REVISTA DO CCEI
ISSN Impresso: 1415-2061
ISSN Online: 2356-6635
Centro de Ciências
Economia e Informática
Revista Técnico-Científica
URCAMP
EDIURCAMP

ARQUITETURA DE SISTEMAS PARA CLUSTERS E GRADES COMPUTACIONAIS: UMA SOLUÇÃO INDEPENDENTE DE FABRICANTE BASEADA EM *CLUSTERS BEOWULF*

Bruno Moura Paz de Moura[1]

Ramon Rosa Maia Vieira Junior [2]

RESUMO

O presente trabalho tem como objetivo identificar os problemas e as tecnologias que podem ser utilizadas na sua resolução, onde a necessidade de alto poder de processamento e alta disponibilidade de serviços existirem. Sobre uma pesquisa exploratória e descritiva são apresentados conceitos, técnicas, metodologias, *softwares*, *hardware* que podem ser utilizados na implementação de uma infraestrutura de ambientes de computação de alto desempenho e alta disponibilidade. Com a crescente demanda de poder computacional, empresas públicas e privadas podem vir a utilizar estes ambientes a custo razoável no que se trata de *hardware* e *software*. Os *clusters Beowulf* acabam por ser uma alternativa viável onde pode se tirar proveito. Este trabalho também aborda as arquiteturas computacionais existentes em relação aos supercomputadores do tipo *Beowulf* destacando as vantagens e desvantagens destes ambientes.

Palavras-chave: *Clusters Beowulf*, MPI, PVM.

SYSTEMS ARCHITECTURE FOR CLUSTERS AND COMPUTATIONAL GRIDS: A SOLUTION BASED MANUFACTURER OF INDEPENDENT IN BEOWULF CLUSTERS

ABSTRACT

This paper aims identify the problems and technologies that can be used in the resolution, where the need for high processing power and high availability services exist. On an exploratory and descriptive research are presented concepts, techniques, methodologies, software, hardware that can be used to implement an infrastructure of high performance computing environments and high availability. With the growing demand for computing power, public and private companies are likely to use these environments at a reasonable cost when it comes to hardware and software. Beowulf clusters turn out to be a viable alternative where you can take advantage. This work also deals with the existing computer architectures in relation to the type Beowulf supercomputers highlighting the advantages and disadvantages of these environments.

Keywords**:** *Clusters Beowulf*, MPI, PVM.

[1]Analista de Tecnologia da Informação, Diretoria de Tecnologia da Informação e Comunicação – DTIC, Universidade Federal do Pampa - UNIPAMPA, AV General Osório, 900, Bagé/RS, Mestrando em Computação, Universidade Federal de Pelotas - UFPEL, brunomoura@unipampa.edu.br.

[2]Prof. Mestre, Faculdade Vitoriana de Tecnologia - FAVI, Brasil, Doutorando em Informática na Educação, Universidade Federal do Rio Grande do Sul - UFRGS, ramonwaia@gmail.com.

1 INTRODUÇÃO

Na atualidade a evolução na fabricação de *chips* e o crescimento do poder de processamento nos computadores pessoais, as redes de computadores de alta velocidade LAN (*Local Area Network*) e o uso de padrões como: *Ethernet*, *Fast Ethernet, Gigabit Ethernet*, estabelece como boas alternativas para a transferência de grande quantidade de dados entre computadores em alta velocidade.

O aumento constante na demanda de poder computacional é em torno de processadores mais rápidos para cada vez mais diminuir o tempo de resposta de processamento de aplicações que necessitem de grandes computações. A computação de alto desempenho, atualmente é utilizada para resolver problemas científicos, multimídia e que envolva grande quantidade de dados, tais como: engenharia fotônica; sensoriamento remoto; engenharia genética; sísmica; meteorologia; pesquisas militares e segurança de reatores nucleares, dentre outras (PITANGA, p. 2008; TANNENBAUM, p. 2001).

Na computação de alto desempenho, utilizada para programação científica, multimídia, e gerenciamento de grandes volumes de dados, uma solução passa por máquinas com múltiplos processadores ou ainda *clusters* proprietário fornecidos por grandes empresas.

A utilização de sistemas distribuídos de alto desempenho e alta disponibilidade empregando tecnologias de *software* livre estimulam os desafios na proposição de soluções aos problemas na área, e dentre eles destacam-se: (i) Os ambientes distribuídos de *clusters* computacionais do tipo *Beowulf* estão aptos a resolver problemas em quais contextos computacionais? (ii) Quais são as metodologias, tecnologias e paradigmas no contexto dos *clusters Beowulf*?

O presente trabalho tem como objetivo central identificar o universo de problemas e as tecnologias que podem ser utilizadas na proposta para soluções em ambientes distribuídos de *clusters* computacionais do tipo *Beowulf*, visando o uso de alto poder de processamento e alta disponibilidade.

2 REFERENCIAL TEÓRICO

2.1 INTRODUÇÃO AOS SUPERCOMPUTADORES

A primeira geração de computadores estabeleceu os conceitos básicos de organização dos computadores eletrônicos e na década de 50 apareceu o modelo computacional que se tornaria a base de todo o desenvolvimento subsequente, denominado “modelo de *Von Neumann*” (PITANGA 2008, p. 1).

O modelo de *Von Neumann* é composto de basicamente por um processador, memória, dispositivos de entrada e saída de dados ou periféricos. O processador executa as instruções sequencialmente, de acordo com a ordem estabelecida por uma unidade de controle.

O avanço dos dispositivos foi acontecendo de forma gradual até os anos 80, com o surgimento dos circuitos integrados de larga escala de integração, os *chips* VLSI (*Very Large Scale Integration*). O aumento da velocidade do *hardware*, por mais rápido que chegue, sempre esbarra no limite tecnológico de cada época (PITANGA 2008, p. 2).

