CEEL - ISSN 2596-2221
Universidade Federal de Uberlândia
13 a 17 de dezembro de 2021

ESTUDO DOS ASPECTOS QUALITATIVOS E COMPARATIVOS DA IMPLEMENTAÇÃO DO *CLUSTER* OPENMOSIX ATRAVÉS DO BCCD E MOSIX

Alana da Silva Magalhães*[1], Eduardo Henrique de Souza[1], Hamsés Peron Ribeiro Pires[1], Lucas Afonso Lima Pereira[1], Wesley Pacheco Calixto[1,2] e Júnio Santos Bulhões[2,3]

[1]IFG - Instituto Federal de Goiás
[2]UFG - Universidade Federal de Goiás
[3]IFMT - Instituto Federal de Mato Grosso

***Resumo* - O avanço tecnológico em que o mundo vive hoje é resultado de inúmeras linhas de pesquisas desenvolvidas pelas várias áreas do conhecimento científico. A computação é sem sombra de dúvidas, a área da ciência que mais tem avançado nos últimos anos. Dentro da área da computação há uma linha de pesquisa voltada ao processamento distribuído que há anos vem sendo estudada por universidades e centros de pesquisas de todo o mundo. A partir desses estudos foi possível a construção do *clustering computing* o que possibilitou às universidades e instituições de pesquisas de pequeno e médio porte construir máquinas com alto poder de processamento a baixo custo, uma vez que supercomputadores independentes são inviáveis devido ao seu alto custo. Com isso foram criados inúmeros softwares com o objetivo de tornar viável o uso dessa tecnologia. Nesse contexto, é apresentado um estudo resumido sobre a tecnologia *clustering* e uma proposta para análise de comparação entre dois *clusters* de mesma categoria, sendo que um é o Mosix, que é um software proprietário e o outro o openMosix, que é um software livre.**

***Palavras-Chave* - *Cluster*, Processamento Distribuído, Computação Paralela, Mosix, openMosix, BCCD.**

STUDY OF QUALITATIVE AND COMPARATIVE ASPECTS OF THE IMPLEMENTATION OF THE OPENMOSIX CLUSTER THROUGH BCCD AND MOSIX

***Abstract* - Technological advancement in the world lives today is the result of numerous lines of research pursued by several areas of scientific knowledge. The computation is undoubtedly the area of science that has advanced the most in recent years within the field of computing there is a line of research aimed at distributed processing that has been studied for years by universities and research centers around the world. From these studies it was possible to build the computing clustering which allowed universities and research institutions to small and medium build machines with high processing power at low cost, since independent supercomputers were unviable due to its high cost. With so many software were created in order to make feasible the use of this technology. In this context we present a brief study on the technology and a proposal for clustering analysis comparing two clusters of the same category, and one is the Mosix, which is proprietary software and other openMosix, which is free software.**

***Keywords* - Cluster, Distributed Processing, Parallel Computing, Mosix, openMosix, BCCD.**

I. INTRODUÇÃO

Com o avanço da vida moderna os computadores tornaram-se ferramentas imprescindíveis para o futuro da humanidade, pois tanto para o entretenimento ou em qualquer área do conhecimento tem-se uma grande demanda computacional, seja tanto para pesquisas como para execução de tarefas diárias. Essa demanda acabou tornando possível a popularização dos computadores pessoais (ou microcomputadores) principalmente pelos preços acessíveis. O que não aconteceu com os computadores de grande porte que ainda mantêm um alto custo se relacionados às empresas e instituições de pequeno e médio porte.

Uma solução que já vem sendo tomada em função desse alto custo é o chamado sistema em *cluster* que consiste em vários microcomputadores ligados em uma rede onde é possível utilizar seus processadores como se fosse um supercomputador com vários processadores. Segundo [1] a computação em *cluster* originou-se com o projeto SAGE (*Semi-Automatic Ground Environment*) construído para o NORAD (*North American Air Defense*) pela IBM em 1962 e consiste de vários sistemas separados trabalhando de forma cooperativa para monitorar invasões aéreas no continente norte-americano. Um *cluster* de máquinas não necessariamente precisa ser utilizado para aumentar o poder de processamento, pode ser também

*alana.magalhaes@ifg.edu.br

utilizado com o objetivo de aumentar a disponibilidade de um determinado serviço ou no balanceamento de carga entre os serviços disponíveis [1].

