# Provendo maior escalabilidade em HNOW com inclusão dinâmica de workers

Francisco Borges Santana
Centro de Pós-graduação e Pesquisa Visconde de Cairu - CEPPEV
Fundação Visconde de Cairu - Salvador, Bahia, Brasil
borgeskaiser@gmail.com

and

Josemar Rodrigues de Souza
Núcleo de Arquitetura de Computadores e Sistemas Operacionais
Universidade do Estado da Bahia - UNEB. Salvador, Bahia, Brasil
josemar@uneb.br

## Abstract

Heterogeneous Networks of Workstations (HNOWs) constitute an efficient and cheap architecture which can provide greater computational power for scientific calculation. The HNOWs provides attractive scalability in terms of computational power and memory. One form to scale a cluster is to dynamically add more workstations, through libraries such as the MPI for example. However, to take off advantage of this feature, it's required the existence of a policy of load balancing that contemplates the dynamic addition of resources. It is necessary that the dynamic increase of workers either follow of a strategy of load balancing. The research considers, implements and evaluates a model of load balancing that deals with the dynamic inclusion of workstations in HNOW. The implementation is done using the MPICH2 in C/C++ with Master/Worker paradigm. For performance evaluation, we use the metric of speedup of the dynamic inclusions. We've made using as benchmark an algorithm of multiplication of matrix. The considered model provides a bigger scalability in HNOW with the dynamic inclusion of workers.

Keywords: Load balancing, Cluster Computing, Grid Computing, Master/Worker, Parallel algorithm.

## Resumo

As Redes Heterogêneas de Estações de Trabalho (Heterogeneous Networks of Workstations - HNOW) constituem uma arquitetura eficiente e barata para prover maior poder computacional para cálculos científicos. As HNOWs proporcionam uma atrativa escalabilidade em termos de poder computacional e memória. Uma forma de escalar o cluster é adicionar mais estações de trabalho dinamicamente, através de bibliotecas como o MPI por exemplo. Contudo, para tirar proveito dessa facilidade, é necessário que exista uma política de balanceamento de carga que contemple a adição dinâmica de recursos. É necessário que o aumento dinâmico de workers seja acompanhado de uma estratégia de balanceamento de carga. A pesquisa propõe, implementa e avalia um modelo de balanceamento de carga que trata a inclusão dinâmica de estações de trabalho em HNOW. A implementação se faz utilizando o MPICH2 em C/C++ no paradigma Master/Worker. Para avaliação dos resultados são utilizadas a métrica speedup das inclusões dinâmicas. São efetuadas diversas medições utilizando como benchmark um algoritmo de multiplicação de matriz. O modelo proposto prove uma maior escalabilidade em HNOW com a inclusão dinâmica de workers.

Palavras chaves: Balanceamento de carga, Computação em Cluster, Computação em Grades, Master/Worker, Algoritmo paralelo.

# 1 INTRODUÇÃO

A possibilidade de dinamicamente incluir processos é um recurso poderoso e interessante para o desenvolvimento de aplicações paralelas. Essa característica, apesar de já existir no PVM, somente foi disponibilizada no MPI-2. Os fatores motivadores, para a inclusão dessa característica na especificação, foram o sucesso de Grid Computing e a necessidade de adaptar o comportamento dos programas paralelos durante a execução por conta da mudança de hardware [2]. A criação dinâmica processo é exemplifica por [8] com o balanceamento de carga, onde o algoritmo pode escolher a localização de onde os processos serão criados baseados em valores dinâmicos, onde a carga pode transparentemente migrar o processo de uma máquina para outra quando um recurso torna-se sobrecarregado ou indisponível. A exemplo da utilização da criação dinâmica de processo encontramos [2] que apresentam um modulo de scheduler, implementado em MPI-2, que determina em tempo de execução em qual processador um novo processo deve ser executado, provendo dessa forma um balanceamento de carga, outro exemplo que aplica essa mesma técnica é [6] que desenvolve um framework de balanceamento de carga. A criação dinâmica de processos também pode ser utilizada para a implementação de tolerância a falha [4]. O algoritmo paralelo pode perceber a indisponibilidade de determinado node e criá-lo em outro. Outra possibilidade da inclusão dinâmica de processos pode ser aplicada em HNOW - Heterogeneous Networks of Workstations , novos workers podem ser adicionados sob demanda para prover mais poder computacional a uma aplicação que iniciou o processamento com poucos workers. No entanto aumentar a quantidade de workers para ajudar no processamento de determinada tarefa não é o bastante para garantir que o algoritmo paralelo execute de maneira mais eficiente. É necessário que o aumento de workers seja acompanhado de uma estratégia de balanceamento de carga, pois a distribuição de trabalho é um fator crítico para o desenvolvimento de eficientes algoritmos paralelos [9]. O modelo é proposto para aplicações que implementam o paradigma Master/Worker com interações sincronizadas, o modelo ajusta a carga do cluster conforme a inclusão de novos workers. O modelo proposto foi desenvolvido para as HNOW, que utiliza o paradigma Master/Worker [1]. O modelo viabiliza o balanceamento de carga em HNOW, em que ocorrem a adição de workers dinamicamente, entende-se por inclusão dinâmica, a inclusão de novos workers no cluster sem a interrupção do processamento.