Desta forma, pode ser observado que a utilização do paralelismo é essencial, a paralelização proporcionou grandes mudanças nas arquiteturas, o Quadro 1, apresenta a evolução das arquiteturas.

Quadro 1: Características na evolução das arquiteturas.

• O aparecimento do conceito de palavra do inglês (*Word*) leva à manipulação de vários *bits* em paralelo, ao invés de unidade básica *byte*;
• Os processadores de entrada e saída possibilitaram o paralelismo entre os periféricos e o processador principal, tornando as tarefas de E/S mais independentes;
• Como as memórias também eram lentas e criavam um gargalo, surgiram técnicas como *cache* e entrelaçamento "*interleave*" para acelerar o desempenho do sistema (que permitiriam a obtenção de mais de uma palavra na memória em paralelo);
• O aparecimento dos *pipelines* ou vias internas do microprocessador, que paralelizam a execução de instruções e operações aritméticas de uma forma implícita;
• No período de 1953 a 1970 o paralelismo existente era transparente ao usuário, influenciando apenas na melhoria do desempenho da máquina, sendo denominado de paralelismo de baixo nível;
• No ano de 1975 nasce o Illiac IV, com 64 processadores, a primeira forma em que o paralelismo diferencia do paralelismo de baixo nível, Foi o computador mais rápido até o aparecimento do Cray;
• O surgimento da filosofia VLSI para projetos de microcomputadores oferece maior facilidade e menor custo no desenvolvimento e microprocessadores cada vez mais complexos e menores (computadores pessoais);
• A partir dessa época surgiram dois caminhos distintos: arquiteturas paralelas e sistemas distribuídos ou baseados em redes e as máquinas vetoriais.

Fonte: Pitanga (2008).

Segundo (COLVERO, p. 2005), algumas áreas de conhecimento, (astronomia, meteorologia e genética) requerem, para os problemas estudados, de muitos recursos computacionais com alto desempenho para suprir cálculos complexos e repetitivos. Os ambientes de *clusters* e grades computacionais que hoje podem ser chamados de *cloud computing* (computação em nuvem) surgiram com o objetivo de execução de aplicações em menor tempo e na grande quantidade de dados da aplicação em comparação com a execução em máquinas *desktops* normais.

Neste contexto três categorias são abordadas: (i) aumento na velocidade dos processadores, (ii) algoritmos mais otimizados, e (iii) ambientes de computação paralela e distribuída (COLVERO, p. 2003). Em ambos os paradigmas, estes esforços são somados garantindo a baixa latência de

comunicação (na ordem de centenas de μs) e a elevada taxa de transmissão (a partir de dezenas de Mbps até, atualmente, dezenas de Gbps) (BUYYA, p. 1999).

2.2 UNIDADES DE MEDIDA PARA MAQUINAS PARALELAS

A unidade de medida utilizada para medir o desempenho de máquinas paralelas conforme (DE ROSE, p. 2003; NAVAUX, p. 2003), é estabelecida pela quantidade de instruções empregadas na execução de uma aplicação para processar em uma determinada máquina. Para medir essas velocidades das Máquinas Paralelas, foi então estabelecida à unidade: o FLOPS (*Floating Point Operations Per Second*).

Como o MIPS (*Millions of Instructions Per Second*) representa a unidade de milhão de instruções executadas num segundo por uma máquina, o MEGAFLOPS, ou MFLOPS, mede o número de milhões de operações em ponto flutuante executadas por segundo. Até então há inexistência de uma convenção fixa que estabeleça quantos MIPS compõe um MFLOPS. Considera-se, em geral, que uma operação de ponto flutuante seja executada por 2 a 7 instruções habitualmente, mas dependendo da máquina pode chegar até 10, ou seja, depende do tipo de computador.

Por outro lado, sabe-se que essa medida não é precisa, pois o poder das instruções varia de máquina para máquina. Uma instrução de um determinado computador pode resultar numa operação muito mais poderosa que precisará de um conjunto maior de instruções para obter o mesmo resultado num outro computador. Resulta desse fato que a forma mais correta de medir o desempenho entre máquinas seja empregar programas padrão que serão rodados nas máquinas objeto da comparação, essa técnica é conhecida por *Benchmarking*.

Segundo (PITANGA, p. 2008), os *bechmarking* são padrões de referência para avaliação de sistemas computacionais e são compostos por vários programas complexos que utilizam diferentes características do ambiente, como entrada/saída, operações em números inteiros e de ponto flutuante, entre outros.

Para (PITANGA, p. 2008), qualquer *benchmarking* deve poder medir os tempos de execução de vários programas de teste, e os resultados devem ser normalizados pelo volume de uso de cada aplicação colocada em teste. O Quadro 2 apresenta alguns *benchmarkings* em nível de *software* e *hardware* que são alternativas a serem utilizadas na avaliação de desempenho em ambientes de clusters *Beowulf*.

Quadro 2: Exemplos de ferramentas de *benchmarking*.

Benchmarking	Nível	
	Software	***Hardware***
AIMS	X	
Alpes	X	
TotalView	X	
IDTrace	X	
Pablo	X	
Paradyn	X	
Linpack		X
Lapack		X
SPEC		X

Fonte: Pitanga (2008).

2.3 CLASSIFICAÇÃO DE FLYNN

Flynn (FLYNN, p. 1972), ainda no começo da década de 70, apresentou uma classificação das arquiteturas de computadores de acordo com o fluxo de instruções e o fluxo de dados existentes em cada sistema.

Para (BORGES, p. 2012), o motivo da enorme diversidade de arquiteturas de computadores, várias taxonomias já foram propostas, com intuito de se uniformizarem, de forma coerente, características dos diferentes sistemas computacionais.

