localização de pontos de iluminação. Pode-se resumir o processo de renderização como sendo uma forma de converter modelos gráficos em uma imagem. Nesse ato de renderizar, os programas responsáveis, fazem cálculos diversos, como da perspectiva dos planos, das sombras e luzes do objeto (PLOE, 2006).

O principal motivo da utilização de renderização nessa monografia é o fato desse ato sempre exigir uma grande capacidade computacional, podendo tirar proveito, até certo ponto, dos dois temas focos: *clusters* e arquiteturas multicore. Esse capítulo não tem por objetivo se aprofundar nos tipos de ferramentas de renderização, pois esse não é o foco do trabalho, e sim fazer uso dessas ferramentas para servirem de estudo para *clusters* e arquiteturas multicore (GONZALEZ, 1993).

3 .1 COMO SURGIU O BLENDER

Em 1988 Ton Roosendaal co fundou o estúdio de animação Holandês NeoGeo. NeoGeo rapidamente se tornou o maior estúdio de animação em 3D na Holanda e uma das casas de animação de liderança na Europa. O estúdio NeoGeo criava premiadas produções (European Corporate Video *Awards* 1993 e 1995) para grandes clientes corporativos, tais como multi nacionais *Philips Electronics Company*. Dentro da NeoGeo, Ton foi responsável tanto pela direção de arte e desenvolvimento interno de software. Após cuidadosa deliberação Ton decidiu que a atual ferramenta 3D interna usada pela NeoGeo era demasiadamente velha e pesada para manter e atualizar, e necessitava ser reescrita do zero. Em 1995 esta reescrita começou e foi destinada a ser a criação da suíte de software 3D que todos nós conhecemos agora como *Blender*. Enquanto a NeoGeo continuava a refinar e melhorar o *Blender*, tornou-se claro para Ton que o *Blender* poderia ser usado como uma ferramenta para outros artistas fora da NeoGeo. (JOOMLA, 20011)

No entanto domingo, 13 de outubro de 2002, o *Blender* foi liberado para o mundo sob os termos do GNU General Public License (JOOMLA, 20011).

O *Blender* é uma ferramenta que permite a criação de vastos conteúdos de 3D. Oferece funcionalidades completas para modelagem, renderização, animação, pós produção, criação e visualização de conteúdo 3D interativo, com os benefícios singulares de portabilidade numa aplicação.

Dirigido a profissionais e artistas desta área, o *Blender* pode ser utilizado para criar visualizações de espaços tridimensionais, imagens estáticas, bem como vídeos de alta qualidade, incorpora ainda um motor 3D em tempo real, que permite também a criação de conteúdo 3D interativo.

3.2 ORIGEM DO FRAME

Segundo Thalmann (1992), em inglês, fala-se em "*film frame*" ou "vídeo *frame*", conforme o produto em questão tenha sido realizado em película (tecnologia cinematográfica) ou vídeo (tecnologia eletrônica, seja ela analógica ou digital). Em produção audiovisual, é comum usar-se a palavra "*frame*" também como unidade de tempo. Neste caso, sua definição depende da cadência de projeção utilizada em cada sistema particular de filme ou vídeo.

Por exemplo, no cinema sonoro (desde 1929), a cadência padrão de projeção é de 24 qps (quadros por segundo), portanto um *frame* equivale a 0,0417 segundos. Mas, no período do cinema mudo, a cadência variava entre 16 e 20 quadros por segundo, dando ao *frame* uma duração entre 0,0500 e 0,0625 segundos. Nos sistemas de vídeo utilizados no Brasil, Estados Unidos, Canadá e Japão (NTSC e variantes), a cadência é de 29,97 imagens por segundo, portanto cada *frame* equivale a 0,0333 segundos. Mas os sistemas de vídeo europeus *SECAM e PAL* (também utilizados na maior parte da América Latina) trabalham com uma cadência de 25 quadros por segundo, o que faz com que cada *frame* dure 0,0400 segundos (THALMANN 1992).

4 METODOLOGIA

Neste capítulo é abordado os aspectos relacionados às atividades metodologicas da monografia. Assim, serão descritas as configurações do *cluster*, as ferramentas de renderização.

Figura 8 – Estrutura pronta do Cluster.
Fonte: Elaborado pelo autor.

A Figura 8, mostra a estrutura física do *cluster*, composta por dois computadores de mesa, um *notebook*, um chaveador, um roteador e um monitor padrão CRT.

4.1 ESTRUTURA FÍSICA DO CLUSTER

Foi elaborado uma estrutura de *clusters*, que foi comparado a uma máquina normal, com a finalidade de realizar testes com a ferramenta de renderização. O número máximo de nós utilizados foram 3, onde o *notebook* era o [30]nó mestre gerenciando as entradas de processamento e mais dois PC's de mesa como escravo, com as seguintes configurações:

Nó Master, um Notebook:	
CPU	Intel i3 core CPU
Memoria Cachê	4MB
Memoria Ram	4 GB DDR2 800 MHz
Placas de rede	10/100/1000
Disco rígido	500 GB SATA

Figura 9 – Nó Mestre.
Fonte: Elaborado pelo autor.

Na Figura 9 descreve em detalhes a parte de *hardware* do nó mestre do cluster.

[30] Nó – é a representação de cada computador dentro do cluster.

1º nó escravo, um PC de mesa:	
CPU	Intel Cor 2 Duo 2,33 GHz
Memoria Cachê	1 MB
Memoria Ram	3 GB DDR 667 MHz
Placas de rede	10/100/1000
Disco rígido	1,5 Tera Sata

Figura 10 – 1º Nó Escravo.
Fonte: Elaborado pelo autor.

Na Figura 10 descreve em detalhes a parte de *hardware* do 1º nó escravo do *cluster*.