# 2 MODELO CONCEITUAL

As HNOW são caracterizadas pelo fato dos workers possuírem recursos computacionais heterogêneos. [11] apresentam diversos fatores de heterogeneidade como arquitetura da máquina, latência de rede, largura de banda, memória, entre outros. Além disso, as HNOW geralmente não são dedicadas para a execução do algoritmo paralelo. Por isso, a capacidade de processamento dos workers é diferente e sofre variações ao longo do cômputo, influenciando a performance do algoritmo. Assim, é necessário que a quantidade de tarefas executadas por determinado worker seja proporcional à sua capacidade de processamento para evitar que todo o cluster fique aguardando pelo término do processamento do worker mais lento. A distribuição das tarefas está relacionada também com a quantidade de workers no cluster. Se a quantidade de workers aumentar a quantidade de tarefas enviadas para os outros workers deve ser reavaliada. A figura

1 apresenta o modelo proposto.

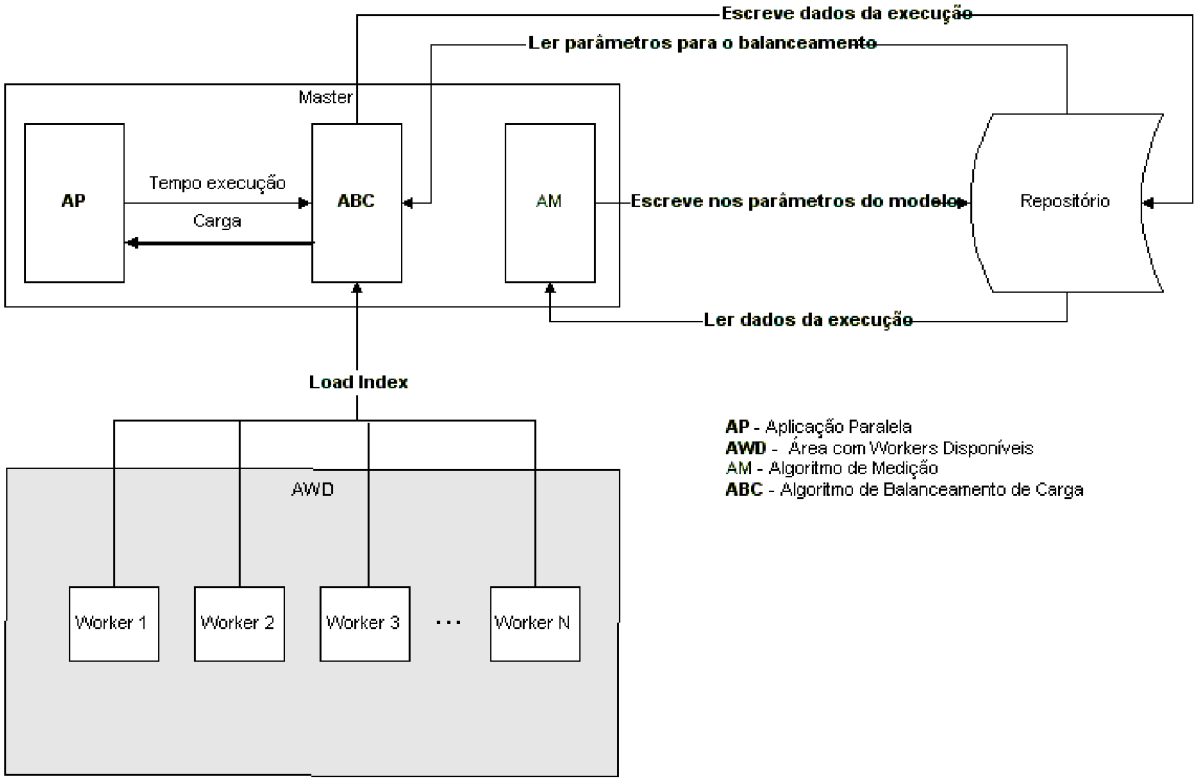



Figura 1: Modelo proposto

O modelo é composto: AWD - Área de Workers Disponíveis, AP - Algoritmo Paralelo, ABC -Algoritmo de Balanceamento de Carga e o AM -Algoritmo de Medição. O repositório é o local onde o master recupera e armazena informações sobre o processamento do algoritmo paralelo e do modelo. O repositório possui os parâmetros de configuração do algoritmo de balanceamento. Um desses parâmetros é o THRESH- OLD MAX que indica, em percentual, o limite superior que o tempo de computo de um worker deve ter. Outro é o THRESHOLD MIN - indica, em percentual, o limite inferior que o tempo de computo de um worker deve ter. O tamanho do bloco inicial que será utilizado para cálculo é indicado pelo parâmetro VALTAM BLOCO. O parâmetro WORKERS ADD que indica quanto workers serão adicionados dinamicamente. E por fim o PERC CONS ADD indica quando, em percentual de dados processados, os workers serão adicionados dinamicamente.
Quando o master executa o algoritmo de balanceamento de carga, as informações contidas no repositório são lidas e passadas como parâmetro para o algoritmo, que efetua os cálculos necessários para a distribuição de carga. Os resultados obtidos após execução são armazenados no repositório. É através do repositório que o algoritmo de balanceamento de carga consegue perceber a existência de recursos computacionais adicionados no cluster.
A Área de Workers Disponíveis é um recurso computacional que possui todos os workers que podem ajudar a executar determinado processamento. Essa área pode compartilhar diversos workers, que podem fazer em dado instante parte de um cluster, que processa um algoritmo A, por exemplo, e em outro momento fazer parte de outro cluster, que processa o algoritmo B. Esse recurso permite a adição

dinâmica de worker e isola a aplicação paralela e o algoritmo de balanceamento de carga, de forma que a manipulação dessa área não os afeta. O algoritmo paralelo deve possuir dados que possam ser divididos em diversas tarefas e cada tarefa deve possuir as mesmas características que a outra.

