Um Algoritmo de Escalonamento Distribuído Baseado em Grupos de Processadores

Celso Maciel da Costa
Pontificia Universidade Católica do Rio Grande do Sul
Programa de Pós-graduação em Ciência da Computação
Porto Alegre, RS, Brasil, 90619-900
celso@inf.pucrs.br

e

Getúlio Almeida dos Santos
Pontificia Universidade Católica do Rio Grande do Sul
Programa de Pós-graduação em Ciência da Computação
Porto Alegre, RS, Brasil, 90619-900
gsantos@inf.pucrs.br

Abstract

In parallel environment, the application has described like a set of cooperative units that are distributed among the processors of a parallel machine. The main goal of that is to minimize the execution time. To achieve that, it is important to balance of the workload in all components of the cluster. Algorithms that balance the compute load are called "scheduling algorithms". One factor, which makes performance degradation, is comunication time between processors, since they are necessary exchanges of messages to keep the load information of the nodes and to attribute a new process to a processor. This work presents a framework to support the development of scheduling algorithm. For this architecture, a distributed scheduling algorithm was developed, that is based in the concept of group of processors. Also a centralized algorithm was developed. The implementation and evaluation of performance of these algorithms was realized in the MDX, a parallel environment. The performance evaluation shows that the use of the concept of groups of processors is adjusted to the distributed scheduling algorithms, since the decision taking through of faster way and the load is distributed of more homogeneous way.

Keywords: Scheduling, Scheduling algorithm, Schedulling in parallel environment.

Resumo

Em um ambiente de programação paralela, a aplicação é descrita como um conjunto de processos distribuídos nos elementos de processamento da máquina paralela. O objetivo principal desta distribuição é a minimização do tempo de execução. Para esta tarefa, é fundamental que o ambiente paralelo garanta a distribuição uniforme da carga de trabalho gerada pela aplicação entre as unidades de execução. Algoritmos que realizam esta atividade são conceituados como algoritmos escalonadores. Sabe-se que um dos fatores que mais influenciam negativamente na performance dos algoritmos de escalonamento distribuídos é o tempo de comunicação entre os processadores, visto que são necessárias trocas de mensagens para manter as informações de carga dos nodos e para atribuir um novo processo a um processador. O presente artigo apresenta uma arquitetura de suporte à implementação de algoritmos de balanceamento de carga. Para esta arquitetura, foi desenvolvido um algoritmo de escalonamento distribuído, baseado no conceito de grupo de processadores. Foi também implementado um algoritmo centralizado, sendo que foram feitas a implementação e avaliação de desempenho destes algoritmos no ambiente MDX. As avaliações realizadas mostram que a utilização do conceito de grupos de processadores é adequada aos algoritmos de escalonamento distribuído, visto que a tomada de decisão é realizada de forma mais rápida e a carga é distribuída de forma mais homogênea.

Palabras claves: Escalonamento, Algoritmos de escalonamento, Escalonamento em sistemas paralelos.

1. Introdução

O principal objetivo dos algoritmos de escalonamento é a minimização do tempo de execução dos processos componentes das aplicações, com o uso eficiente do poder de processamento do ambiente de execução. Para isto, é necessário que a carga seja distribuída de maneira uniforme entre a rede de processadores, de maneira que não existam nodos sobrecarregados enquanto outros nós estejam ociosos ou subtilizados. Embora estejam sendo estudados já há vários anos, constitui-se ainda um grande desafio o projeto e a implementação de algoritmos eficientes que atinjam os objetivos pelo quais são propostos.

Sabe-se que um dos fatores que mais influenciam negativamento na performance dos algoritmos de escalonemto é o tempo de comunicação entre os processadores, visto que são necessárias trocas de mensagens para manter as informações de carga dos nodos e para atribuir um novo processo a um processador. O presente trabalho apresenta uma arquitetura de suporte à implementação de algoritmos de balanceamento de carga para a qual foi desenvolvido um algoritmo de escalonamento distribuído, baseado no conceito de grupo de processadores. Foi também implementado um algoritmo centralizado, sendo que foram feitas a implementação e avaliação de desempenho destes algoritmos no ambiente MDX. Tendo em vista a verificação da eficiência do uso co conceito de grupos de processadores, as avaliações realizadas tiveram por objetivo analizar os tempos de tomada de e a distribuição da carga entre os processadores.