A classificação dos ambientes de *hardware* na área de arquiteturas de computadores é conhecida como taxonomia de *Flynn*. Esta classificação foi definida há mais de trinta anos e, é aceita até os dias atuais devido a sua enorme abrangência, é considerado o número de instruções executadas em paralelo por conjunto de dados para as quais as instruções são submetidas. A Figura 1 apresenta as quatro categorias que classificam as arquiteturas existentes segundo a taxonomia de *Flynn,* e para melhor descrevê-las, o Quadro 3 apresenta um detalhamento das classes que são formadoras dessa taxonomia.

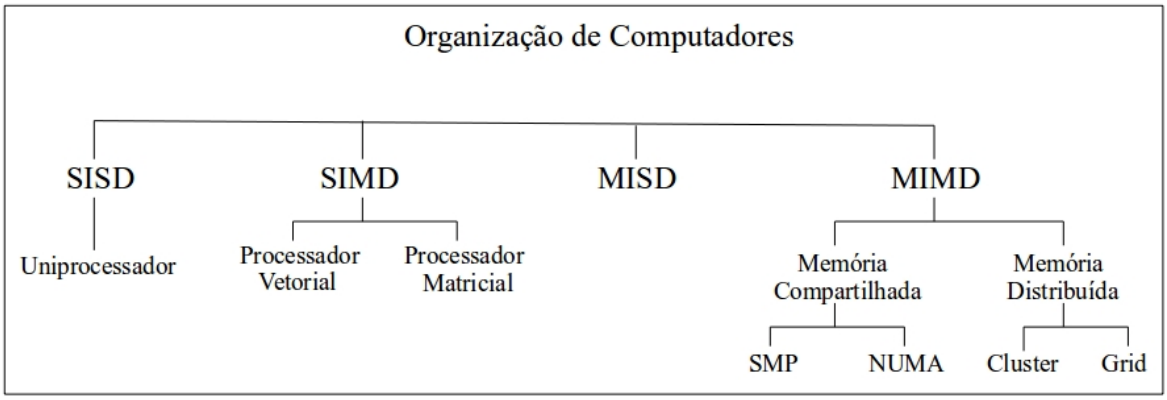

Figura 1: Extensão da classificação de *Flynn*.
Fonte: Borges (2012).

Quadro 3: Descrição da taxonomia da *Flynn*.

Classe	Descrição
***Single Instruction Single Data* (SISD):**	É a identificação mais simples onde o equipamento é considerado sequencial, pois só é executada uma instrução por vez para cada dado enviado. São as máquinas *Von Neumann* tradicionais, sem qualquer paralelismo. Ex.: máquinas mono processadas (Mac, PC, Worktation, Sun);
***Multiple Instruction Single Data (MISD)*:**	São máquinas que executam várias instruções ao mesmo tempo sobre um único dado. Este tipo de classificação não possui representante e até mesmo Flynn duvidou que algum dia isso pudesse existir. Mas poderíamos criar por esta teoria uma máquina com vários processadores tentando quebrar um código de criptografia;
***Single Instruction Multiple Data (SIMD)*:**	É o equivalente ao paralelismo de dados, onde uma simples instrução é executada paralelamente utilizando vários dados de forma síncrona, em que se executa um único programa ao mesmo tempo. Neste contexto também se encaixa, pela característica, a tecnologia MMX (*MultMedia eXtension*) existente dentro dos processadores atuais e nos computadores vetoriais como Cray 1, CM-2;
***Multiple Instruction Multiple Data (MIMD)*:**	Refere-se ao modelo de execução paralela, a qual cada processador está essencialmente agindo independentemente, havendo, portanto, múltiplos fluxos de instruções e múltiplos dados como se fosse um conjunto de máquinas SISD, em que cada processador é capaz de executar um programa diferente. Ex.: servidores com múltiplos processadores, sistemas MPP (Processadores Massivamente Paralelos) e os *clusters* de computadores.

Fonte: Pitanga (2008)

2.4 CLUSTERS DE COMPUTADORES

Clusters são grupos de computadores configurados para trabalhar em conjunto com aplicações específicas. O modo como são configurados dá a impressão da serem como um único computador (FERREIRA, p. 2008).

A estrutura e agrupamento de computadores definida como:

> Estrutura de agrupamento de computadores ou *clusters* apresenta vantagens competitivas em relação aos ambientes multiprocessados de memória compartilhada (computadores com diversos processadores em uma placa-mãe), permitindo que o acréscimo de computadores torne o sistema mais rápido, possuindo componentes de fácil disponibilidade e manutenção (PITANGA, 2008 p.10).

Os *clusters* podem ser conceitualizados nas categorias de *High Availabitity (HA)* ou Alta Disponibilidade, e *clusters* de *High-Performance (HP*) ou Alto Desempenho, (FERREIRA 2008, p.683).

2.4.1 Clusters High-Availability - HA

Alta disponibilidade é uma técnica que consiste na configuração de dois ou mais computadores para que passem a trabalhar em conjunto. Dessa forma cada computador monitora os demais e quando ocorrer falhas assume o serviço que fica indisponível. Para este subgrupo de computadores atribui-se o nome de *clusters* de alta disponibilidade (FERREIRA 2008, p.683). Também se destacam as seguintes classes de disponibilidade, expostas no Quadro 4, apresentando-se como alternativa para organização dos *clusters Beowulf*.

Quadro 4: Classes de Disponibilidade.