O objetivo deste trabalho é usar a tecnologia de processamento distribuído ou (*cluster*) para o aumento da capacidade de processamento dos computadores de baixo rendimento que equipam alguns dos laboratórios, da área de Telecomunicações do IFG. Possibilitando assim, novas aplicações que antes eram consideradas inviáveis por se tratar de maquinas que já se encontram um tanto defasadas devido a sua idade. Nesse trabalho é abordado a implementação de dois *clusters* de mesma categoria, o Mosix e o openMosix. Os dois *clusters* foram instalados nas mesmas máquinas com o objetivo de garantir o mesmo ambiente computacional para que pudessem ser avaliados sobre a mesma ótica e sem nenhuma interferência externa, o que vale ressaltar é que a diferença entre os dois está em que o Mosix é um software proprietário e openMosix um software livre.

II. CLUSTER

Cluster pode ser definido como um conjunto de computadores, chamados nós, interligados em rede, que dividem entre si o processamento de uma tarefa de forma homogênea, ou seja, realizam essa atividade como se fossem uma só maquina o que é chamado de imagem única, pois toda a parte de controle e distribuição da carga entre os computadores ficam transparentes ao usuário final [2, 3].

A. Tipos de Cluster

Os tipos de *cluster* são nomeados de acordo com o fim para que são implementados. Segundo [4] pode-se dividir os *clusters* em: *Cluster* de Alta Disponibilidade (HA – *High Availability*) *Cluster* de Balanceamento de Carga (HS – *Horizontal Scaling*), *Cluster* de Alto Poder de Comutação (HPC – *High Performance Computing*).

A função essencial de um *cluster* HA é não deixar um determinado serviço parar de funcionar por causa de defeitos de hardware nos dispositivos que o disponibilizam. Mesmo que haja uma falha em um dos equipamentos que compõe o *cluster*, o fato do serviço ser replicado e balanceado entre os outros nós, não o deixa ser interrompido. De fato, pode haver perda de desempenho no processamento, mas o foco de estar sempre disponível será mantido.

Para [4] um Cluster de Balanceamento de Carga (HS), tem a função de distribuir o tráfego e requisições de máquinas que estão no sistema de forma rápida e equilibrada.

Para fazer esse balanceamento esse cluster pode usar algoritmos de escalonamento. Dentre os mais conhecidos: Least Connection que encaminha as requisições para as maquinas com menos número de solicitações; Round Robin que encaminha as solicitações de forma sequencial e Weighted Fair que encaminha as requisições para a máquina mais robusta ou com maior capacidade de carga.

Um *cluster* HPC, visa atender aplicações que demandam grande quantidade de recursos de processamentoe pode ser dividida em dois grupos: Bibliotecas de Programação Paralela [5] e Sistema de Imagem Única (SSI - Single System Image) [6].

Nas Bibliotecas de Programação Paralela há a necessidade de adaptar o software para as bibliotecas de programação paralelas que compõe o *cluster*, dessas bibliotecas as mais utilizadas são o MPI (*Message Passing Interface*) e o PVM (*Parallel Virtual Machine*), o que torna mais complicado o uso de aplicações desenvolvidas de forma sequencial, pois haverá a necessidade de analisar os pontos onde precisará de mais desempenho computacional e a possibilidade de paralelização.

No SSI o *cluster* simula uma única máquina com os recursos de todos os nós ligados nele, ou seja, centraliza todos os recursos de hardware do *cluster* em um único ponto, o que faz com que, pelo menos na teoria, qualquer aplicação que suporte processadores paralelos seja utilizada no *cluster*. Esse tipo de *cluster* atende também à disponibilidade dos serviços, uma vez que a adição ou subtração de nós não afeta o funcionamento do *cluster* em si.