2. Escalonamento Global

Uma aplicação paralela é classificada quanto ao número de processos como estática ou dinâmica. Em uma aplicação estática o número de processos é conhecido no início de sua execução e não se modifica até o término do processamento. O grafo de processos não se altera. Aplicações dinâmicas são aquelas que o número de processos podem variar durante a execução. No primeiro caso, os processos são alocados aos nós processadores da rede durante a inicialização e permanecem no mesmo nó até o final de sua execução. Não ocorre migração de processos e não são criados novos processos.

No segundo caso, a alocação dos processos criados dinamicamente deve ser feita de maneira a garantir um bom desempenho global do sistema (*throughput*). A carga de trabalho deve ser distribuída de uma forma balanceada para evitar-se que coexistam no sistema nós ociosos e nós sobrecarregados. Pode-se utilizar a migração de processos para obter este balanceamento de carga.

Em [1] é apresentada uma classificação dos algoritmos de escalonamento. De acordo com esta classificação, um algoritmo global pode ser estático ou dinâmico. Ele é estático quando o nó onde o processo será executado é definido antes da execução. Depois de realizada a alocação o processo permanece neste nó até o final de sua execução. Nos algoritmos dinâmicos a decisão do nó é feita à medida que os processos são criados. Para melhorar o balanceamento de carga, pode ocorrer a migração de processos.

Outro importante aspecto da classificação é relacionado à escolha do nó onde será executado o processo. Em algoritmos centralizados a informação sobre a carga dos nós é mantida em um nó do sistema, e a decisão de alocação também é centralizada. Em algoritmos distribuídos a informação sobre a carga dos nós é distribuída ao longo dos nós do sistema, a decisão de alocação ocorre também de forma distribuída. Quanto às decisões, elas podem ser cooperativas quando envolvem os outros nós do sistema ou não cooperativas quando não envolvem os outros nós.

Os algoritmos são constituídos de quatro elementos básicos, que implementam a política de informação, a política de transferência, a política de localização e a política de negociação. A política de informação é responsável pelo armazenamento dos dados relativos a carga dos nós. A política de transferência é responsável pelas tarefas de transferência entre nós. A política de localização determina em que nó do sistema o processo será criado ou para que nó o processo será migrado. A política de negociação representa a interação entre os nós durante o processo de decisão.

3. O Sistema MDX

O MDX [9] é um ambiente de programação paralela desenvolvido pelo grupo de processamento paralelo e distribuído do Programa de Pós-Graduação em Ciência da Computação da PUCRS. Este sistema possui uma arquitetura cliente/servidor sendo executado em um cluster de PC´s com o sistema operacional Linux. O MDX suporta dois modelos de programação: Troca de mensagens e memória compartilhada distribuída. No modelo baseado em troca de mensagens, uma *task* inicial dispara outras *tasks*, que serão criadas localmente ou à distância, e a comunicação ocorrerá através da troca de mensagens. No modelo baseado em memória compartilhada distribuída, um programa composto por várias *threads* é replicado nos nós componentes do cluster e a execução

começará na *thread* incial do primeiro nó. A criação das demais *threads* é feita localmente, ou remotamente e a comunicação entre as *threads* ocorrerá através de dados definidos como compartilhados, em uma memória global compartilhada.

O sistema é concebido como uma biblioteca de funções e suporta um conjunto de primitivas que permite aos programadores escrever programas paralelos. Diferentes tipos de primitivas disponíveis no ambiente permitem: criação dinâmica de *tasks* e *threads*, criação dinâmica de portas e barreiras e de variáveis compartilhadas.