O algoritmo de balanceamento de carga é iniciado quando o master sincroniza as tarefas enviadas para os workers. Nesse momento, o algoritmo de balanceamento efetua os cálculos necessários e envia a carga apropriada de cada worker. Conforme [10], todo trabalho executado no paradigma master/worker é iniciado quase simultaneamente. Logo, a alocação de tarefas deve ser proporcional à capacidade computacional [3], para permitir que a execução de todos os workers em cada interação termine aproximadamente no mesmo tempo. Com as informações obtidas do repositório e com as informações enviadas dos workers sobre a performance(load index), o algoritmo de balanceamento faz a predicação da carga que deve ser distribuída. Em ambientes homogêneos e dedica- dos, os recursos computacionais, tais como, memória disponível, capacidade de processador, largura de rede e latência de rede são conhecidos, diferentemente de ambientes heterogêneos e não dedicados, onde esses valores variam constantemente. Uma forma de se obter esses valores é através da predicação. Conforme [12], predicar a futura performance baseada em informações do passado é uma abordagem comum, feita para balanceamento de carga em HNOW.

O reajuste de carga é efetuado, quando o tempo de computo ($T_{comp}$) dos workers está fora do intervalo definido pelo threshold superior e inferior em determinada interação. O valor do threshold é calculado sobre o valor do pior tempo de execução obtido em cada interação. Assim como salientam [10], levar-se-´a em consideração o pior caso em que o tempo de comunicação e o de computação não são sobrepostos. Dessa forma, pode-se identificar qual parâmetro de heterogeneidade está influenciando no tempo de execução da interação: tempo de comunicação ou tempo de computação. A carga somente é ajustada para aqueles workers cujo tempo de computação $T_{comp,i}$ não esteja no intervalo dos thresholds inferior e superior, previamente definido. Os thresholds inferior e superior são um percentual sobre o tempo previsto para a execução de uma interação. Se o $T_{comp,i}$ estiver abaixo do threshold inferior, indica que o worker está subutilizado. Acima do threshold superior, indica uma sobrecarga. Entre esses intervalos, indica que a carga está em uma situação ideal. Portanto, nota-se que o tempo de computação da tarefa $T_{comp,i}$ é dependente do processador e/ou da memória disponível no worker para a execução da tarefa. Logo, infere-se, que se o tempo de computação for maior que o tempo previsto para a interação, há uma sobrecarga de trabalho ou o worker, não possui recursos Disponíveis, no momento, para atender a requisição de forma eficiente. O tempo de execução de uma tarefa, porém, depende da aplicação paralela e da heterogeneidade do ambiente. Por isso, a necessidade de se caracterizar cada worker na primeira interação. A caracterização consiste no master enviar uma tarefa com as mesmas características para ser processada, a fim de obter a capacidade computacional do worker. O pior tempo de computação obtido será considerado como o tempo necessário para executar uma tarefa. Assim, a performance ficará limitada ao worker mais lento. A quantidade de tarefas enviadas para cada worker na primeira interação é definida pela Eq. 1.

$$QtarefaW = TAM\ BLOCO * FatorHet_i \quad (1)$$

Onde TAM_BLOCO é o tamanho do inicial enviado para todos os workers o qual é definido em arquivo de configuração do modelo e $FatorHet_i$ é o fator de heterogeneidade do worker que receberá a carga. Como trata-se da primeira interação esse valor é igual a 1 para todos os workers. No entanto esse fator de heterogeneidade pode variar em cada interação e pode variar em diferentes aplicações paralelas. O fator de heterogeneidade, representado na equação 2, é calculado dividindo a quantidade de blocos enviados na interação anterior sobre a quantidade de blocos inicial. Por isso esse fator é sensível aos recursos computacionais disponíveis, se um worker processar mais em uma interação significa que seu fator de heterogeneidade será maior na próxima interação, logo receberá mais trabalho, o contrário ocorre caso o fator de heterogeneidade seja menor. Além disso, esse fator retrata a diferença computacional existente entre os workers. A tendência é que esse valor seja uma constante durante a execução do algoritmo. Nem sempre a quantidade de tarefas que deve ser enviada será um número inteiro, nesses casos efetua-se um arredondamento para a parte inteira.

$$FatorHet_i = \frac{FatorHet,i\ Anterior}{TAM\_BLOCO} \quad (2)$$

Cada worker irá processar Ns tarefas conforme sua capacidade de processamento e disponibilidade de recursos - processador e memória. A quantidade de tarefas executadas por cada worker a partir da segunda interação é definida no momento da sincronização, seguindo os critérios abaixo:

1. Se o tempo de computação for menor que o threshold inferior, aumenta a quantidade de tarefas para o worker, indicando que está subutilizado.

$$QtarefaW = ((\frac{Thresholdinferior}{Tcomp,i}) * TAM\_BLOCO) * FatorHet,i \quad (3)$$

2. Se o tempo de computação for maior que o threshold superior, diminui a quantidade de tarefas para o worker, indicando que está sobrecarregado.

$$QtarefaW = ((\frac{Thresholdsuperior)}{Tcomp,i}) * TAM\_BLOCO) * FatorHet,i \quad (4)$$

3. Se o tempo de computação estiver entre o threshold inferior e o superior, a quantidade de tarefas para o worker permanece a mesma que a anterior, indicando que está com o balanceamento de carga ideal.