B. Ferramentas de Programação Paralela

Em um *cluster* os processadores necessitam trocar informações para que haja cooperação entre eles no processamento. Para haver essa comunicação entre os processadores de cada nó do *cluster* durante a execução de um programa há a necessidade de ferramentas específicas para que tal programa permita o processamento paralelo. Dentre as ferramentas têm-se as bibliotecas PVM e MPI [7].

A biblioteca PVM permite o funcionamento de forma eficiente e transparente da distribuição de tarefas entre máquinas ligadas em rede, além de possibilitar um eficiente gerenciamento dos recursos dessa rede com n maquinas, por simular uma maquina virtual com n processadores. É um mecanismo de distribuição livre e oferece uma grande quantidade de recursos para a computação paralela.

MPI é um padrão de interface para troca de mensagens entre máquinas paralelas com memória distribuída. É resultante de um trabalho conjunto de fabricantes de computadores paralelos, pesquisadores de universidades, laboratórios, e autoridades governamentais, principalmente da Europa e dos Estados Unidos. Nesse padrão, é definido um conjunto de rotinas com o objetivo de facilitar a comunicação entre os processos.

O SSI visa tornar transparente ao usuário a complexidade dos sistemas distribuídos, ou seja, o usuário terá todo recurso do sistema com a impressão de ter a frente a uma única máquina. Dessa forma o sistema aceita o uso de nós heterogêneos que são facilmente gerenciados [8].

C. Mosix

O projeto Mosix consiste em um sistema operacional distribuído, desenvolvido inicialmente pela equipe do professor Amnon Barack, na Universidade Hebrew em Jerusalém, Israel [9, 10]. Foi utilizado nos anos 80 pela força aérea americana para construção de um *cluster* de computadores PDP11/45. O projeto foi desenvolvido em sete fases, para diferentes versões de UNIX e arquiteturas de computadores. A primeira versão para PC foi desenvolvida para BSD/OS (Berkeley Software Design/Operational System). A última versão

foi para o sistema operacional Linux em plataforma Intel [1].

D. OpenMosix

O motivo principal para a criação do openMosix foi o fato do Mosix se tornar uma ferramenta comercial o que chocou com os princípios de alguns pesquisadores, que acreditavam na importância desse sistema ter código aberto. Moshe Bar que trabalhou como co-gerente de projeto e gerente geral do Mosix, criou uma nova empresa de *clusters* (Qlusters) e decidiu continuar o projeto Mosix com um novo nome e nova licença: openMosix com licença GPLv2 [11]. Porém o Mosix ainda disponibiliza uma versão livre de custo para um *cluster* com até seis máquinas.

E. BCCD

O BCCD (*Bootable Cluster CD*) é um sistema operacional desenvolvido para criar ambientes de processamento distribuídos em laboratórios estudantis onde geralmente não se tem grandes recursos de hardware e muito menos de pessoal para administrar e configurar ambientes de *clusters* mais complexos.

O BCCD é inerentemente customizável por projeto. O sistema todo é dinamicamente construído a partir de fontes na web, buscados em um processo chamado de compilação cruzada. Este processo de construção reúne o código fonte com um ambiente personalizado, a fim de produzir a imagem final, o CDROM (Compact Disc Read-Only Memory). Isto permite especialmente a fácil construção de imagens personalizadas do BCCD, tais como imagens que contêm aplicativos adicionais ou imagens que requerem uma chave privada RSA pessoal que pode ser usada na construção de autenticação de usuário. O ambiente BCCD não requer configurações de inicialização e utiliza configurações mínimas de sistema, pode-se dizer que uma das suas grandes vantagens está no fato de não ser necessário a sua instalação nas unidades de discos rígidos das estações de trabalho que farão parte da rede do *cluster*.