No sistema MDX a criação de processos é suportada pela primitiva *LMDX_create_thread* que implementa funcionalidades de criação local e remota de *threads* e *tasks*. As Figuras 1 e 2 mostram a funcionalidade da arquitetura do escalonador em um modelo de programa onde a expressão do paralelismo é feita através de um programa *multithreaded*, a Figura 1 apresenta um exemplo de criação local da thread enquanto a Figura 2 exemplifica a criação remota. A Figura 1 mostra dois processos em execução em um nó do sistema, o processo de uma aplicação paralela (B) e o processo *mdx_kernel* com as *threads LB_loc* e *exec_manager* (A).

No processo do usuário pode-se imaginar inicialmente em execução somente a *thread main*. O programa paralelo quando precisa criar uma *thread* executa uma chamada de procedimento *Em_create_thread ("th01")* (1), esta solicitação é tratada pelo servidor *exec_manager* que faz uma chamada à *thread LB_loc*. Esta realiza uma pesquisa nas estruturas de dados locais do escalonador e identifica que a *thread* será criada localmente. Esta informação é retornada ao *exec_manager* (3) que atualiza suas estruturas de controle de *threads* e retorna uma requisição de criação de *threads* para o programa que cria e executa a *thread* "th01" (4).

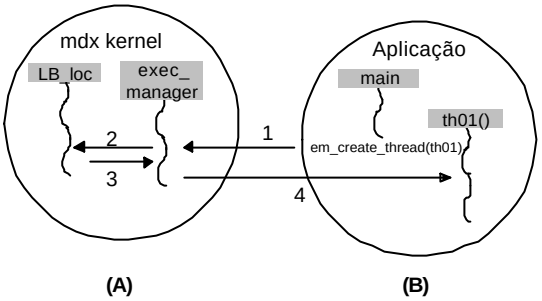

Figura 1. Criação local de *threads.*

A Figura 2 exemplifica a situação em que a criação da *thread* será remota ao nó solicitante. A figura mostra dois processos em execução em dois nós do sistema, o nó "A" onde é feita a requisição de criação de *thread* e o nó remoto "B". A aplicação quando precisa criar uma *thread* executa uma chamada de procedimento *em_create_thread ("th01")* (1), esta solicitação é tratada pelo servidor *exec_manager* que faz uma chamada `a *thread* LB_loc. Esta realiza uma pesquisa nas estruturas locais do escalonador e identifica que será criada a *thread* no nó remoto "B". Esta informação é retornada ao *exec_manager* (3) que atualiza suas estruturas de controle de *threads* e faz uma requisição de criação remota de *threads* para o *exec_manager* do nó "B" (4). O *exec_manager* do nó "B" executa então criação da *thread* "th01" (5)

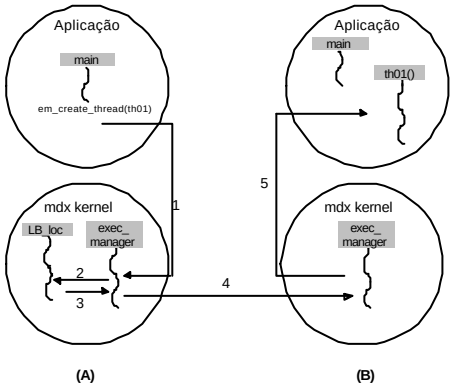

Figura 2. Criação remota de *threads.*

A criação de *task*s, no modelo baseado em troca de mensagens, ocorrerá de forma análoga. A diferença principal é que deverá ser carregado um código executável (o da *task*) para a memória, no nó onde será criada a *task*, o que não ocorre no modelo memória compartilhada distribuída, no qual o programa paralelo multi*thread*ed é replicado no

início da execução, em todos os processadores.

4. Algoritmos de balanceamento de Carga

Foram desenvolvidos dois algoritmos, um centralizado e outro distribuído. O algoritmo centralizado é um algoritmo global e dinâmico. Os nós do sistema informam periodicamente sua carga de trabalho a um nó centralizador. Todas as atividades de criação de processos são gerenciadas pelo servidor Gerente de Execução que consulta o nó centralizador para obter o nó com menor carga para a execução do processo. A funcionalidade do algoritmo está implementada pela política de informação e pela política de localização.