2º nó escravo, um PC de mesa	
CPU	AMD 2,1 GHz
Memoria Cachê	512 MB
Memoria Ram	1 GB DDR 667 MHz
Placas de rede	10/100/1000
Disco rígido	500GB 7200RPM Sata

Figura 11 – 2º Nó Escravo.
Fonte: Elaborado pelo autor.

Na Figura 11 descreve em detalhes a parte de *hardware* do 2º nó escravo do *cluster.*

De forma a gerenciar todos os nós, obteve-se um chaveamento KVM 4 portas, modelo TK 400, um dispositivo eletrônico que auxilia no comando de uma só vez 4 computadores simultaneamente, com objetivo de diminuir a quantidade de monitor, mouse e teclado e consequentemente, ter um maior espaço físico.

Figura 12 – Chaveador.
Fonte: Elaborado pelo autor.

Como mostra na Figura 12, um chaveador, onde através desta tecnologia foi feito o controle de dois computador de mesa e um *notebook*, com apenas um monitor *CRT*.

Os nós pertencentes a todas as estruturas do *cluster* eram homogêneos com as configurações descritas na Figura 13.

INFRAESTRUTURA DE REDE	
Fast Ethernet	10/100 Mbps
Switch	10/100 8 portas
Cabo par trançado	10/100
Conector	RJ45
Protocologo	TCP/IP

Figura 13 – Infraestrutura de Redes de Computadores.
Fonte: Elaborado pelo autor.

Nós com configurações iguais não são necessários, mas já ajudam a evitar, por exemplo, problemas decorrentes de compilação de um código para arquiteturas diferentes, ou por ter também uma carga de trabalho desbalanceada, onde os nós mais rápidos irão terminar primeiro que os mais lentos, de modo que os mais rápidos terão que ficar esperando. Assim, a homogeneidade não é uma condição necessária para um *cluster*, mas é um fator que irá reduzir bastante à quantidade de problemas.

Os nós estão interligados por um *switch* 10/100 Mbps e por cabos padrão *Ethernet.*

4.2 CONFIGURAÇÃO DO CLUSTER

Desde o início da escolha do tema buscou-se um *software* que fosse uma alternativa prática e eficiente para montagem e realização dos estudos com *clusters*.

A tecnologia de *cluster* escolhida foi um SO Server *Linux* 11 x 86, com a ferramenta [31]DrQueue como forma de paralelismo e rodando [32]*kernel* versão 2.4.

A configuração dos nós consistia basicamente a máquina mestre que faz a distribuição com [33]Ubuntu Server 11 x 86, havendo uma comunicação para cada computador componente do cluster.

Na tabela 1 abaixo tem-se o passo a passo dos comandos de configuração de rede utilizados nesse processo.

[31] DrQueue – Software responsavel pela distriuição e balanceamento de carga no cluster.
[32] kernel versão 2.4 – Dentre o Linux Sever que estava sendo utilizado no cluster, a versão do nucleo era de 2.4.
[33] Ubuntu Server 11 x 86 – É uma versão server do Linux onde dentro do trabalho desenvolvido teve a função de atuar como um sistema operacional, gerindo todas as tarefas e softwares.

$ sudo su
ifconfig eth0 192.168.0.1 netmask 255.255.255.0
route add default gw 192.168.0.1

Tabela 1. Configuração os nós do cluster.
Fonte: Elaborado pelo autor.

Os IPs foram atribuídos em sequencia dos endereço privado, e atribuindo ao primeiro [34]*host*, escolhido na ordem com que os computadores se encontravam distribuídos nos computadores, o valor 1, conforme mostra a tabela acima. Não há nenhuma restrição quanto a essa atribuição exigida pelo *DrQueue*, esta foi apenas uma convenção adotada para saber qual IP estava relacionado a cada máquina.

Apesar de não ser preciso alcançar outras redes a não ser a rede local do *cluster*, foi necessária a configuração do [35]*gateway* padrão "porta de ligação, é uma máquina intermediária geralmente destinada a interligar redes, separar domínios de colisão, ou mesmo traduzir protocolos" para o funcionamento do sistema de autodescoberta de nós.

Depois de ser feito o processo de configuração, foi executado o *software* de gerenciamento *DrQueue*. Através dessa ferramenta pode-se visualizar todos os *IPs* dos nós que compõe o *cluster*, e poder saber se estes nós estão ativos ou não, através de uma numeração do lado esquerdo dos *IPs*. No caso da Figura 14, todos os nós estão ativos, como mostra o [36]*ETH0, ETH1, ETH2*.

[34] Host ou hospedeiro - é qualquer máquina ou computador conectado a uma rede, podendo oferecer informações, recursos, serviços e aplicações aos usuários ou outros nós na rede. No nó, é o responsável por implementar a estrutura da camada de rede de endereçamento.
[35] Gateways padrão - desempenham um papel importante na rede TCP/IP. Eles fornecem uma rota padrão para os hosts TCP/IP usarem durante a comunicação com outros hosts em redes.
[36] ETH0,1,2 – Nomenclatura utilizada dentro do Software DrQueue para cada computador na rede, sendo distribuido.