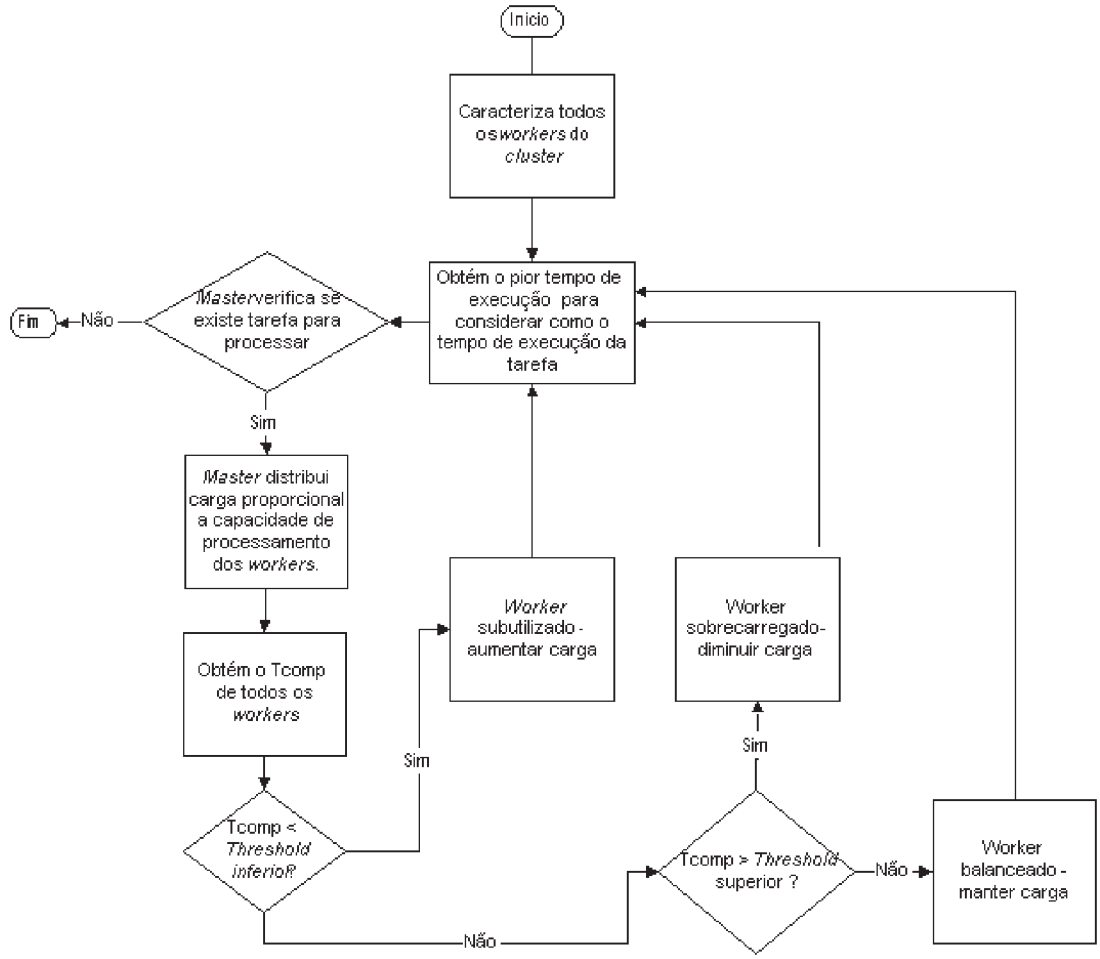
$$QtarefaW = TAM\_BLOCO * FatorHet,i \quad (5)$$

Onde $QtarefaW$ indica a quantidade de carga que será enviada para o worker; threshold inferior e threshold superior indicam os valores em ms dos threshold inferior e superior respectivamente; Tcomp,i representa o tempo de computo da interação anterior do worker $i$ e $FatorHet,i$ indica o fator de heterogeneidade do worker i. Em cada interação, o algoritmo de balanceamento de carga verifica qual é a possível quantidade de tarefas que os workers têm capacidade de processar.
Cada worker, após o término do processamento de cada tarefa, informa ao algoritmo

de balanceamento o load index - tempo de computação e tempo de comunicação. [7] afirma que um load index comum utilizado em muitos algoritmos paralelos é a diferença entre o tempo que o worker leva para iniciar e finalizar uma tarefa particular.

O fluxograma do algoritmo de balanceamento de carga é apresentado na Fig. 2.

Figura 2: Fluxograma do algoritmo de balanceamento de carga




As estratégias de distribuição de bloco existentes, não atendem aos requisitos do modelo proposto, por conta disso é necessária a proposta de outra estratégia de distribuição. O modelo proposto necessita que a carga enviada seja dinâmica e varie conforme performance do worker envolvido no processamento. A carga enviada pode ser diferente entre os workers e diferentes para o mesmo worker em interação diferente. A estratégia proposta é uma variação na estratégia de distribuição conhecida como Block- cyclic distribution. A Block-cyclic dinamic distribution, estratégia proposta exibida na figura 3, consiste em variar o tamanho do bloco que é utilizado na Block-cyclic distribution, nessa distribuição o bloco tem tamanho fixo e é enviado ciclicamente para os processo. Na Block-cyclic dinamic distribution o tamanho do bloco varia conforme a capacidade de

processamento do worker que é obtido pelo load index, Além de ser enviado ciclicamente também.