O propósito deste algoritmo é manter atualizadas as informações de carga de todo o sistema e utilizar estas informações sempre que ocorrer a criação de um processo ou uma *task*. É um algoritmo de implementação simples e eficiente.

Um índice de carga[1] expressa em termos quantitativos o estado de carga da máquina: Ociosa, carregada ou sobrecarregada. A definição de um índice de carga não é uma tarefa trivial [2] [3].

Em [4] é feito uma comparação da influência de diversos descritores como índices de carga nas decisões dos algoritmos de escalonamento. Utiliza-se nos algoritmos do MDX os seguintes descritores propostos e avaliados neste trabalho:

- Número de processos na *run queue;*
- CPU *context switch rate*;
- Percentual do tempo de CPU livre;
- Tamanho da memória livre;
- Média do número de processos na *run queue* no último minuto.

O índice de carga que deve ser utilizado no escalonador é definido como parâmetro de inicialização do sistema MDX. O escalonador utiliza o descritor "média do número de processos prontos para execução no último minuto" como o índice de carga *default* de execução. Vários autores utilizam este índice de forma similar [2] [5] [6] [10] [11] [12].

O Sistema Linux disponibiliza esta informação de carga através do vetor de carga média de CPU (loadavg). Neste vetor são armazenados três valores: O primeiro é o número médio de processos prontos para execução no último minuto, o segundo, o número médio de processos prontos para execução nos últimos cinco minutos e o último, o número médio de processos prontos para execução nos últimos quinze minutos. O vetor é atualizado por uma *thread* do *kernel* a cada cinco segundos, a média de um minuto é calculada, desta forma, com base em doze medições. Os outros descritores são obtidos através da leitura dos arquivos disponibilizados no *filesystem* "/proc" [7] [8].

A *thread LB_load* ciclicamente obtêm o descritor de carga do sistema operacional em uma freqüência definida pelo usuário na inicialização do Sistema MDX. O algoritmo utiliza um valor mínimo de índice de carga que define o limite em que as solicitações de criação de *thread* ou processos fase serão feitas localmente, este valor $C_{mín}$, é definido como parâmetro de inicialização do sistema.

A política de informação é responsável pela disseminação das informações de carga ao nó centralizador. As informações são transmitidas ao nó centralizador de duas maneiras: Ciclicamente, na freqüência f definida pelo usuário na inicialização do Sistema MDX; ou sempre que ocorre uma mudança na carga do nó. A freqüência f pode ser alterada para obter um melhor desempenho no algoritmo. Aplicações em que a criação de processos é muito freqüente, exigem, para uma informação de carga mais consistente, que a freqüência de envio de mensagens de atualização seja maior. Deve ser observado, no entanto, que o índice de carga utilizado pelo sistema é atualizado pelo *kernel* do *Linux* a cada cinco segundos. A implementação da política de informação é feita pela *thread LB_inf* que a cada ciclo definido pela freqüência f envia mensagens assíncronas de atualização de carga para o nó centralizador.

A política de localização é responsável pela definição do nó do Sistema no qual será executado o processo ou a *task*. Quando ocorre a criação de um processo o gerente de execução requisita à política de informação local o nó de criação. A política de localização compara o índice de carga local com o limite $C_{mín}$, sendo maior, requisita ao nó centralizador o endereço de execução. De outra forma, cria o processo ou a *thread* localmente.

A implementação da política de localização é feita pelas *threads LB_loc*. Uma *thread* é executada em todos os nós e

[1] Neste artigo, os termos carga e índice de carga são utilizados como sinônimos.

executa as funções de comunicação entre o gerente de execução e a *thread LB_loc* do nó centralizador. Esta *thread*, quando recebe uma requisição, percorre as estruturas de dados que mantém as informações de carga dos nós em busca do endereço do nó com menor carga. Esta informação e devolvida ao nó que requisitou a informação.

4.1 Algoritmo distribuído

O algoritmo distribuído é um algoritmo de escalonamento global, dinâmico e cooperativo. Os nós do sistema são agrupados em uma estrutura hierárquica com o objetivo de reduzir o número de mensagens que devem ser enviadas entre os nós. A hierarquia possibilita que o sistema possua informações de todos os nós e que utilize esta informação de maneira cooperativa para efetuar o balanceamento de carga de todo o ambiente.