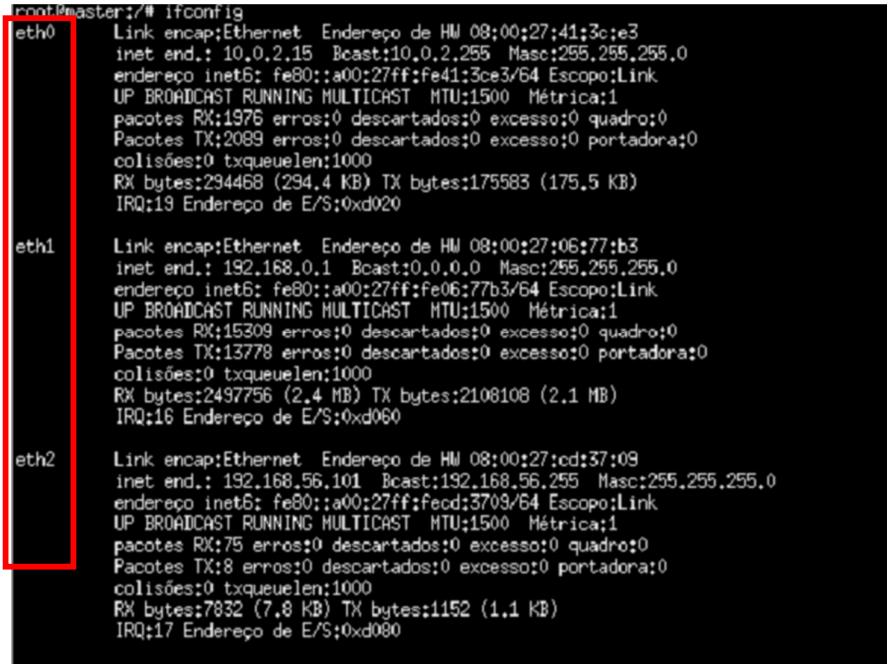

Figura: 14 – Imagem dos nós ativos.
Fonte: Elaborado pelo autor.

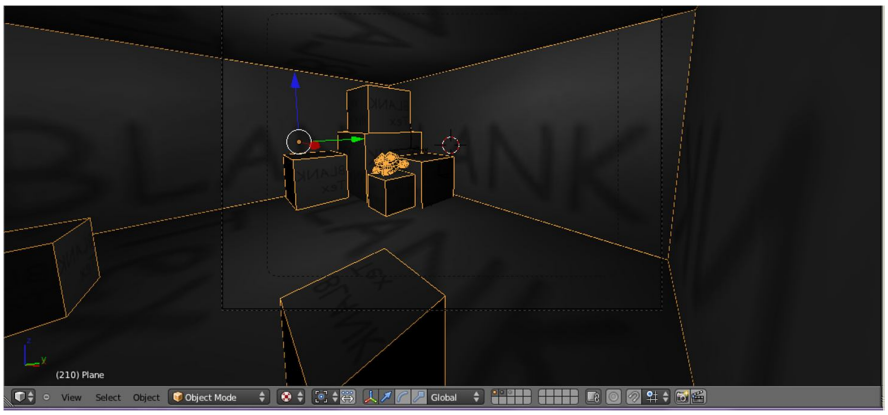

Figura 15 – Exemplo 1 quarto geométrico (0.413GB).
Fonte:Blender ([1995]).
Nota: Adaptado pelo autor.

Já na Figura 16 tem-se o 2º exemplo de renderização, onde uma sala mais elaborada com várias texturas, algumas caixas e uma arma, simula um ambiente de jogo, tendo assim 1000 *frames*, no formato *blender*, fontes extraídas do blender.

Figura: 16 – Exemplo 2 galpão de guerra (0.850GB).
Fonte:Blender ([1995]).
Nota: Adaptado pelo autor.

A Figura 17 com o 3º exemplo de renderização, tem-se um uma sala com uma escadaria toda em texturas, parede com texturas e iluminação, a frente uma esfera, tendo 1.148 *frames*, no formato *blender*, fontes extraídas do blender.

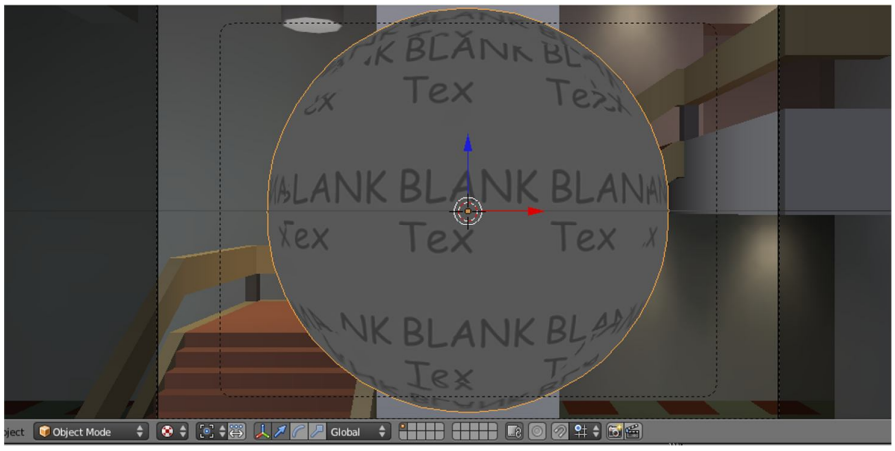

Figura: 17 – Exemplos 3 sala cenográfica (1.900GB).
Fonte:Blender ([1995]).
Nota: Adaptado pelo autor.

Na Figura 18 tem-se o 4º exemplo para renderização, com um chão de textura e efeitos, um boneco com várias formas geométricas, cores e textura, tendo assim 1.350 *frames*, no formato *blender*, fontes extraídas do blender.