III. DESENVOLVIMENTO

Foram implementados dois *cluster*s do tipo SSI. Um é o Mosix e o outro é o openMosix através da plataforma BCCD. Para que pudesse ser feita uma comparação entre eles, optou-se por instalá-los nos mesmos computadores e dessa forma também foi utilizada a mesma rede com a finalidade de conservar o mesmo ambiente computacional.

A. Hardware

Foram utilizados quatro PCs para implementar o *cluster*. Todos eles possuem a mesma configuração como mostra a Tabela 1.

Tabela 1: Configuração dos Computadores Utilizados nos Clusters. Fonte: [Autores]

Placa Mãe	VIA P4MA Pro Slot 533
Processador	Intel Pentium 4 2.8 Ghz
Disco Rígido	Seagate Barracuda ST340014A 40GB
Memória RAM	1 GB DDR SDRAM 200 MHz
Drive Óptico	SONY CD-ROM CDU5221 52x
Placa de Rede	Realtek RTL8139 Family PCI Fast Ethernet 10/100
Sistema Operacional 1	Debian Squeeze
Sistema Operacional 2	Live CD BCCD 2.2.1

B. Configuração do Mosix

Primeiramente, o Mosix faz o que se chama de recompilação do kernel e para isso é necessário que a versão do kernel a ser recompilado seja compatível com a versão do Mosix escolhida. Para este projeto foi utilizada a versão 2.29.0.2 do Mosix que é compatível com o Kernel do GNU/Linux na versão 2.6.37.1. O sistema operacional utilizado foi o Debian Squeeze.

C. Ferramentas de Monitoramento

O pacote de instalação do Mosix oferece poucas ferramentas para monitorar o *cluster*, porém é suficiente para acompanhar o seu funcionamento, sendo as principais o mon e o mosps. O mon é um programa monitor que tem o objetivo de exibir no Terminal do Linux o estado dos recursos de cada nó e de todo o *cluster*. Ele mostra informações básicas de cada nó como o nível de carregamento de processos, a quantidade de memória RAM (*Random Access Memory*) e memória *Swap* utilizada no momento e os nós ativos e inativos. Cada um desses recursos é acessado dependendo do pós-fixo utilizado com o comando mon.

Outra ferramenta utilizada para monitorar o *cluster* foi o mosps. Ela também é exibida no terminal do Linux e tem o objetivo de fornecer detalhes dos processos que estão sendo executados no *cluster* como: o número de identificação de cada processo, onde eles estão sendo carregados, de qual máquina eles vieram, qual o comando aplicado para que eles fossem executados, entre outros.

No pacote de instalação do Mosix não há nenhuma ferramenta para monitorar o tempo de execução dos processos no *cluster*. Para isso utilizou-se um comando nativo do Linux conhecido como date. Ele oferece o horário atual do sistema e desta forma pode-se aproveitar a funcionalidade para intercalar o comando date como pré-fixo e pós-fixo do comando de execução do algoritmo de teste. Desta forma é possível visualizar o tempo de execução do programa em cenários variados de quantidade de nós inseridos no *cluster* para que se possam fazer comparações de desempenho.

IV. RESULTADOS

A metodologia utilizada para verificar o desempenho dos dois *clusters* consiste em realizar uma sequência de testes com uma mesma quantidade de nós para obter uma média dos dados coletados como tempo de processamento e carga dos nós. O algoritmo DistKeyGen foi executado dez vezes seguidas,

iniciando com um nó no aglomerado. Depois de concluído acrescentava-se mais um nó e fazia-se o mesmo procedimento. Este era repetido até finalizar com quatro nós. Para tornar mais consistente os resultados, foi feita a média aritmética dos valores obtidos em todos os testes em cada cenário.

V. ALGORITMO DE PROCESSAMENTO PARALELO

Com o objetivo de realizar testes de desempenho e demonstrar o funcionamento dos *clusters*, foi utilizado um algoritmo escrito em C++ nomeado por DistKeyGen, construído com o objetivo de criar 4000 pares de chaves criptográficas RSA, sendo que cada chave tem um tamanho de 1024 bits.