Na inicialização do sistema são formados grupos que possuem um nó líder. A quantidade de nós que formam o grupo é informada como parâmetro de configuração na inicialização do MDX. Os nós possuem informação da carga de todos os nós do grupo e utilizam esta informação para a criação local ou remota de *threads*. Os nós líderes também formam grupos de nós que trocam entre si a informação de carga de seus grupos. O algoritmo utiliza dois valores de carga que definem os limites de cada fase, os valores C_{min} e C_{max}.. O índice C_{min} é o limite entre as fases local e intragrupo. O índice C_{max} é o limite entre as fases intragrupo e entre grupos. Os valores de C_{min} e C_{max} são definidos como parâmetros na inicialização do sistema MDX.

A política de informação é responsável pela disseminação das informações de carga aos outros nós do grupo. É a mesma política de informação utilizada no algoritmo Centralizado, descrita anteriormente. Uma funcionalidade adicional é a transmissão de informações sobre os grupos a todos os nós líderes. Também se utiliza nesta transmissão, o ciclo definido pela freqüência *f*.

A política de localização determina o nó do Sistema no qual será criado a task ou a thread. De maneira análoga ao algoritmo centralizado, a política de localização é implementada pela thread *LB_loc*.

Em função das informações de carga do sistema são definidos três níveis de escalonamento que são:

- Escalonamento local;
- Escalonamento intragrupo;
- Escalonamento entre grupos.

O Escalonamento local é simplesmente a criação local das *threads* que o programa de aplicação em execução solicita, ocorre quando a carga do nó estiver abaixo do limite definido por C_{min}.

O escalonamento intragrupo ocorre quando a carga do nó estiver entre os limites C_{min} e C_{max} . Neste caso, é selecionado um nó considerando-se somente os nós do grupo. É feita uma pesquisa na tabela local de cargas em busca do nó com menor carga.

O escalonamento entre grupos ocorre quando a carga do nó for maior que C_{max}. A cada solicitação de criação de uma nova *thread* ou *task*, será selecionado o nó com menor carga considerando-se os demais grupos existentes.

4.1.1 Escalonamento local

Cada nó mantém as informações de carga de todos os nós de seu grupo e também os valores de C_{min} e C_{max}. Estas informações são atualizadas pela thread *LB_inf* responsável pela implementação da política de informação. A thread *LB_loc* consulta as estruturas de dados locais e, se o valor da carga do nó for menor que C_{min} a criação será local e o escalonamento é local. Caso contrário, o escalonamento deverá ser verificado como possível intragrupo.

4.1.2Escalonamento intragrupo

A *thread LB_loc* consulta as estruturas de dados locais, se o valor da carga do nó estiver entre C_{min} e C_{max}, a criação será intragrupo, isto é, no nó do grupo que possuir a menor carga. Deve-se salientar que todos os nós do grupo contêm uma tabela local que armazena a carga atual de cada nó e também os valores de C_{min} e C_{max}. As *threads LB_inf* com a freqüência *f* atualizam as informações de suas cargas enviando uma mensagem contendo sua carga para cada um dos nós que fazem parte do grupo.

4.1.3 Escalonamento entre grupos

Quando uma aplicação em execução solicita a criação de uma thread e o valor da carga do nó for maior que C_{max} o escalonamento será feito considerando os outros grupos do sistema.

Para possibilitar a verificação mais rápida, o líder do grupo mantém as cargas de todos os nós de seu grupo e também uma entrada para cada grupo vizinho. Nesta entrada é armazenada a menor carga entre os nós do grupo deste líder. Pode-se observar esta situação na Figura 3 onde o nó D, líder do grupo 1, mantém a carga dos nós A,B,C

e D, pertencentes ao seu grupo e a carga dos nós F e H, seus vizinhos.