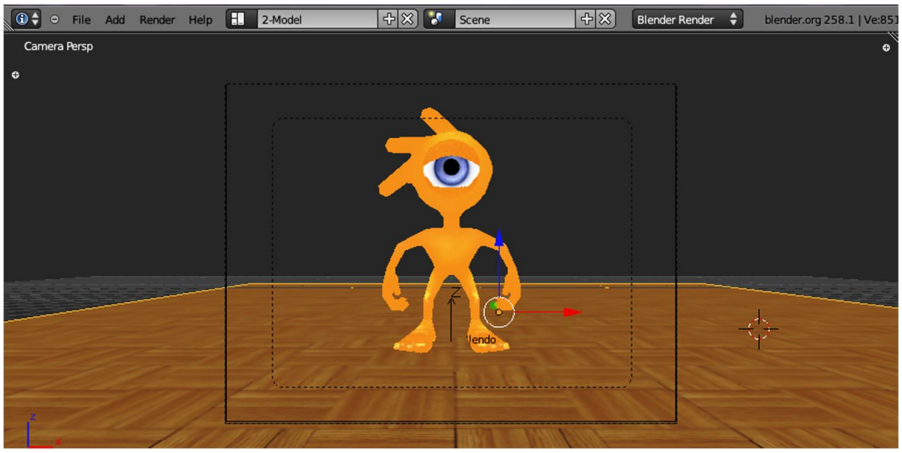

Figura: 18 – Exemplo 4 boneco blender (2.010GB).
Fonte:Blender ([1995]).
Nota: Adaptado pelo autor.

4.4 RESULTADOS

Através da utilização da ferramenta e coleta de dados, das figuras exemplo 1 quarto geométrico, exemplo 2 galpão de guerra, exemplo 3 sala cenográfica e exemplo 4 boneco blender, citado nas paginas 46 e 47, foi identificado o poder computacional do *cluster*, como mostra a Tabela 2.

4.3 CLUSTER VERSUS DRQUEUE

O *DrQueue* é um programa, desenvolvido para paralelizar, num ambiente de *cluster*, onde faz a interface entre o *Blender* e o *Cluster*. Ele também consiste na divisão de uma cena em um conjunto de sub tarefas, (que serão processos independentes) possibilitando a migração de processos entre os nós do *cluster*.

Para a instalação e configuração do *DrQueue*, foi necessário verificar as dependências para sua compilação:

Foi necessário setar duas variáveis de ambiente, que são: / etc/bashrc #export

DRQUEUE_ROOT=”/mnt/shared/drqueue”

/* Essa variável seta o diretório de instalação.

*/

#export DRQUEUE_MASTER=”192.168.0.1”

/* Essa variável seta o endereço de rede da máquina master */

Foi necessário criar um usuário *DrQueue* que será usado por questão de direitos de acesso, caso contrário o script de instalação não irá compilar os arquivos.

#useradd drqueue /* não precisa atribuir senha para ele com o passwd*/

O local de instalação do *DrQueue* deve ser compartilhada e montada em todas as máquinas no mesmo ponto, assim todas as máquinas terão uma pasta /mnt/shared/drqueue que apontarão para a mesma máquina no caso a mestre, para onde serão enviadas as renderizações.

Como usuário root:

#make

#make INSTROOT=$DRQUEUE_ROOT install

Utilização:

-Para iniciar o mestre foi utilizado o aplicativo “/mnt/shared/drqueue/bin/master”

-Para iniciar o escravo foi utilizado “/mnt/shared/drqueue/bin/slave”

-Para enviar um *job*, foi utilizado o drqman : “/mnt/shared/drqueue/bin/drqman”

Figura: 19 – Tela de apresentação do DrQueue
Fonte: Elaborado pelo autor.

Na Figura 19 tem-se as funções onde apresenta-se o *DrQueue* com as seguintes colunas e funções: ID identificação do nó dentro do *cluster*, *name* nome do nó atribuído, *status* se o nó está ativo ou não, *processors* quantidade de núcleos do nó, *left* visualiza a quantidade de jobs atribuída, *done*, a quantidade de jobs executadas, *failed*, a quantidade de jobs com problemas de execução, total quantidade do total de tarefas atribuídas para aquele determinado nó, *PRI* rotinas processadas no momento, *pool status* do nó dentro do *Cluster*.

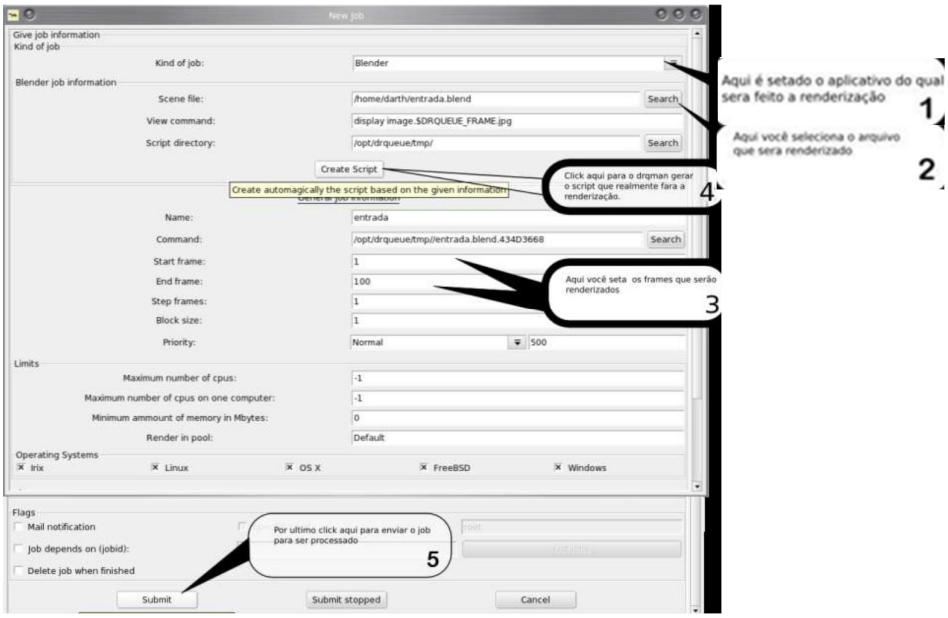

Figura: 20 – Tela de set de processo.
Fonte: Elaborado pelo autor.