Com esse algoritmo foi possível realizar testes consistentes com os *clusters* Mosix e openMosix, uma vez que esse programa carrega consideravelmente o processador para criar combinações diferentes de chaves criptográficas.

A. *Teste do Cluster Mosix*

Primeiramente, foi realizado o teste sem o *cluster* Mosix (ou pode-se dizer com apenas um nó), com o objetivo de se saber qual o desempenho que uma única máquina com um único processador pode atingir.

A partir disso podem-se fazer análises comparativas de uso entre um computador e um aglomerado de computadores. Depois de concluída essa primeira etapa de verificação, realizou-se a adição de nós ao *cluster*. Foram feitos testes com dois, três e quatro nós. Na Tabela 2 pode-se verificar o tempo que cada um levou para ser concluído nos dez testes realizados em cada cenário.

Tabela 2: Tempo de Execução do Algoritmo de Teste no *Cluster* Mosix
Fonte: [Autores].

Teste	1 nó	2 nós	3 nós	4 nós
1	494s	309s	234s	187s
2	502s	274s	238s	191s
3	499s	286s	234s	189s
4	486s	294s	234s	189s
5	497s	290s	233s	197s
6	492s	285s	233s	194s
7	494s	289s	229s	190s
8	494s	301s	230s	196s
9	511s	274s	230s	186s
10	498s	280s	233s	190s
Média	495s	287s	233s	190s

Usando o monitor do *cluster* Mosix pode-se perceber também o nível de carregamento de cada nó. A Figura 1 mostra o nível de carga de processamento para o *cluster* com um, dois, três e quatro nós.

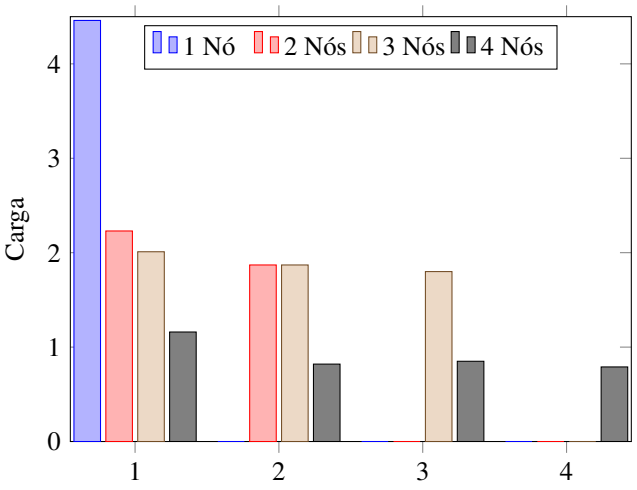

Figura 1: Nível de Carregamento dos Nós do *Cluster* no Quarto Cenário
Fonte: [Mosix]

Utilizando como referência o tempo médio de execução do algoritmo no *cluster*, pode-se destacar que houve uma melhora de 42% no tempo de execução do algoritmo com um cenário de dois nós, de 53% com três nós e de 62% com quatro nós quando se compara com a sua execução em apenas um nó.

Pode-se perceber também que o ganho de tempo entre os cenários com dois nós e três nós foi menos significativo, atingindo um percentual de 18%. Esse mesmo valor foi encontrado na comparação do cenário de três nós com de quatro nós.

Apesar de o ganho de tempo ser menor à medida que se acrescentam mais nós no *cluster*, ao final do teste com todos os nós possíveis, tem-se um ganho bastante significativo que pode ser muito útil para se executar vários processos em um computador ao mesmo tempo. Através da Figura 2 tem-se uma noção visual de quanto se ganha em desempenho a medida que se acrescenta nós no *cluster*.