Um programa em execução solicita a criação de um *thread*, o algoritmo realizado pelo escalonador é o seguinte:

? Solicita ao nó líder do grupo, qual o nó dos grupos vizinhos que têm a menor carga;

? A informação do nó recebida pelo líder é repassada ao Gerenciador de Execução;

? O gerenciador de Execução solicita ao nó destino a criação remota da *thread*.

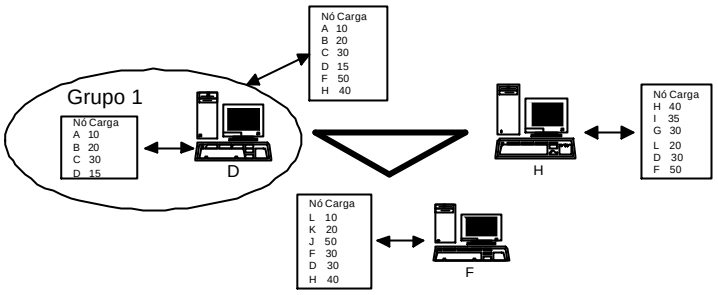

Figura 3. Escalonamento entre-grupos.

5. Avaliação de desempenho

Foram avaliados a sobrecarga de processamento ocasionada pelo processo de decisão do escalonador e o índice de carga resultante em cada nó do sistema. Foram ainda comparados os desempenhos dos algoritmos centralizado e distribuído, através de medições realizadas com quatro e seis máquinas. Nestas medições foram utilizados, de forma exclusiva, computadores pessoais com capacidades homogêneas. As máquinas são *Pentium Celeron* – 300 MHz, com cache L2 de 128 kbytes e memória de 64 Mbytes. As máquinas estão conectadas através de uma rede padrão *Ethernet* de 10Mbits. O sistema operacional é o Linux (*kernel* 2.2.19). Os resultados foram obtidos através da execução de um programa paralelo SPMD, desenvolvida especialmente para esta medição. O escalonador foi alterado para criar um arquivo com o histórico das decisões.

5.1 Programa de testes

O programa que se utilizou nas medições foi desenvolvido especialmente para esta avaliação. Os aspectos que nortearam a criação do programa foram os seguintes:

? Os algoritmos de escalonamento são dinâmicos, a aplicação tem que produzir dinamicamente um número considerável de *threads* para se avaliar os seus desempenhos;

? As decisões dos algoritmos são tomadas de forma cooperada e cada decisão deve considerar o estado global do sistema.

? O programa deve produzir *threads* dinamicamente em todos os nós do sistema originando decisões distribuídas ao longo destes mesmos nós.

O programa implementado cria de forma randômica, três tipos de *threads*: uma *thread* CPU-bound (*T_loop*), uma *thread wait* (*T_sleep*) e uma *thread* criadora (*T_create*). A *thread T_loop* é um loop infinito que tem por objetivo aumentar a carga de processamento da máquina. A *thread T_sleep* tem por objetivo apenas de aumentar o número de *threads* em execução sem, no entanto aumentar a carga de processamento e finalmente, a *thread T_create* tem o objetivo de criar outras instâncias dela mesma ou das outras duas *threads*. O programa possui dois parâmetros: O primeiro é a quantidade de *threads T_create* iniciais que devem ser criadas, e o segundo, é a freqüência de criação destas *threads T_create* iniciais. Seguindo o modelo proposto para execução SPMD, o programa acima é replicado em todos os nós do sistema. O código inicial *main* do programa é executado somente no nó onde está em execução o Servidor de Nomes do MDX. Nos demais nós o processo se registra no MDX e fica bloqueado até o recebimento de uma mensagem para execução de um serviço. Ao receber a mensagem, o serviço solicitado corresponde à criação de uma *thread* que pode ser 1,2 ou 3 (*threads T_create*,*T_sleep* e *T_loop*).

5.2 Medição da sobrecarga do escalonador

O objetivo desta avaliação é comparar o tempo gasto pelo escalonador para definir o melhor nó destino para a criação de uma *thread* ou processo, considerando as implementações dos algoritmos centralizado e distribuído.