Na figura 20 tem-se as funções onde apresenta-se a tela de entrada de processos, dentro do *cluster* com os campos, *Kid of job* campo para seta o tipo de arquivo que vai ser processado, *Scane file* seta o arquivo a ser processado, *Start Frame* e *End Frame* seta a quantidade de *frames* que vão ser processada, *Create Script* botão que gera um diagrama do processo, *Submit* botão onde executa o processo dentro do *cluster*.

Para que seja analisado, se uma tarefa já tenha sido finalizada, é necessário verificar se o camp *Lef* está com o valor zero, onde mostra o termino da distribuição e renderização.

Para utilizar o *Blender*, foi necessário baixar seu código-fonte e compilá-lo. Após a compilação, o programa foi executado e configurado na plataforma linux.

Logo abaixo estaremos visualizando os projetos renderizado no *cluster*, a fim de quantificar sua capacidade de processamento.

Na Figura 15 temos o 1º exemplo de renderização, onde temos um quarto em 3D com algumas caixas, tendo 900 *frames*, no formato *blender*.

Tabela 2 – Tabela de calculo computacional.

Arquivos	Frame	Taman. dos Proj. GB	Processos
Exemplo 1	900	0,413	13.148
Exemplo 2	1.000	0,850	13.594
Exemplo 3	1.148	1,900	15.356
Exemplo 4	1.350	2,010	18.092
Total	4.398	5,173	60.190
Média	1.100	1,293	15.048

Fonte: Elaborado pelo autor.

Na Tabela 2 é composta pelas seguintes colunas, arquivos, a descrição dos arquivos que foram utilizados, já na coluna *frame* tem-se a quantidade de *Frames* de cada exemplo, na coluna tamn. Dos proj. GB temos o quanto cada projeto representa em GB e na 4º e última coluna processo, extraídos do monitoramento do *DrQueue,* onde mostra a quantidade de processo executado para cada exemplo apresentado.

Baseado na quantidade de 1.100 Frame como mostra na tabela 2, 4ºcoluna, 6º linha, como média, a fim de garantir o calculo; chegou-se a uma capacidade computacional do cluster de 15.048 processos como mostra na tabela 2, 2ª coluna, 6ª linha, como média, para cada 1.100 frames apresentados.

4.5 COMPORTAMENTO DOS NÓS DO CLUSTER, DURANTE A RENDERIZAÇÃO

Depois de passar um tempo os processos começam a migrar para outras máquinas do cluster até chegar em um momento em que todas as máquinas estão sendo utilizadas e desempenhando o seu papel, como pode-se notar na Figura 21 na migração das informações para os ID's 1 e 0. Isso ocorre devido ao algoritmo de balanceamento de carga, de modo a executar em todos os nós, tentando fazer a redução da carga entre os nós, com isso migrando os processos de um nó mais sobrecarregado para um nó que possua mais recursos em disponibilidade no cluster.

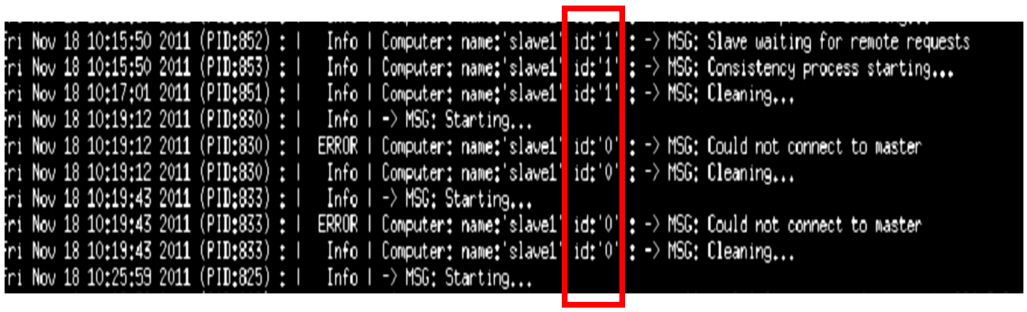

Figura: 21 – As distribuições sendo feita entre os nós do cluster.
Fonte: Elaborado pelo autor.

Com o tempo, depois que todos os nós desempenharem seu papel, voltará apenas um nó a ficar em atividade, normalmente este nó é aquele que executa a ferramenta de *renderização*, e depois deste terminar seu papel a imagem se formará por completo.

De forma a mensurar o quanto de tempo que se ganharia com a construção do c*luster* para *renderização* de imagem.

Lembrando que este teste não foi o foco do estudo, mas sim uma forma de mensurar o quanto significava o ganho na *renderização* com *cluster* em uma máquina comum.

Nos testes realizados, foram feitas variações na quantidade de nós, conforme mostra as tabelas 3 e 4. Na tabela 3, na 4º coluna com o *cluster* completo com 3 nós *renderizando*. Os primeiros testes foram feitos com quatro exemplos, conforme mostra a tabela 3 na 2º coluna.

Tabela 3 – Numeros do 1º Teste com Cluster.

Arquivos	Frame	Taman. dos Proj. GB	Nós
Exemplo 1	900	0,413	3
Exemplo 2	1.000	0,850	3
Exemplo 3	1.148	1,900	3
Exemplo 4	1.350	2,010	3
Total	4.398	5,173	12
Média	1.100	1,293	3

Fonte: Elaborado pelo autor.

No segundo teste foram utilizados os mesmos quatro exemplos como mostra a tabela 4, na 1º coluna, porém com um só nó renderizando, conforme mostra na tabela 4, 4º coluna.

Tabela 4 – Numeros do 2º Teste sem Cluster.