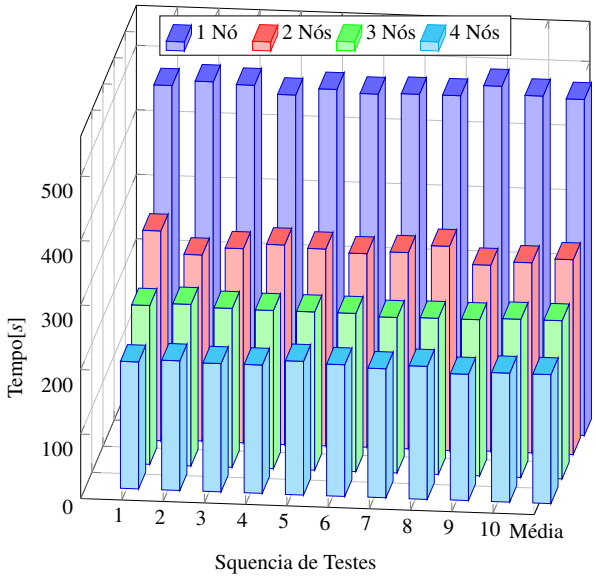

Figura 2: Tempo de Processamento do Algoritmo no *Cluster* Mosix em Todos os Cenários Fonte: [Mosix]

B. *Teste do Cluster OpenMosix*

Este teste foi feito de forma análoga ao teste do Mosix, como descrito no item A. Primeiro foi realizado o teste em um único nó do *cluster* com o intuito de saber qual o desempenho de só uma máquina executando o algoritmo. Este teste também se utilizou do mesmo aplicativo DistKeyGen. O *cluster* funcionou como se tivesse apenas um nó. Dessa forma foi possível conseguir parâmetros para a comparação do poder de processamento em único computador e o poder de processamento de um *cluster*. Foram coletados resultados nos quatro cenários diferentes (com um, dois, três e quatro nós). A Tabela 3 contém o tempo que cada um levou para ser concluído nos dez testes realizados em cada cenário.

Tabela 3: Tempo de Execução do Algoritmo de Teste no *Cluster* openMosix Fonte: [Autores].

Teste	1 nó	2 nós	3 nós	4 nós
1	672s	367s	288s	232s
2	679s	356s	285s	224s
3	674s	353s	280s	229s
4	674s	360s	276s	219s
5	670s	359s	270s	224s
6	684s	356s	271s	233s
7	679s	358s	275s	224s
8	682s	355s	289s	224s
9	669s	357s	289s	228s
10	673s	361s	290s	224s
Média	675s	357s	281s	226s

Utilizando o monitor mosmon pode-se visualizar o nível de carga de cada nó. A Figura 3 mostra o nível da carga de processamento para o *cluster* com um, dois, três e quatro nós.

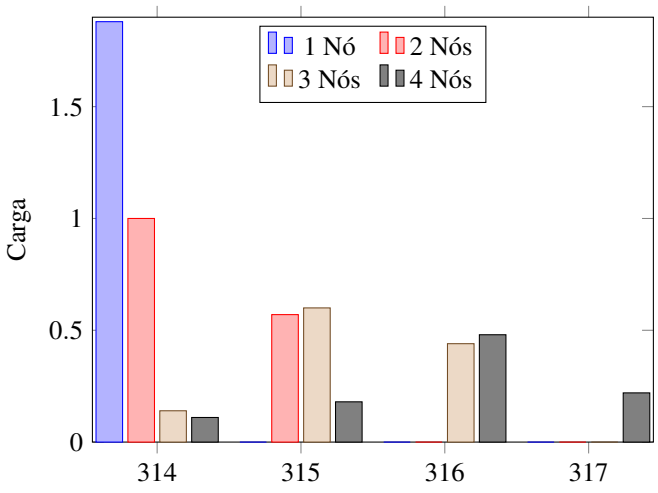

Figura 3: Nível de Carregamento dos Nós do *Cluster* no Quarto Cenário Fonte: [Mosix]

Analisando o tempo médio de execução do algoritmo no *cluster*, pode-se notar que houve uma melhora de 47,11% no tempo de execução do algoritmo com um cenário de dois nós, de 58,37% com três nós e de 66,52% com quatro nós quando comparado com a sua execução em apenas um nó.