Classe de Disponibilidade	**Descrição**
Disponibilidade convencional:	É aquela encontrada em computadores comuns disponível no mercado, sem nenhum recurso extra de *software* ou *hardware* que oculte as eventuais falhas. Para esta classe de computadores, a disponibilidade é em torno de 99% a 99,9%, Isso significa que em 1 ano de operação o computador pode ficar indisponível por um período de 9 horas a 4 dias;
Alta disponibilidade:	É encontrada em computadores mais sofisticados com recursos de detecção, recuperação e ocultamento de falhas. Os computadores dessa classe possuem disponibilidade de 99,99% a 99,9996%. Isso significa que em 1 ano de operação o computador pode ficar indisponível por um período de pouco mais de 5 minutos;
Disponibilidade contínua:	É aquela encontrada em computadores bem mais sofisticados com recurso de detecção, recuperação e ocultamento de falhas onde se obtém disponibilidade cada vez mais próxima de 100%, reduzindo o tempo de inatividade do computador, de forma que esse venha a ser insignificante ou até mesmo inexistente.

Fonte: Ferreira (2008).

Uma solução possível para alta disponibilidade é baseada em quatro sistemas básicos: (i) sistema de arquivos robusto como (*ext3/reiserfs*), (ii) replicação/sincronização de discos executada por *software* um exemplo o DRBD (*Data Replicator Block Device*), (iii) monitoramento de nós executada por *software* um exemplo é o *Heartbeat,* e (iv) monitoramento de serviços exemplo o *software* Mon, estas ferramentas podem ser executadas individualmente ou em conjunto (FERREIRA 2008, p. 684).

O DRBD é um módulo de *kernel* e *scripts* associados que fornecem um dispositivo de bloco projetado para construir *clusters* de alta disponibilidade, tudo feito através de espelhamento de bloco via rede dedicada, pode se considerar com RAID (*Redundant Array of Independent Disks*) via rede (FERREIRA 2008, p. 686).

O DRBD encarrega-se de responsabilizar-se dos dados escritos no disco rígido local e os envia ao outro *host*, no outro *host* escreve os dados no disco. Outros componentes necessários são os serviços de relacionamento entre as máquinas do *cluster* o qual pode ser o *Heartbeat*, e alguma aplicação que trabalhe no topo do dispositivo de bloco como um *filesystem* e *fsck*, um *journaling FS,* ou um banco de dados com capacidade de restauração (FERREIRA 2008, p. 686).

Heartbeat do inglês batimento cardíaco, esse termo é usado para definir pulsos enviados entre dois computadores que indicam que estão vivos, ou seja, estão funcionando e disponível para executar tarefas. *Heartbeat* trabalha enviando um pulso entre dois computadores através da porta serial, uma placa de rede ou ambas (FERREIRA 2008, p. 688).

Caso o pulso venha a falhar, o computador secundário irá assumir que o computador primário falhou e tomar os serviços que estão rodando no computador primário. O *Heartbeat* define um endereço *Internet Protocol* (IP) para o *cluster*, que deve ser diferente dos endereços IP do servidor primário e secundário que será o endereço IP procurado pelos clientes. O computador que estiver vivo primário ou secundário assumira o endereço do *cluster* (FERREIRA 2008, p. 688).

2.4.2 *Clusters High-Performance Beowulf*

Beowulf é uma categoria de multicomputador que pode ser utilizada para computações paralelas, normalmente é composto por um nó servidor e um ou mais nós clientes conectados via rede. É construído com *hardware* comum, como qualquer computador capaz de operar com Linux, placa de rede padrão *Ethernet* e *switches*. Para esta classe não é utilizado nenhum *hardware* feito sobre encomenda e é facilmente reproduzível (FERREIRA 2008, p. 691).

Beowulf usa *software* comum como o sistema operacional Linux, *Paralell Virtual Machine* (*PVM*), e *Message Passing Interface (MPI)*. O nó servidor é encarregado de controlar todo o grupo de computadores e serve arquivos para os nós clientes, também é a porta de saída para o mundo exterior (FERREIRA 2008, p. 691).

Grandes máquinas *Beowulf* podem ter mais de um nó servidor e possivelmente outros nós dedicados para tarefas específicas como console e estações de monitoramento, na maioria dos casos o nós clientes são computadores simples.

Os nós clientes são configurados e controlados pelo nó servidor e fazem somente o que são ordenados a fazer. Em uma configuração *Beowulf* em que os nós clientes não têm disco rígido, os nós clientes não sabem os seus endereços IP nem os seus nomes, até que o servidor diga a eles quem eles são.

A principal diferença entre o *Beowulf* e um COW (*Cluster Of Workstation*) é que se comporta como se fosse um único computador e não como um grupo de muitas estações de trabalho. Na maioria dos casos, os nós clientes não têm teclados, monitores nem *mouses* e são acessados via *login* remoto (FERREIRA 2008, p. 691).

Podem ser caracterizados como um conjunto de placa-mãe (composto por processador e memória), placa de rede, placa de vídeo, em algumas configurações mais disco rígido.

Um *Beowulf* não é um pacote de *software* especial, é uma tecnologia de agrupar computadores utilizando o sistema operacional Linux para formar um supercomputador paralelo virtual.

Embora existam vários *softwares* como PVM, MPI dentre outras ferramentas que torna o computador *Beowulf* mais rápido e fácil de ser configurado, qualquer pessoa pode construir um computador *Beowulf* com distribuição Linux sem qualquer *software* adicional (FERREIRA 2008, p. 692).

Pode se dizer que dois computadores Linux conectados à rede, configurados para exportar o diretório */home* ou */usr/local* ou equivalente via NFS (*Network File System*) e executar *shell* remoto *rsh* entre ambos já caracteriza um computador *Beowulf*. A Figura 2 apresenta dois esquemas da arquitetura de *clusters* do tipo *Beowulf*, (a) um *cluster* composto por um nó controlador (*Front-End*), interconectado via *switch,* com oito nós escravos (*slave*); (b) um *cluster* constituído de um nó controlador, interconectado via *switch,* com cinco nós escravos.