Arquivos	Frame	Taman. dos Proj. GB	Nós
Exemplo 1	900	0,413	1
Exemplo 2	1.000	0,850	1
Exemplo 3	1.148	1,900	1
Exemplo 4	1.350	2,010	1
Total	4.398	5,173	4
Média	1.100	1,293	1

Fonte: Elaborado pelo autor

Como mostra na tabela 5, foram feitas 8 séries de testes e tirado uma média entre elas, para aumentar a confiabilidade nos resultados apresentados. O tempo total de simulação no *cluster* foi de aproximadamente de 17 horas de processamento, como mostra na tabela com *cluster* 5ª coluna 6ª linha, já na máquina comum mais de 66 horas de processamento, como mostra na tabela sem *clustes* 5ª coluna 5ª linha.

Tabela 5 – Tempo de Processamento em Hóras do Cluster Versos Sem Cluster.

ComCluster

Arquivos	Frame	Processos	Taman. dos Proj. GB	Tempo Horas	Nós
Exemplo 1	900	13.148	0,413	3,7	3
Exemplo 2	1.000	13.594	0,850	3,8	3
Exemplo 3	1.148	15.356	1,900	4,3	3
Exemplo 4	1.350	18.092	2,010	5,0	3
Total	4.398	60.190	5,173	16,7	12
Média	1.100	15.048	1,293	4	3

Sem Cluster

Arquivos	Frame	Processos	Taman. dos Proj. GB	Tempo Horas	Nós
Exemplo 1	900	3.984	0,413	14,6	1
Exemplo 2	1.000	4.119	0,850	15,1	1
Exemplo 3	1.148	4.653	1,900	17,1	1
Exemplo 4	1.350	5.482	2,010	20,1	1
Total	4.398	18.239	5,173	66,9	4
Média	1.100	4.560	1,293	17	1

Fonte: Elaborado pelo autor.

ESTUDO DE CASO 1:

Numa primeira situação foi adotada como critério a divisão de processos iguais ao número de máquinas, sendo assim, uma máquina, um processo, duas máquinas, dois processos, assim por diante.

Tabela 6 – Resultados obtidos no processamento dos dados.

Arquivos	Frame	Taman. dos Proj. GB	Processos
Exemplo 1	900	0,413	13.148
Exemplo 2	1.000	0,850	13.594
Exemplo 3	1.148	1,900	15.356
Exemplo 4	1.350	2,010	18.092
Total	4.398	5,173	60.190
Média	1.100	1,293	15.048

Fonte: Elaborado pelo autor.

Através dos dados apresentados acima é possível notar que ocorreu um aumento gradativo nas quantidades de processamentos, conforme mostra a 4º coluna, coletado do monitoramento do *DrQueue*, fazendo com que se exigisse cada vez mais do processamento do *cluster*.

Um segundo teste foi analisado dentre os processos, dos quatros exemplos a observar em quanto tempo e quanto que seria consumido de *cpu* e *memória ram* de todo o *cluster*, conforme apresentado na Tabela 7.

Tabela 7 – Resultados obtidos na eficiência do Cluster.

Arquivos	Tempo Horas	CPU	Memoria Cons. MB	Nucleo	Memoria Total MB	Nós
Exemplo 1	3,7	66%	4.383	6	8.192	3
Exemplo 2	3,8	68%	4.531	6	8.192	3
Exemplo 3	4,3	77%	5.119	6	8.192	3
Exemplo 4	5,0	90%	6.031	6	8.192	3
Total	16,7	301%	20.063	24	32.768	12
Média	4	75%	5.016	6	8.192	3

Fonte: Elaborado pelo autor.

Com este levantamento, como pode ser observado na Tabela 7, obteve-se um tempo aproximadamente de 16 horas de renderização, conforme mostra a 2º coluna, 5º linha, um desvio padrão de *cpu* de 75% de uso, conforme mostra a 3º coluna, na 6º linha, e por fim um consumo de *memória ram*, em desvio padrão de mais de 5 mil *MB*, como mostra na 4º coluna, 6º linha, já na 5º coluna obteve-se a quantidade de núcleos sendo utilizados, na 6º coluna o total de *memória ram* disponível no *cluster* e 7º coluna a quantidade de nós sendo executadas, em todas estas quatro renderizações.

Nos estudos de caso com o *cluster* foram feitas variações manuais dentro do extraídos do monitoramento do *DrQueue* na quantidade de processos com a finalidade de se investigar alguma vantagem em fazer o aumento do número de processos, de forma a explorar ao máximo todos nós do *cluster*, a fim de chegar em mais de 60 mil processos, de forma a tentar baixar as 16 horas de processamento, no entanto mesmo com os *mixer* de configurações não conseguimos obter uma melhora significativa, nos processamentos e tempos de processamento.

ESTUDO DE CASO 2

No estudo de caso 2 descreve os dados detalhados onde obteve-se com a renderização feita a partir de uma máquina com as seguintes configurações:

CPU – AMD 2,1 GHz

Memória cachê – 512 MB

Memória Ram – 1 BG 667 MHz

Disco rígido – 500 GB 7200 RPM Sata

O qual obteve-se os seguintes resultados de processamento de dados, conforme listado na Tabela 8.

Tabela 8 Resultados obtidos no processamento dos dados.

ComCluster

Arquivos	Frame	Processos	Taman. dos Proj. GB
Exemplo 1	900	13.148	0,413
Exemplo 2	1.000	13.594	0,850
Exemplo 3	1.148	15.356	1,900
Exemplo 4	1.350	18.092	2,010
Total	4.398	60.190	5,173
Média	1.100	15.048	1,293

Sem Cluster

Arquivos	Frame	Processos	Taman. dos Proj. GB
Exemplo 1	900	3.984	0,413
Exemplo 2	1.000	4.119	0,850
Exemplo 3	1.148	4.653	1,900
Exemplo 4	1.350	5.482	2,010
Total	4.398	18.239	5,173
Média	1.100	4.560	1,293

Fonte: Elaborado pelo autor.