5.2.1 Execução com quatro nós.

A primeira medição foi realizada com 5 execuções do programa com um número inicial de *thread T_create*=100 e freqüência de criação *f*=2 segundos. O algoritmo utilizado foi o Centralizado. O número total de máquinas utilizado foi quatro.

A Segunda medição foi realizada com os mesmos parâmetros para o programa e foi selecionado o algoritmo Distribuído. O número de máquinas foi o mesmo do Centralizado, os grupos foram formados com dois nós. Os totais de *threads* nas medições dos dois algoritmos foram praticamente os mesmos em todas as dez execuções. Isto significa que o programa teve o comportamento semelhante no que se refere à produção dinâmica das *threads* e que o número de requisições aos algoritmos escalonadores é igual. A Figura 4 mostra um gráfico comparativo dos totais.

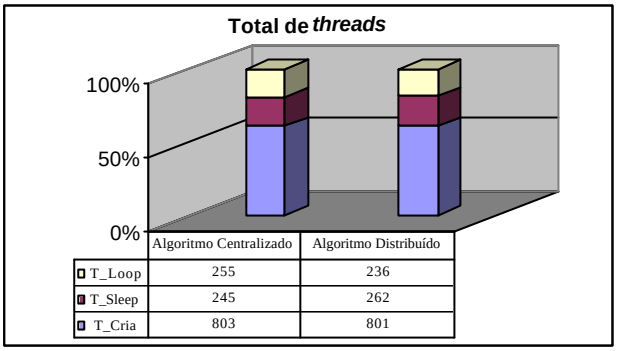

Figura 4. Gráfico comparativo dos totais de *threads* na execução com 4 nós.

As requisições do programa foram tratadas de duas formas pelo algoritmo Centralizado: Escalonamento local, é quando a carga do nó está abaixo do índice mínimo (seção 4.1.1) e Escalonamento remoto, quando é necessária uma pesquisa ao nó que mantém as informações de carga. Os totais obtidos nas medições podem ser visualizados na Tabela 1.

Tabela 1. Tempos do escalonamento Centralizado com 4 nós.

Tipo de escalonamento	Número de Requisições	Tempo (?sec)	Média (?sec)
Local	306	1718	5,40
Remoto	997	541003	757,70
Total	1303	542721	573,50

Pode-se observar na Tabela 1 que a média do tempo de requisição remoto é 140 vezes maior que a média do tempo local, o tempo de pesquisa remota é composto do tempo de envio da requisição, tratamento da requisição pelo servidor centralizado e mais o tempo de retorno com o endereço do nó destino. A sobrecarga média imposta à aplicação para cada *thread* criada é na ordem de 573,50 ?seg.

No algoritmo distribuído as requisições são tratadas de acordo com as fases do escalonamento: Fase local, quando é feita uma pesquisa local ao escalonador somente para verificar que fase se encontra o algoritmo; a fase intragrupo onde é feita uma pesquisa às estruturas de dados locais em busca do nó destino e finalmente a fase entre grupos, na qual é feita uma consulta ao líder do grupo para obtenção do nó destino, considerando o índice de carga nos outros grupos. Os totais obtidos nas medições com o algoritmo Distribuído podem ser visualizados na Tabela 2.

Tabela 2. Tempos do escalonamento Distribuído com 4 nós.

Tipo de escalonamento	Número de Requisições	Tempo (?sec)	Média (?sec)
Local	459	2566	5,59
Intragrupo	208	1955	9,40
Entre-grupo	634	168301	265,46
Total	1301	172822	132,84

Pode-se observar na Tabela 2 que o tempo gasto intragrupo é aproximadamente 1,5 vez maior que o tempo da fase local. A fase entre grupos, por sua vez, despende mais tempo para ser executada, já que envolve a comunicação com o líder do grupo. A relação entre as duas médias é aproximadamente 28 vezes. A sobrecarga média imposta a uma

aplicação para cada *thread* criada é na ordem de 132,84 ?seg.

A Figura 5 mostra um gráfico comparativo da sobrecarga imposta à aplicação considerando os resultados obtidos. Os valores do tipo de escalonamento local e intragrupo do modelo Distribuído foram agrupados como Local para facilitar a comparação entre os dois algoritmos. Isto reflete o fato de que cada membro do grupo possui uma Tabela com a carga de todos os componentes do grupo, sendo, portanto local a seleção do nó que irá executar a *thread*.