Pode-se perceber também que o ganho de tempo entre os cenários com dois nós e três nós foi menos significativo, atingindo um percentual de 21,9%. No outro cenário foi ainda menor o valor encontrado na comparação do cenário de três nós para com o de quatro nós o valor atingido foi de 19.57%.

Apesar de o ganho de tempo ter sido sempre menor à medida que foram acrescentados mais nós no *cluster*, mas, ao final do teste com todos os nós possíveis, é visivelmente notável um ganho bastante significativo no poder de processamento o que pode ser muito útil para se executar vários processos em um computador ao mesmo tempo. Através da Figura 4 tem-se uma noção visual de quanto se ganha em desempenho à medida que se acrescentam nós no *cluster*.

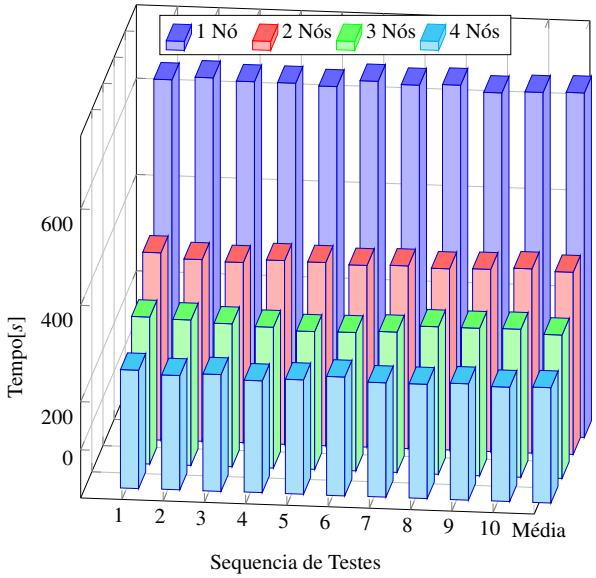

Figura 4: Tempo de Processamento do Algoritmo no *Cluster* openMosix em Todos os Cenários Fonte: [Autores]

A Tabela 4 apresenta a diferença entre os resultados obtidos com o *cluster* Mosix e o openMosix nos quatro cenários diferentes (com um, dois, três e quatro nós). Os dados apresentados contém o tempo que cada um levou para ser concluído nos dez testes realizados em cada cenário.

Tabela 4: Comparativo de tempo entre Mosix e OpenMosix. [Autores].

Teste	1 nó	2 nós	3 nós	4 nós
1	180s	58s	54s	45s
2	177s	82s	47s	33s
3	175s	67s	46s	40s
4	188s	66s	42s	30s
5	173s	69s	37s	27s
6	192s	71s	38s	39s
7	185s	69s	46s	34s
8	188s	54s	59s	28s
9	158s	83s	59s	42s
10	175s	81s	57s	34s
Média	179s	70s	48s	35s

Na média o resultado do *cluster* Mosix teve melhora de 26,52% no tempo de execução do algoritmo com um cenário de um nó em relação ao openMosix. Nota-se a melhora do *cluster* Mosix de 19,61% no tempo com um cenário de dois nós, de 17,08% no tempo com um cenário de três nós e de 15,49% no tempo com um cenário de quatro nós em relação ao openMosix.

VI. CONCLUSÃO

Verificou-se neste trabalho o estudo de *clusters*, em especial o SSI que promove a migração preemptiva de processos entre nós de um aglomerado de computadores, sendo este o objeto de implementação realizada.