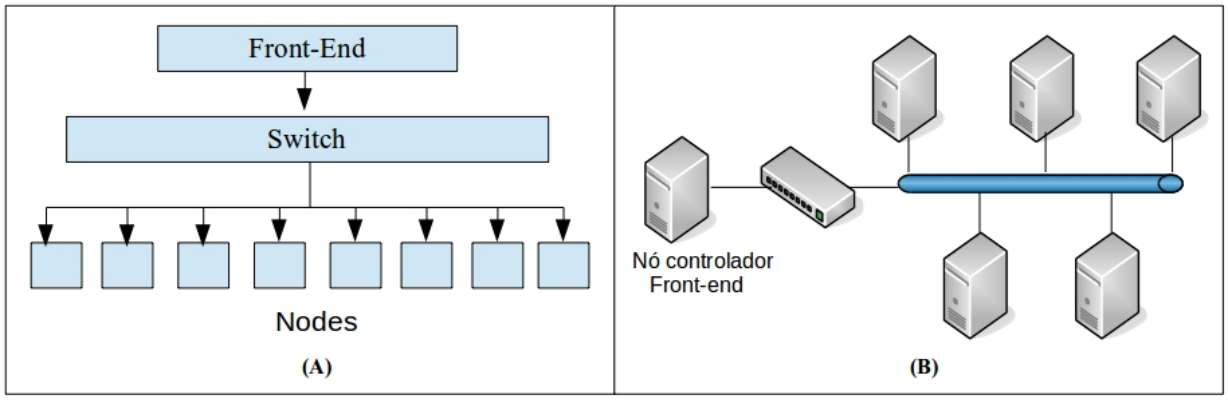

Figura 2: (a) Esquema, (b) Arquitetura de *clusters Beowulf* com um nó controlador.
Fonte: Bacellar (2009).

2.5 REDES PARA CLUSTERS

Em sistemas de agregados de computadores, é extrema a importância às tecnologias de redes de computadores. A conectividade é definida em como os elementos de processamento (CPU e memória) de um sistema de alto desempenho serão interligados. Neste contexto a topologia da rede é vista como um grafo onde os *switches* são os vértices e os *links* são as arestas (PITANGA 2008, p. 85).

O desempenho de uma rede é geralmente medido em termos de latência e largura de banda. A latência é o custo do tempo para enviar um dado de um computador até outro, incluindo o tempo para o *software* construir a mensagem e transferir os *bits*. A largura de banda é o número de *bits* por segundo que podem ser transmitidos pelo *hardware* de interconexão de rede (PITANGA 2008, p. 88).

O ideal para o bom desempenho de uma aplicação é ter uma rede com baixa latência e grande largura de banda. É necessária a utilização de protocolos de comunicação eficientes, que minimizem a sobre carga do *software* de comunicação e *hardware* mais rápido.

São protocolos que podem ser citados e que evitam a intervenção do sistema operacional e ao mesmo tempo fornecem um serviço de mensagens em nível de usuário através de redes de alta velocidade: *Active Messages, Fast Messages*, VMCC, U-net e BIP (PITANGA 2008, p. 88). No Quadro 5, são apresentados os critérios que podem ser utilizados para avaliação das diferentes redes de interconexão utilizadas nos ambientes dos *clusters Beowulf*, que são eles: (i) escalabilidade, (ii) desempenho, (iii) custo, (iv) confiabilidade, (v) funcionalidade.

Quadro 5: Avaliação das diferenças entre redes de interconexão.

Escalabilidade:	Capacidade de adaptação às necessidades do usuário. No caso das redes de interconexão, refere-se à capacidade da rede de interligar componentes adicionais, por exemplo, caso o usuário necessite de mais desempenho, mantendo as características originais;
Desempenho:	Indica a capacidade e a velocidade da transferência dos dados na rede. É influenciado por vários fatores como: desempenho físico das ligações utilizadas na rede, distâncias a serem percorridas e grau de paralelismo na transferência (quantos nós podem transferir dados simultâneos). Os principais indicadores de desempenho em redes de interconexão são a latência e a vazão. A latência indica o tempo que uma unidade de dados demora para ser transferida, e a vazão, quantas unidades de dados foram transferidas por unidade de tempo. Um carro com capacidade para 5 passageiros que leve 10 minutos para chegar a seu destino tem latência de 10 minutos e uma vazão de 30 passageiros/hora 5 passageiros * 6 vezes por hora). Outro fator a ser considerado é se as ligações são unidirecionais ou bidirecionais, podendo o último caso ser capaz de transferir dados em ambas às direções simultaneamente (*full-duplex*) ou não (*half duplex*).
Custo:	No caso das redes de interconexão, o custo cresce proporcionalmente em função do número de ligações e de sua capacidade e transferência (vazão e latência).
Confiabilidade:	A existência de caminhos alternativos redundantes entre componentes aumenta a confiabilidade da rede de interconexão em caso de falhas;
Funcionalidade:	Juntamente com a função de transferência de dados propriamente dita, as redes de interconexão podem eventualmente implementar outros serviços, como por exemplo, armazenamento temporário dos dados transmitidos (quem foi enviado primeiro chega primeiro), ou roteamento automático implementado em *hardware*.

Fonte: De Rose (2003).

A Myrinet é um padrão público e aberto, publicado e registrado na ANSI (ANSI/VITA 26-1998). Esta tecnologia foi desenvolvida para prover alto desempenho, comunicação eficiente de rede e comutação com uma razoável relação custo eficiência (MYRICOM, 2014, np). A tecnologia tem como objetivo a formação de c*lusters* de estações de trabalho, PCs e servidores. Para este objetivo uma configuração Myrinet dispõe de *switches* e placas de redes especiais para interligação do ambiente de rede. Para (PITANGA 2008, p. 85, 86), a divisão das redes define-se em dois modelos. No Quadro 6 são expostos os modelos de tipos de rede: (i) estática, (ii) dinâmica, destacando-se como categorias a serem usadas para classificação.