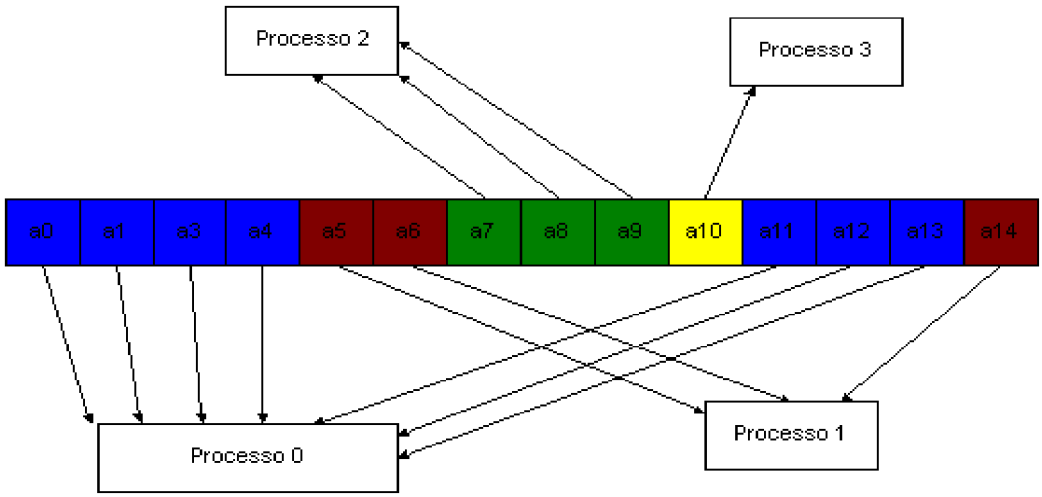



Figura 3: Block-Cyclic Dinamic Distribution de 15 blocos distribuídos entre 4 processos com tamanho de bloco variando conforme performance do processo.

Na Block-cyclic dinamic distribution, é possível que algum processo fique sem receber trabalho na última interação. A razão disso é porque o envio de dados para os processos ocorre de forma dinâmica e com a variação no tamanho da carga enviada. Porém acredita-se que os benefícios obtidos pela distribuição sobreponham esse fato.

O algoritmo de Medição tem a função de verificar se a estratégia de balanceamento está sendo eficiente e se os parâmetros informados no modelo estão apropriados para o ambiente, sugerindo resultados que permitam ajustar os parâmetros de maneira adequada. O algoritmo de Medição para efetuar essa verificação, obtém as informações sobre a execução do algoritmo de balanceamento que é registrado no repositório e da saída da aplicação paralela. Com essas informações pode-se inferir a eficiência dos parâmetros utilizados no modelo.

## 3 IMPLEMENTAÇÃO DO MODELO CONCEITUAL

A linguagem de programação utilizada foi o C/C++, a implementação do MPI utilizada foi o MPICH2 [8]. A principal razão para a utilização do MPICH2 é porque essa implementação separa os processos de comunicação e de gerenciamento, sendo o ambiente de execução padrão composto por um conjunto de daemons chamados de MPD - Managing Process Daemon que estabelece a comunicação entre as máquinas antes do processo da aplicação inicializar [5], essa característica permite facilmente a implementação do AWD. A aplicação paralela desenvolvida para ser utilizada no modelo foi a multiplicação de matriz quadrada. Considerando as matrizes quadradas A = (aij ) e B = (bij ) de ordem n, então C = (cij ) = AB é uma matriz quadrada de ordem n

e cij é obtido efetuando a multiplicação da linha ith de A pela coluna jth de B conforme a equação 6 .

$$cij = ai0b0j + ai1b1j + \ldots + ai,n-1bn-1,j \quad (6)$$

Apesar do algoritmo de Medição ser proposto no modelo, o mesmo não foi implementado, pois o propósito do mesmo é analisar os resultados gerados pelo modelo para sugerir parâmetros otimizados para o modelo, como por exemplo o tamanho do bloco inicial, o threshold inferior e superior. A implementação desse algoritmo pode- ria ser utilizando-se algumas técnicas como inteligência artificial, estatística, mineração de dados ou qualquer outra que pudesse efetuar alguma aferição. A Medição foi efetuada utilizando-se da observação dos valores obtidos durante as medições.

## 4. CARACTERIZAÇÃO E NORMALIZAÇÃO DO CLUSTER

O Cluster utilizado para obter os resultados da pesquisa foi composto por 6 máquinas, as quais são descritas na tabela 1.

Tabela 1: Caracterização do Cluster

| Nome | Configuração das máquinas |
|---|---|
| master | AMD Athlon 64 bits 2.210 Mhz 1 Gb RAM 5Gb HD |
| worker01 | Digital Prioris MX 6200, 256 MB RAM, 2 Proc. Pentium Pro200Mhz HD SCSI |
| worker02 | Digital Server 3000, 512 MB RAM, Pentium II 266Mhz SCSI |
| worker03 | Digital Prioris MX 6200, 256 MB RAM, 2 Proc. Pentium Pro 200Mhz SCSI |
| worker04 | AMD-K6 II 500 Mhz 256 Mb RAM HD IDE 4.3 Gb |
| worker05 | Pentium II 133 MHz 128 MB RAM HD IDE 4 GB |

Como pode ser visto na tabela 1 o cluster é heterogêneo, logo a capacidade de processamento entre as máquinas são diferentes. Por isso, para a análise da efficiency, não se pode considerar como a quantidade de processos envolvidos no cálculo a quantidade física de máquinas no cluster, é necessário normalizar a capacidade de processamento entre essas máquinas. Normalizar significa equiparar a capacidade de cada máquina em relação à mais potente. Para identificar a máquina mais potente executou-se o beackmark de multiplicação de matriz serial para uma matriz quadrada de 600 em todos os workers. Os tempos obtidos são listados na tabela 2.