Foram realizados testes de tempo de execução de processos nos dois modelos de *cluster* escolhidos utilizando um algoritmo de teste e através dos resultados colhidos observa-se uma diferença mais significativa entre os resultados quando comparados com uma quantidade menor de nós no aglomerado. O *cluster* openMosix foi o que obteve o desempenho mais baixo, porém, deve-se levar em consideração que ele foi utilizado através da plataforma live-cd BCCD que utiliza a memória RAM para carregar todo o conteúdo do CD e que também é uma ferramenta livre. Além disso, o openMosix possui uma oferta maior de ferramentas de análise, principalmente ferramentas gráficas que facilitam o entendimento do resultado coletado. Já o *cluster* Mosix, que é uma ferramenta proprietária é instalado diretamente no sistema operacional sendo que o utilizado foi o Debian Squeeze. Esse modelo de *cluster* obteve o maior desempenho, entretanto não se tem muitas ferramentas de análise de resultados. Torna-se necessário utilizar métodos manuais ou ter conhecimento maior na área para examinar resultados obtidos nas ferramentas disponibilizadas em modo texto.

É preciso entender que o *cluster* que obteve o maior desempenho não necessariamente deve ser escolhido para qualquer aplicação prática. Cada um possui suas vantagens e desvantagens, devendo analisar qual se encaixa melhor no projeto pretendido. Uma proposta interessante para uma pesquisa seria a aplicação dos sistemas de *cluster* para renderização de imagens, com máquinas que possuem placas de vídeo dedicada. Através deste trabalho pode-se concluir que a tecnologia *clustering* é uma alternativa aos supercomputadores independentes, pois tem um melhor custo benefício, desta forma o objetivo deste projeto foi alcançado com sucesso.

Neste trabalho foi constatada também uma importante ferramenta a ser utilizada em sala de aula para o estudo de *clustering* e processamento distribuído. Trata-se do BCCD. Como já foi dito ele é uma imagem em um live-cd que inicializa um ambiente de computação distribuída já pré-configurado e também provê aplicações e exemplos de repertórios com amplas ferramentas de desenvolvimento, proporcionando assim a facilidade de um ambiente de aprendizado a alunos de graduação e de pós-graduação que não têm acesso a um sistema de *cluster* dedicado. O cenário comum para o uso do BCCD como um ambiente de aprendizado começa com um instrutor em uma sala de aula e com um aluno por estação de trabalho.

REFERÊNCIAS

[1] Marcos Pitanga. *Construindo supercomputadores com Linux*. Brasport, 2004.

[2] Esli P. Faustino and Reinaldo B. Freitas. Construindo supercomputadores com linux: Cluster beowulf. Trabalho de conclusão de curso, Redes de Comunicação. Centro Federal de Educação Tecnológica de Goiás. Goiânia, 2005.

[3] MA MEMON, AK JUMANI, MY KOONDHAR, M Memon, and AG MEMON. A technique to differentiate clustered operating systems. *Sindh University Research Journal (Science Series)*, 50(3D):89–94, 2018.

[4] José C. Aluvino. Alto poder de computação com cluster de computadores usando mosix. Trabalho de conclusão de curso, Curso Superior de Tecnologia em Informática. Faculdade de Tecnologia de Guaratinguetá. Guaratinguetá, 2008.

[5] Michael Biffi Nascimento. Cluster de alto desempenho para uso na disciplina de computação paralela e distribuída utilizando contêineres. 2019.

[6] Namer Ali Al Etawi. A comparison between cluster, grid, and cloud computing. *International Journal of Computer Applications*, 179(32):37–42, 2018.

[7] Kamran Jafarzade. Comparative analysis of the software used in supercomputer technologies. *Problems of Information Technology*, 09:92–97, 01 2018.

[8] Marcelo V. Neves. Modelagem e dimensionamento do custo de migração de processos em programas mpi. Dissertação (mestrado), Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande do Sul. Porto Alegre, 2009.

[9] A Barak and A Shiloh. The mosix cluster operating system for high-performance computing on linux clusters, multi-clusters, gpu clusters and clouds. 2014.

[10] Sarvesh Kumar Dubey and Virendra Upadhyay. Cluster, grid, cloud and parallel distributed computing: An overview with comparison. 2017.

[11] Carlos R. R. da Costa. Construção e configuração de um aglomerado com computadores em laboratórios de informática. Trabalho de conclusão de curso, Curso de Ciência da Computação. Universidade Federal de Santa Maria Centro de Tecnologia. Santa Maria, 2009.