Quadro 6: Tipo de rede.

Tipo	Descrição
Rede estática:	Estas redes possuem conexões permanentes entre os elementos de processamento, ou seja, as ligações são diretas entre dois pontos. Ex.: (Rede em barra, anel, malha, completa, árvore, estrela, hipercubo, fat *tree*, dentre outras);
Rede dinâmica:	Neste tipo de rede não existe ligações diretas entre os elementos computacionais e, portanto, variam em relação ao tempo. Ex.: (Redes ômega, *crossbar*.)

Fonte: Pitanga (2008).

Se os componentes de máquina (processadores, memórias) estão interligados através de ligações fixas, de forma que, entre dois componentes, exista uma ligação direta dedicada, a rede de interconexão é denominada estática (ponto-a-ponto). Redes estáticas são utilizadas, na maioria dos casos, em multicomputadores. Nesse caso, a topologia (estrutura de interligação) determina as características da rede.

Máquinas paralelas possuem quase sempre estruturas regulares com ligações homogêneas, enquanto sistemas distribuídos, pelo motivo da localização geográfica e de sua integração gradual, possuem estruturas irregulares com ligações heterogêneas (ligações heterogêneas entre redes locais) (DE ROSE 2003, p. 76).

Na interconexão de componentes com redes dinâmicas, não existe uma topologia fixa que defina o padrão de comunicação da rede. Quando uma conexão entre dois pontos faz-se necessária, a rede de interconexão adapta-se dinamicamente para permitir a transferência dos dados (DE ROSE 2003, p. 82).

Para isso os produtos (MYRICOM, 2014, np) *Ethernet* 10 *Gigabit* para obtenção extrema de desempenho através da combinação de seus adaptadores de rede (*Network Interface Cards - NIC*) e *software* especializado são uma alternativa para este contexto aplicação.

2.6 PROGRAMAÇÃO PARALELA

De acordo com (CAVALHEIRO, p. 2004), vários modelos são propostos nas diversas áreas das ciências para descrever um item da realidade, por esta razão, o modelo permiti uma visão sistemática. Para (PITANGA, p. 2008), implementar programas para sistemas distribuídos e paralelos é muito parecido como escrever programas sequenciais, é claro que com algumas diferenças bem significativas. No Quadro 7 são apresentadas as diferenças significativas do paradigma da programação paralela em relação ao paradigma programação sequencial, destacando-se: (i) a organização, (ii) o acesso simultâneo, e (iii) a distribuição dos dados e das aplicações implementadas para estes ambientes de *clusters Beowulf*.

Quadro 7**:** Diferenças significativas da programação paralela em comparação a programação sequencial.

• A memória é distribuída: os dados da aplicação são distribuídos. Portanto, dados precisam ser subdivididos;
• As partições de dados podem ser acessadas simultaneamente por diferentes partes da aplicação, com isso os acessos aos dados devem estar sincronizados para evitar condições críticas;
• A distribuição dos dados pode causar degradação no desempenho devido às trocas de mensagens, então os programas paralelos devem ser escritos cuidadosamente para a obtenção de desempenho aceitável;
• Como consequência da distribuição, a própria aplicação deve ser subdivida e distribuída em *nodos* do *cluster,* o programador deve então aprender a utilizar técnicas de construções de algoritmos paralelos;
• A divisão do programa em partes cooperantes pode introduzir um baixo desempenho devido a ma atribuição destas partes aos processadores, por isso as partes dos programas devem ser cuidadosamente alocada aos processadores na etapa de escalonamento, a fim de se criar um balanceamento adequando das tarefas no *cluster*;
• As distribuições dos dados e das funções tornam o programa de aplicações mais complexo. O processo de programação precisa incorporar etapas de desenvolvimento adicionais para localizar e remover erros de sincronização.

Fonte: Pitanga (2008).

2.7 NÍVEIS DE GRANULOSIDADE DE CONCORRÊNCIA

O termo granulosidade definido por (CAVALHEIRO, 2004, p. 4):

> A concorrência em um programa em execução pode ser encontrada em diferentes níveis de granulosidade. O termo granulosidade diz respeito à razão entre o tempo necessário ao cálculo de uma determinada operação e os custos envolvidos nas trocas de dados entre esta operação e as demais operações de programa em execução. Em linhas gerais, a granulosidade de um programa pode ser relacionada ao tamanho médio das operações envolvidas neste programa. Assim um programa pode ser classificado em função de sua granulosidade: grossa, média, fina, e todas as variantes possíveis (muito - fina, muito – grossa...).

A Tabela 1 apresenta diferentes níveis de granulosidade correlacionando com suas respectivas classes que são exploradas em arquiteturas de computadores. A granulosidade pode afetar o desempenho do controle de concorrência na extração de paralelismo em estruturas de controle de programas paralelos. Nesta tabela encontram-se identificadas, para cada nível, arquiteturas típicas para o suporte a execução a cada classe de granulosidade.

Tabela 1: Níveis de granulosidade oferecidos por diferentes arquiteturas de computadores.

Níveis de granulosidade em diferentes arquiteturas de computadores		
Granulosidade	**Classe**	**Ocorrência**
Muito fina	Intra – instrução	Processadores VLIW[3] e Super - escalares
Fina	Entre instruções	Processadores vetoriais
Fina/Média	Blocos	UMA
Média	Procedimentos	UMA/NUMA
Grossa	Processo	SC-NUMA/NORMA
Muito Grossa	Aplicações	Qualquer arquitetura

Fonte: Cavalheiro (2004).