Tabela 2: Tempos obtidos com *beackmark*

| Worker | Tempo obtido(s) |
|---|---|
| Worker01 | 29,448000 |
| Worker02 | 28,964000 |
| Worker03 | 29,450000 |
| Worker05 | 83,812000 |
| Worker04 | 111,390000 |

No beackmark o melhor tempo obtido foi do worker02, esse tempo é usado como dividendo nos outros tempos para obter-se o valor normalizado. A exemplo, o valor nor- malizado do worker04 é 0,260023 que corresponde a 28,964000 dividido por 111,390000.
A tabela 3 apresenta os valores normalizados do cluster.

Tabela 3: Normalização do Cluster

| Worker | Tempos beackmark(s) | Melhor tempo obtido(s) | Valor normalizado |
|---|---|---|---|
| Worker01 | 29,448000 | 28,964000 | 0,983564 |
| Worker02 | 28,964000 | 28,964000 | 1 |
| Worker03 | 29,450000 | 28,964000 | 0,983497 |
| Worker05 | 83,812000 | 28,964000 | 0,345583 |
| Worker04 | 111,390000 | 28,964000 | 0,260023 |

## 5. SPEEDUP DO ALGORITMO DE BALANCEAMENTO DE CARGA

Para a validação do modelo foi verificado o speedup do algoritmo sem a inclusão dinâmica de workers. A figura 4 mostra que o algoritmo de balanceamento de carga sem a inclusão dinâmica apresenta-se escalável pois, com o aumento de workers no cluster houve um aumento nos speedups. Acredita-se que pode-se obter speedups melhores ajustando o tamanho do bloco e o parâmetro do threshold através implementação do algoritmo de Medição.

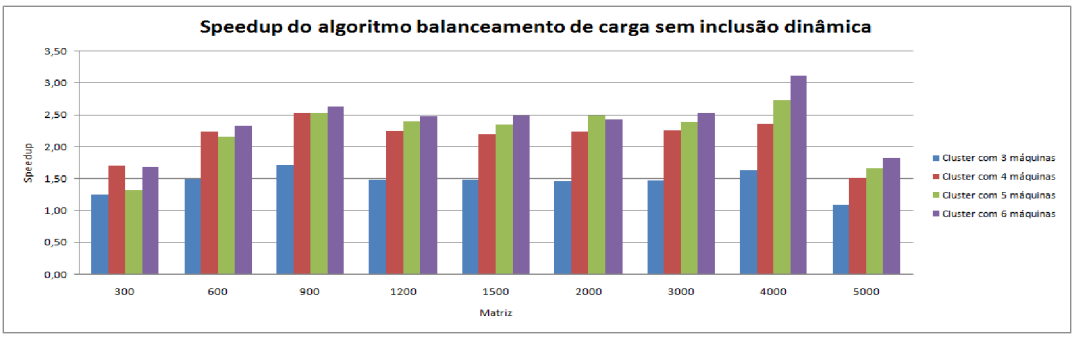



Figura 4. Speedup do algoritmo de balanceamento de carga sem inclusão dinâmica.

O algoritmo apresentou speedups crescentes com o aumento da quantidade de workers no cluster, com exceção das matrizes de 300, 600 e 2000. Na matriz de 300 o speedup para o cluster formado por cinco e seis máquinas foram inferiores a do cluster formado por quatro e na matriz de 600 o speedup para o cluster formado por cinco máquina foi inferior a do cluster formado por quatro. As razões para esse comportamento são:

1. Etapa da caracterização - na primeira interação o algoritmo precisa verificar a capacidade computacional de cada worker, para isso é enviado uma carga de trabalho igual para todos os workers, assim workers com poder

computacional diferentes recebem a mesma quantidade de tarefas. Essa operação tem impacto no tempo total de execução quando a matriz não é relativamente esparsa, como por exemplo, as de 300 e 600.

2. Heterogeneidade dos workers - a normalização indica o quanto um worker equivale ao worker mais potente cluster. Dependendo do índice de normalização do workers que fazem parte da caracterização o impacto será maior ou menor no tempo total de execução. Quanto maior for a heterogeneidade maior será o tempo de processamento do worker com menos poder computacional no momento da caracterização. Observa-se na própria matriz de 300 quando o cluster é formado por três e quatro máquinas que o speedup foi crescente, pois o índice de normalização para o worker03 que compõe o cluster formado por quatro máquinas é de 0,983497 sendo 1 e 0,983564 o índice do workers02 e worker01 respectiva- mente, dessa forma o poder computacional desses workers são bem próximos, diferentemente do worker05, cujo índice é de 0,345583.
3. Overhead - quanto maior a quantidade de worker presentes no cluster maior será o overhead o ocorrido nas interações do algoritmo para o envio, recepção e outros cálculos efetuados pelo algoritmo. Quanto mais esparsa for a matriz menos significante será a influência desses cálculos no tempo total de execução do algoritmo.