Na tabela 8 na coluna arquivos tem-se a descrição de cada exemplo, já na coluna frame tem-se a quantidade de *frames* de cada exemplo, na coluna processo tem-se . a quantidade de processamento executados para cada exemplos apresentados e na coluna taman. dos proj. GB temos o quanto se refere cada exemplo em GB.

Como é possível verificar, uma queda de mais de um terço no processamento da média, conforme mostra a 3º coluna, 6º linha da tabela sem *cluster*, comparada com a 3º coluna, 6º linha da Tabela com *cluster*.

Em uma outra analise dentre os processos, dos quatros exemplos a observar em quanto tempo e quanto seria consumido de cpu e memoria ram, da máquina comum, conforme apresentado na Tabela 9.

Tabela: 9 Resultados obtidos na eficiência da maquina comum.

Arquivos	Tempo Horas	CPU	Memoria Cons. MB	Nucleo	Memoria Total MB	Nós
Exemplo 1	14,6	99%	1.003	1	1.024	1
Exemplo 2	15,1	99%	1.009	1	1.024	1
Exemplo 3	17,1	99%	1.010	1	1.024	1
Exemplo 4	20,1	99%	1.020	1	1.024	1
Total	66,9	396%	4.042	4	4.096	4
Média	17	99%	1.011	1	1.024	1

Fonte: Elaborado pelo autor.

Como pôde-se analisar nos resultados obtidos na Tabela 9, um grande consumo de *cpu* de 99% como mostrado na 3º coluna, 6º linha, como desvio padrão, e um outro dado relevante é com relação ao consumo de *memória ram*, onde obteve-se um consumo como desvio padrão

de 1.011 MB, como mostra na, 4º coluna, 6º linha, onde se leva em conta que tinha-se um total de 1.024 MB, como mostra na 6º coluna, 6º linha.

Vale apena ressaltar que em vários momentos deste teste observou-se alguns travamentos por parte do alto consumo de processamento sendo executados, tomando como consequência, mais de 67 horas para processar os mesmos quatros exemplos utilizados no *cluster*, como mostra na, 2º coluna, 5º linha.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi possível concretizar um estudo do ambiente de computação em *cluster* utilizando como sistema operacional *Ubuntu* Server 11 x86, *DrQueue* e o *Blender*, com a montagem de uma estrutura e configuração de um *cluster* utilizando esta tecnologia. Executando aplicações de renderização de imagem no *cluster*, pôde-se constatar que aplicações como estas que exigem grande poder de processamento podem tirar vantagens de uma estrutura de *cluster*, baseado em computadores pessoais.

Um dos pontos fundamentais que foram observados com o desenvolvimento deste trabalho, é a diferença notória de uma máquina comum, para um *cluster* com as mesmas máquinas comuns, pórem, trabalhando de forma paralelas em rede. Um outro ponto bastante notório é o grande trabalho de paralelismo do *software DrQueue*, ele não paraleliza a aplicação que será executada, identifica um nó mais livre na rede, e migra do contexto de *software* para outro nó, balanceando as aplicações de forma a ter como requisitos a compatibilidade com a migração de processos, para tirar proveito do *cluster*.

No capítulo de estudo da ferramenta de *renderização* chamada *Blender*, nota-se que este *software* de criação e renderização de imagem, conseguiu tirar proveito da estrutura de *cluster* montada, dentre todas as animações *renderizadas*, onde obteve grande resultado na sua distribuição dentro dos nós.

Neste teste, o foco foi visualizar o quanto se obtinha de processamento num *cluster* com três máquinas *renderizando*, partindo deste propósito, colhe-se os resultados, de forma a saber o quanto o *cluster* renderia para cada *frame* colocado para renderizar, onde obteve-se grandes resultados com um desvio padrão de cada 1.100 *frames* compilados, tínha-se um desvio padrão de 15.048 de processamento renderizado, com um desvio padrão de tempo de 4 horas de processamento, ou seja, tendo a cada hora *renderizada* 275 *frames*, sendo assim a cada hora *renderizando* 25% de todo o seu trabalho.

Além do estudo e testes realizados, foi executado uma comparação de quanto tempo de renderização um *cluster* efetivamente ganhou, ao invés de fazer em uma máquina comum. Obtendo os seguintes números, para os mesmos arquivos processados no *cluster* colocado em uma máquina comum, obteve-se uma perda de 50,16 horas a mais, enquanto todos estes arquivos no *cluster renderizava* em 16,7 horas em uma máquina comum, os mesmos arquivos demoravam para *renderizar* 66,88 horas.

Este fato ocorreu devido ao grande número de processamento em poucos minutos processados no claster , utilizando técnicas de paralelismo e balanceamento de carga entre os nós,

enquanto para demora todas estas mais de 66 hóras de renderização tivemos apenas uma só maquina processando, sem nenhuma técnicas de paralelismo e balanceamento de carga.

Como trabalho futuro pode–se pensar em uma maior quantidade de máquinas, de modo a ganhar maior poder computacional, em paralelo buscar um melhor tráfego de rede, de forma a expandir os 10/100 Mbps de tráfego, obtido neste projeto, visto que dentro do *cluster* como cita Pitanga (2004), um dos maiores problemas de performance é a rede, devido ao grande número de *delay* (colisões de pacotes).

REFERÊNCIAS

VIEIRA, E. Cluster: principais conceitos. **Tu Visse?**, [2011]. Disponivel em; < http://www.elenilsonvieira.com.br/entendendo-clusters/>.Acesso em: 07 abr. 2012.

ENTENDA os supercomputadores, potentes máquinas usadas na pesquisa científica. **Geek**, 2011 Disponivel em,<IT http://www.geek.com.br/posts/15978> Acesso em: 12/11/2011.