[3]Para (DE ROSE e NAVAUX 2003, p, 61-62), O próprio nome VLIW, *Very Instruction Word*, Palavra de Instrução Muito Longa, indica ser uma arquitetura que possui mais de uma operação por instrução.

Em extensão ao modelo *Flynn*, surgem os termos *Single Program, Multiple Data* (SPMD) e *Multiple Programs, Multiple Data* (MPMD) respectivamente um único programa, múltiplos dados e múltiplos programas múltiplos dados. Essa extensão leva a estruturas de execução onde existe: (i) um único código sendo executado por vários processadores; e (ii) vários códigos distintos sendo executados por vários processadores. Em ambos os casos manipulando conjunto de dados distintos.

Já o modelo de (REWINI e T. LEWIS, p. 1997), é um bidimensional, sendo baseado no grau de acoplamento provido pela arquitetura (de fortemente acoplada à base de dados compartilhados) e pela granulosidade da aplicação (fina, média, grossa).

Outra classificação bastante conhecida é a de (SKILLIOCORN e TALIA, p. 1998), a qual se refere exclusivamente à questão de programação. Como na proposta anterior, são utilizadas duas dimensões para classificar os diferentes modelos. A primeira diz respeito ao grau de abstração que o programador pode contar no desenvolvimento de seu código concorrente. Esta primeira dimensão possui as seguintes possibilidades:

- Modelo com paralelismo implícito;
- Modelo com paralelismo explícito, mas decomposição implícita;
- Modelo com mapeamento implícito, mas com comunicação explícita;
- Modelo com comunicação explícita, mas com sincronização implícita;
- Modelo onde o programador é responsável por todas as tarefas.

A primeira dimensão distingue os modelos de programação concorrente em função ao grau de abstração que o programador pode ter no desenvolvimento de seu código. Para a segunda dimensão caracteriza os modelos em função das comunicações realizadas.

- Número dinâmico de processos envolvidos no cálculo e volume de comunicação ilimitado;
- Número fixo de processos e volume de comunicação ilimitado;
- Número fixo de processos e volume de comunicação limitado.

De acordo com (PITANGA, 2008, p.156), as sincronizações dos processos geram muitos problemas ao programador, pois as tarefas executadas em paralelo esperam a finalização mútua para poder então coordenar os resultados ou trocar os dados, e reiniciar novas tarefas em paralelo.

É necessário que haja uma perfeita coordenação do tamanho dos processos (granulosidade) e da comunicação entre eles para que o tráfego gerado não seja maior do que o processamento e que por isso, consequentemente, tenha perda no desempenho dos processos em execução.

O tempo perdido para coordenar as tarefas que executam em paralelo é resumido em: (i) tempo para inicializar uma tarefa; (ii) tempo para sincronizar as tarefas; (iii) tempo para comunicação entre as tarefas; e (iv) tempo para finalizar uma tarefa.

2.8 MODELOS DE PROGRAMAÇÃO

Para (LEOPOLD, p. 2001), modelos são na verdade descritos segundo o ponto de vista do programador. Isto quer dizer que estão de fato associados ao uso prático no desenvolvimento de aplicações. Dada esta característica, não é de se surpreender caso um modelo seja encontrado em diferentes ambientes e ou linguagens ao contrário esta situação é bastante comum. O Quadro 8 expõe a relação entre o paradigma do Paralelismo de tarefas versus o Paralelismo de dados, visto que, o primeiro destaca a precaução do programador em detectar as atividades concorrentes da aplicação visando bons escalonamentos aos recursos disponíveis, e posteriormente, o paralelismo é aplicado em decorrência dos dados a serem acessados, consistindo na integridade dos dados. Para o modelo de Memória compartilhada versus Troca de mensagens, neste caso, primeiramente o compartilhamento ocorre através da aplicação de técnicas para o acesso a memória, e em seguida, a comunicação é em decorrência da troca de mensagens através da rede que interconecta os *hosts* do ambiente computacional.

Quadro 8: Modelo de Programação.

Paralelismo de dados vs. Paralelismo de tarefa	
Paralelismo de Tarefa:	O primeiro modelo apresentado trata do paralelismo de execução de tarefas. Este modelo tem como característica a execução paralela de diferentes atividades sobre os conjuntos distintos de dados. Neste caso, o programador dedica seu esforço em identificar as atividades concorrentes da aplicação e deve ser observada a distribuição destas tarefas nos recursos de *hardware* disponíveis. Embora este modelo possa parecer um tanto óbvio, ele contrasta com o modelo de paralelismo de dados;
Paralelismo de Dados:	Com uma abordagem completamente ortogonal ao modelo de paralelismo de tarefas, o modelo de paralelismo de dados é caracterizado pela execução paralela de uma mesma atividade sobre diferentes partes de uma mesma coleção de dados. Neste caso, são os dados que determinam a concorrência da aplicação e a forma como o cálculo deve ser distribuído na máquina;
Memória compartilhada vs. Troca de mensagens	
Compartilhamento de Memória:	Neste modelo, as tarefas em execução compartilham um mesmo espaço de memória. A comunicação entre estas pode ser efetivada através do simples acesso a esta memória. Variáveis compartilhadas são, portanto o meio de comunicação utilizado;
Troca de Mensagens:	Caso não exista um espaço de endereçamento comum, a opção para efetivar a troca de dados entre tarefas recai em utilizar troca de mensagens de forma explicita. Ou seja, uma rede de interconexão é explorada para o envio e recebimento de mensagens.

Fonte: Cavalheiro (2004).

3 METODOLOGIA

O âmbito do assunto selecionado foi na classificação de clusters de computadores do tipo *Beowulf*, baseados em arquiteturas abertas, uma vez que, tem como objetivo central buscar soluções aos problemas no contexto da computação de alto desempenho e alta disponibilidade com emprego de *hardware* de baixo custo e *software* livre.