A outra verificação efetuada para a validação do modelo foi a execução do algoritmo de balanceamento de carga com a inclusão dinâmica. Na figura 5 pode-se verificar, com exceção da matriz de 300, que todos os speedups obtidos foram superiores a curva do cluster formado por 3 máquinas. Nessa configuração a inclusão dinâmica de um worker a partir da matriz de 300 já fornece tempos execuções melhores.

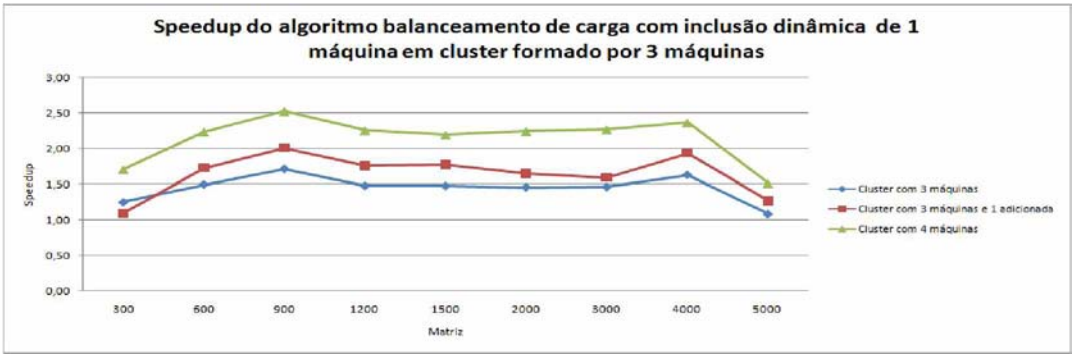



Figura 5: Speedup do algoritmo de balanceamento de carga com inclusão dinâmica de uma máquina em Cluster formado por três.

Na figura 6 pode-se verificar que os speedups obtidos com a inclusão dinâmica tem speedups crescentes a partir da matriz de 300, no entanto, esses speedups somente são superiores a curva do cluster formado por 3 máquinas depois da matriz de 2000. Nessa Medição houve speedup superior as curvas do cluster com quatro e cinco máquinas.

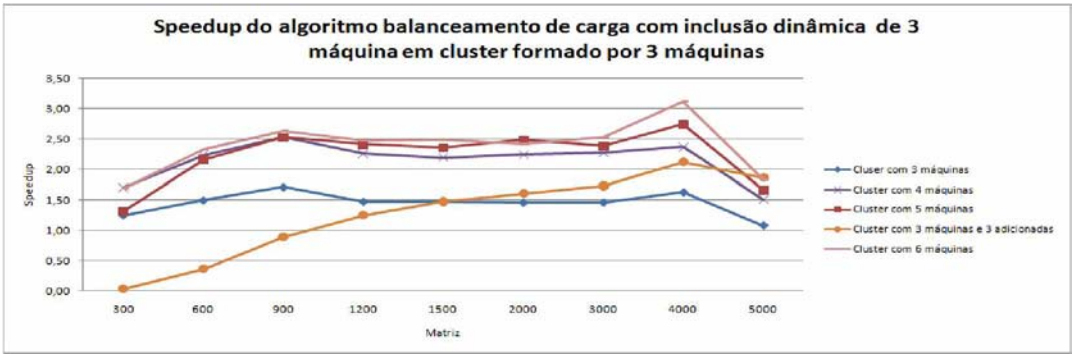



Figura 6: Speedup do algoritmo de balanceamento de carga com inclusão dinâmica de 3 máquinas em Cluster formado por três.

A figura 7 apresenta um gráfico com todos os speedups obtidos. Pode-se observar nas curvas que os piores tempos, foram obtidos quando três workers foram adicionado em um cluster formado por três máquinas. Aferiu-se quanto tempo era gasto para criar processos remotamente e verificou-se que esse tempo varia de máquina para máquina. Sendo assim, o tempo para criar um processo remoto no worker03 é menor do que criar no worker05 e criar no worker05 por sua vez, leva menos tempo do que o worker04. Por isso a quantidade de workers adicionados e qual worker foi adicionado, influencia diretamente no tempo de execução.

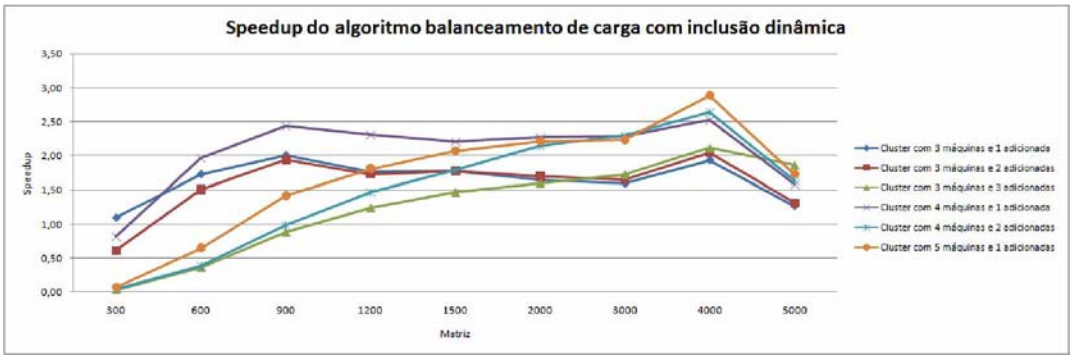



Figura 7: Speedup do algoritmo de balanceamento de carga com inclusão dinâmica.