BARROSO, L. A.; DEAN, J.; HOLZLE, U. Web Search for a Planet: the Google Cluster Architecture. **IEEE Micro,** LOCAL DE PUBLICAÇÃO, v. 23, n. 2, p. 22-28, 2003.

BODEN, N. J.et al. Myrinet: a Gigabit-per-Second Local Area Network. **IEEE Micro,** (local de publicação), v. 15, n. 1, p. 29-36, 1995. Disponivel em; < http://www.infowester.com/cluster.php>.Acesso em: 24 jun. 2011.

Beowulf.org, c2009. Disponível em: <http://www.beowulf.org/>. Acesso em: 12 maio 2011.

BUYYA, R. **High Performance Cluster Computing:** architectures and systems: volume 1. New Jersey : Prentice-Hall, 2004.

CELEBIOGLU, O.; RAJAGOPALAN, R.; ALI, R. **Exploring InfiniBand as an** HPC Cluster Interconnect. In Dell Power Solutions. Dell, 2004. Acesso em 28 Nov 2012 Disponível em <http://www.dell.com/powersolutions>.

Clarke, L.; Glendinning, I.; Hempel, R. (1994) The MPI message passing interface standard. Knoxville: University of Tenessee. Relatório técnico.

O QUE É processador dual core. **Windows x Linux**, c2012. Disponivel em.< http://winxlinux.com/o-que-e-processador-dual-core/>. Acesso em: 3 jun. 2011.

Clarke, L.; Glendinning, I.; Hempel, R. (1994) The MPI message passing interface standard. Knoxville: University of Tenessee. Relatório técnico.

D.A. Patterson; J.L. Hennessy, Computer Organization and Design, 4th Ed, Morgan Kaufmann, 2009

D.F. Rogers, Procedural Elements for Computer Graphics
Ferreto, T.; De R., César A. F. (2002) RVision: uma ferramenta aberta e configurável

Monitoração de clusters. II Escola Regional de Alto Desempenho (ERAD 2002).

Dignow 2011 http://www.dignow.org/post/com-o-fujitsu-k-jap%C3%A3o-tem-o-supercomputador-mais-potente-do-planeta-2365240-60186.html Acessado em 03/12/2011

FOLEY 1996 FOLEY, James D. et al. Computer Graphics: Principles and Pratices, Second Edition in C. Massachusetts: Addison-wesley, 1996.

FERW 2003 Ferwerda, James A. Three Varities of Realism in Computer Graphics. Proceedings SPIE Human Vision and Electronic Imaging, [S.I.] ,p. 290-297, 2003.

Ferreto, T.; De R., César A. F. (2002) RVision**: uma ferramenta aberta e configurável para monitoração de clusters**. II Escola Regional de Alto Desempenho (ERAD 2002).

FILHO, N. A. P. **Linux, Clusters e Alta Disponibilidade**. 2002. 123f. Dissertação (Mestrado em) - Universidade de São Paulo, São Paulo, 2002. Disponível em: < http://www.brlink.com.br/s/linux/cluster-linux>.

Gonzalez (2003) Gonzalez, R. F.; Woods, R. E. **Digital Image Processing**. Addison-Wesley, p 716. 2003.

Hennessy, J. L.; Patterson, D. A (2003). **Computer Architecture**: A Quantitative Approach. Elsevier Science.

HAGE86 HAGEN, M. Varieties of Realism. Cambridge: Cambridge University Press: 1986.

Joomla . Blender Brasil, 2011 Disponivel em <http://www.blender.com.br/index.php> Acesso em: 07 jun. 2011.

Miers, L. S. (2003) Implementação do Método dos Elementos de Contorno para Elasticidade Tridimensional em Ambiente Paralelo de Memória Distribuída. Dissertação (Mestrado em Ciência da Computação). Universidade Federal do Rio de Janeiro p59.

MACHADO, M., Hofmam, M. Sistemas distribuídos MPI, ed. 3º Editora. São Paulo 2005.

(Multicor) Copyright (2000) Disponivel em: <http://www.tecmundo.com.br/1627- multi-core-realidade-e-tendencia.htm> acesso em 28/10/2011

MACHADO, M., Hofmam, M. Sistemas distribuídos MPI, 2005.
Michael J. Quinn Parallel Programming in C with MPI and OpenMP
New York: McGraw-Hill

Miers, L. S. (2003) Implementação do Método dos Elementos de Contorno para Elasticidade Tridimensional em Ambiente Paralelo de Memória Distribuída. Dissertação de Mestrado. Universidade Federal do Rio de Janeiro.

N. M. Thalmann, D. Thalmann, Computer Animation - Theory and Practice, Springer-Verlag, 1992.

OpenMosix (2005) openMosix, an **Open Source Linux Cluster Project**. Disponível online em http://openmosix.sourceforge.net. Acesso em Janeiro de 2011.

[PLOE] Atze van der Ploeg. Interactive Ray Tracing, the replacement of rasterization. Tese B.Sc., Vrije Universiteit, Dezembro 2006.

PITANGA, **Marcos. Construindo supercomputadores com Linux**. 2. ed. Rio de Janeiro: Editora Brasport, 2004.

PITANGA, M. Computação em Cluster: o estado arte da computação. Rio de Janeiro: Brasport, 2003.

Perry S. Marshall e John S. Rinaldi Industrial Ethernet 2nd Edition 2005

Supercomputador-do-Mundo-hoje Disponível online (http://www.infoblogs.com.br/view.action?contentId=38991&O-Maior-Supercomputador-do-Mundo-hoje.html) Acesso em Março de 2011.

Technology 2011 Disponivem em< http://www1.us.dell.com/content/>, Multi-Core Technology Brief, acesso em 22/06/2011

YOKOKURA, Alex Yuichi, Estudo de Viabilidade da Implantação de Técnicas de Cluster ao Projeto Servidor de Estações de Trabalho 2007 (SET).