Utilizou-se na pesquisa, a abordagem metodológica do tipo exploratória, descritiva, a coleta de subsídios foi através de pesquisa bibliográfica em livros artigos e sites relacionados aos assuntos abordados. Análise dos dados foi baseada em dados obtidos que são apresentados em tópicos próprios.

4 CONSIDERAÇÕES FINAIS

O universo da problemática tratada pela computação de alto desempenho e alta disponibilidade com a utilização dos *clusters Beowulf* já é uma realidade há algum tempo, grupos de pesquisa em diversos lugares do mundo vêm trabalhando para cada vez mais obter o melhor proveito nestes ambientes.

Na maioria dos casos destaca se os problemas científicos, simulações que envolvam grandes computações, cálculos complexos que geram grandes saídas, problemas de multimídia e que envolva grande quantidade de dados, assim como, engenharia fotônica, sensoriamento remoto, engenharia genética, sísmica, meteorologia, pesquisas militares e segurança de reatores nucleares. Manter um sistema disponível na maior parte do tempo é difícil, e o gerenciamento é uma tarefa complexa.

As tecnologias de redes nos dias de hoje permitem efetuar o gerenciamento destes aglomerados e abstraem do usuário as tarefas de monitoramento e escalonamento deixando os programadores livres para preocuparem-se com o problema principal, os algoritmos.

Fica evidenciado que o principal objetivo da computação paralela é a divisão de tarefas, no entanto, existe um limite que precisa ser tratado pois o trabalho só será considerado eficiente se existir uma perfeita sincronização das tarefas, e na totalidade não são todos os problemas que podem ser paralelizados para obter sua solução rápida.

Neste contexto nasce o problema, o quanto que uma tarefa pode ser paralelizada sem que tenha a perda de desempenho ou até mesmo a sua inviabilidade de paralelização, pela dificuldade da sincronização das partes.

Para problemas que exijam uma constante e grande troca de informações os *clusters* não são uma boa alternativa, pois limita se nas tecnologias de rede. Fica a carga então ao programador para elaborar os programas com o mínimo possível de troca de informações e diminuir o tempo de espera.

A alta disponibilidade dos serviços e a tolerância a falhas e escalabilidade são vantagens obtidas com o uso dos *clusters* de computadores.

REFERÊNCIAS

BACELLAR, H. Viana. **Cluster: Computação de Alto Desempenho**. Instituto de Computação, Universidade Estadual de Campinas. Campinas, São Paulo, 2009. Disponível em: <http://www.ic.unicamp.br/~ducatte/mo401/1s2010/T2/107077-t2.pdf> Acesso em: 6 out. 2014.

BORGES, Emerson da Silva. **Grids Computacionais: Uma Proposta de Método de Planejamento de Execução de Aplicação Baseada no Tipo de Tarefa com o Foco na Análise do Desempenho**. Dissertação de Mestrado. Centro Estadual de Educação Tecnológica Paula Souza, Programa de Mestrado em Tecnologia: Tecnologia da Informação Aplicada, São Paulo, 2012.

CIRNE, ALFREDO. **Grids Computacionais: Arquiteturas, Tecnologias e Aplicações**. Escola Regional de Alto Desempenho (ERAD),Santa Maria/RS, 2003.

COLVERO, TAÍS. DANTAS, MAR. CUNHA, DANIEL PEZZI. **Ambientes de Clusters e Grids Computacionais: Características, Facilidades e Desafios**. Anais SULCOMP, Vol. 1, No 1. 2005.

CAVALHEIRO, GERALDO HOMRICH. **Princípios da Programação Concorrente**. ERAD 2004, 4ª Escola Regional de Auto Desempenho. Pelotas, RS Anais 2004.

C. LEOPOLD. **Parallel and Distributed Computing**, John Wiley and Sons, New York, 2001. ERAD 2004, 4ª Escola Regional de Auto Desempenho. Pelotas, RS Anais 2004.

DE ROSE, CESAR A. F, NAVAUX PHILIPE O. A. **Arquiteturas Paralelas**. 1ª e.d: Porto Alegre-RS,Instituto de Informática da UFRGS: Sangra Luzzato, 2003.

DANTAS, M. **Computação Distribuída de Alto Desempenho**. 1. ed. Rio de Janeiro: Axcel, 2005.

SKILLICORN, David B.; TALIA, Domenico. **Models and languages for parallel computation**. Acm Computing Surveys (Csur), v. 30, n. 2, p. 123-169, 1998.

FERREIRA, Rubem E. **Linux Guia do Administrador do Sistema**-2ª Edição. Novatec Editora,2008.

FLYNN, M. J. (1972). **Some Computer Organizations and Their Effectiveness**. IEEE Transactions on Computers, v.21, n.9.

EL-REWINI, Hesham; LEWIS, Ted G. **Distributed and parallel computing**. Manning Publications Co., 1998.

M. A. R. DANTAS. **Tecnologias de Redes de Comunicação e Computadores**. 1ª e.d: Rio de Janeiro: Axcel Books, 2003.

MYRICOM. ***Ultra-High Performance Ethernet Networking for a Broad Range of Applications***. 2014. Disponível em: <https://www.myricom.com/solutions/overview.html> Acesso em: 5 ago. 2014.

PITANGA, Marcos. **Construindo Supercomputadores com Linux**. 3ª e.d, Brasport, Rio de Janeiro, 2008.

BUYYA, Rajkumar. **High Performance Cluster Computing: Architecture and Systems**, Volume I. Prentice Hall, Upper SaddleRiver, NJ, USA, v. 1, p. 999, 1999.

TANNENBAUM, A. **Sistemas Operacionais Modernos**. Rio de Janeiro: LTC, 2001.