Não se pode afirma que quanto maior a quantidade de workers a serem adiciona- dos menor será o speedup obtido, pois depende do custo da criação do processo remoto, o qual depende do worker a ser adicionado. Nas medições o decréscimo dos speedups foram observados com o aumento da quantidade de workers adicionados, porém isso ocorreu porque a ordem de inserção sempre foi worker03, worker05 e worker04, cujo custo de adição remota de processo aumenta. Outra evidência da influencia do tempo na criação de um processo remoto pode ser constatada nas curvas ”Cluster com 5 máquinas e 1 adicionada”e ”Cluster com 4 máquinas e 1 adicionada”. Os speedups apresentados por essa curva são melhores do que a outra, apesar de possuir menos máquinas, pois a máquina adicionada na primeira curva é o worker04 já na segunda é o worker05.

## 6 CONCLUSÃO

A pesquisa propôs, implementou e avaliou um modelo para balanceamento de carga em HNOW que contempla a inclusão dinâmica no cluster em aplicações master/worker. Em HNOW o principal fator desafiador é a heterogeneidade do ambiente. O modelo apresentado trata como fatores de heterogeneidade a capacidade de processamento de cada worker - enviando para cada worker um trabalho correspondente a sua capacidade; e a inclusão dinâmica de workers que é ajustar toda a carga do cluster conforme capacidade de processamento do worker adicionado. O algoritmo de balanceamento de carga foi avaliado com e sem a inclusão dinâmica de workers. Nas duas avaliações o algoritmo distribuiu a carga conforme capacidade de processamento dos workers próximo aos valores normalizados de cada worker, atingindo assim o seu propósito. Os speedups obtidos do algoritmo de balanceamento de carga sem a adição dinâmica quase em todas as medições foram maiores quando o cluster era formado por uma quantidade maior de workers. Isso indica que o algoritmo apresentou-se escalável sem a inclusão de novos workers. Os speedup obtidos do algoritmo de balanceamento de carga com a adição dinâmica também apresentou-se escalável, porem não em todas medições, indicando que a escalabilidade depende da capacidade, normalizada, do worker adicionado. Percebe-se que a razão para isso são os tempos gastos para a criação e gerenciamento remoto de processos. O trabalho constatou que é possível obter escalabilidade do cluster HNOW utilizando-se de criação dinâmica de processo para adicionar novos workers. Pretende-se, como trabalho futuro, ajustar e implementar o modelo em multi-cluster.

## REFERÊNCIAS

[1] Rajkumar Buyya. High Performance Cluster Computing: Architectures and Systems, volume 1.Prentice Hall PTR, New Jersey, USA, 1999.

[2] Márcia C. Cera, Guilherme P. Pezzi, Elton N. Mathias, Nicolas Maillard, and Philippe Olivier Alexandre Navaux. Improving the dynamic creation of processes in mpi-2. In Bernd Mohr, Jesper Larsson Traff, Joachim Worringen, and Jack Dongarra, editors, PVM/MPI, volume 4192 of Lecture Notes in Computer Science, pages 247–255. Springer, 2006.

[3] Mark A. Franklin and Vasudha Govindan. A general matrix iterative model for dynamic load balancing. Parallel Computing, 22(7):969–989, 1996.

[4] G. A. Geist, J. A. Kohla, and P. M. Papadopoulos. Pvm and mpi: A comparison of features. Calculateurs Paralleles, 8(2):137–150, 1996.

[5] William Gropp, Ewing Lusk, David Ashton, Darius Buntinas, Ralph Butler, Anthony Chan, Rob Ross, Rajeev Thakur, and Brian Toonen. Mpich2 user's guide. Technical report, Mathematical, Information, and Computational Sciences Division, Novem- ber 2005.

[6] Kaoutar El Maghraoui, Travis Desell, Boleslaw K. Szymanski, James D. Teresco, and Carlos A. Varela. Towards a middleware framework for dynamically reconfigurable scientific computing. In L. Grandinetti, editor, Grid Computing and New Frontiers of High Performance Processing, volume 14 of Advances in Parallel Computing, pages 275–301. Elsevier, 2005.

[7] Shahzad Malik. Dynamic load balancing in a network of workstations. Technical Report 219762, Carleton University, November 2000.

[8] Message Passing Interface Forum MPIF. Mpi-2: Extensions to the message-passing in- terface. Technical Report, University of Tennessee, Knoxville, 1996.

[9] Ashraf Osman. Designing a scalable dynamic load-balance algorithm for pipelined sin- gle program multiple data applications on a non-dedicated heterogeneous network of workstations. Phd thesis, West Virginia University, Morgantown, West Virginia, USA, 2003.

[10] Gary Shao, Rich Wolski, and Fran Berman. Performance effects of scheduling strategies for master/slave distributed applications. Technical Report CS98-598,

University of California, San Diego, 1998.

[11] Mohammed J Zaki, Wei Li, and Michal Cierniak. Performance impact of processor and memory heterogeneity in a network of machines. Technical Report 574, Department of Computer Science, University of Rochester, 1995.

[12] Mohammed Javeed Zaki, Wei Li, and Srinivasan Parthasarathy. Customized dynamic load balancing for a network of workstations. Journal of Parallel and Distributed Computing, 43(2):156–162, 1997.