Não foi considerado o tempo de criação remota de *thread*.

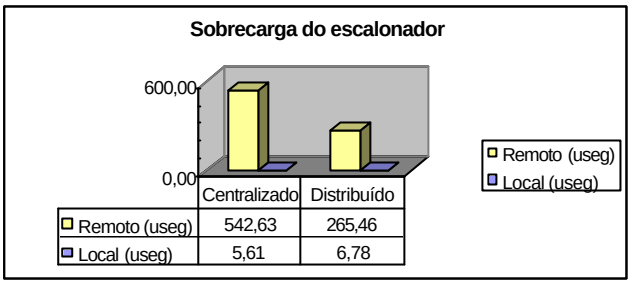

Figura 5. Sobrecarga média do escalonador com 4 nós.

5.2.2 Execução com seis nós

Foram realizadas duas medições com cinco execuções do programa com um número inicial de *thread T_create*=100 e freqüência de criação *f*=2. A primeira medição foi para algoritmo Centralizado e a segunda, para o algoritmo Distribuído com grupos formados por três nós. A Figura 6 apresenta um gráfico com os totais acumulados do número de *threads* geradas durante as duas medições.

Nas execuções com seis máquinas, a produção de *threads* no processamento do algoritmo distribuído foi maior. Esta diferença, no entanto, não produziu um número muito diferente de *threads T_loop* nas duas medições, sendo que estas *threads* têm influência direta nos testes por produzirem um aumento de carga.

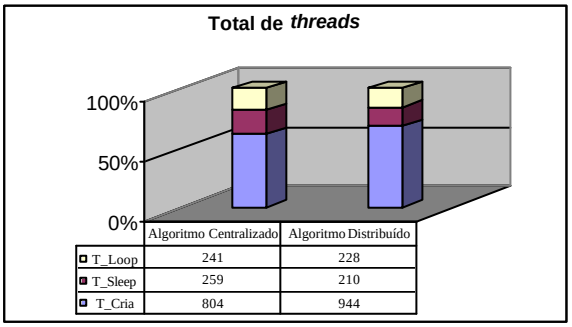

Figura 6. Totais de *threads* geradas na execução com seis nós.

De maneira análoga à execução com quatro nós, foram medidas o tempo gasto com o escalonamento nos dois algoritmos. A Tabela 3 mostra os resultados das medições com o algoritmo centralizado.

Tabela 3. Tempos do escalonamento Centralizado com seis nós.

Tipo de escalonamento	Número de Requisições	Tempo (?sec)	Média (?sec)
Local	481	2526	5,25
Remoto	804	804285	1000,35
Total	1285	806811	627,87

Pode-se observar na Tabela 3 que o tempo médio de requisição remoto é 190 vezes maior que o tempo médio local, confirmando a tendência natural da sobrecarga no nó centralizador aumentar devido ao incremento de novos nós. A sobrecarga média imposta ao programa para cada *thread* criada é na ordem de 627,87 ?seg. Este valor também é maior que na execução com quatro nós.

Tabela 4. Tempos do escalonamento Distribuído com seis nós.

Tipo de Escalonamento	Número de Requisições	Tempo (?sec)	Média (?sec)
Local	445	2472	5,56
Intragrupo	264	8510	32,23
Entre-grupo	402	81886	203,70
Total	1111	92868	83,59

Observa-se na Tabela 4 a sobrecarga média imposta à aplicação para cada *thread* criada é na ordem de 83,59 ?seg. A sobrecarga do algoritmo distribuído com seis nós é inferior ao centralizado da mesma forma que o desempenho com quatro nós.

A seguir a Figura 7 mostra um gráfico comparativo da sobrecarga imposta à aplicação considerando os resultados obtidos de todas as medições. Os valores do tipo de escalonamento local e intragrupo do modelo Distribuído foram agrupados como local para facilitar a comparação entre os dois algoritmos.

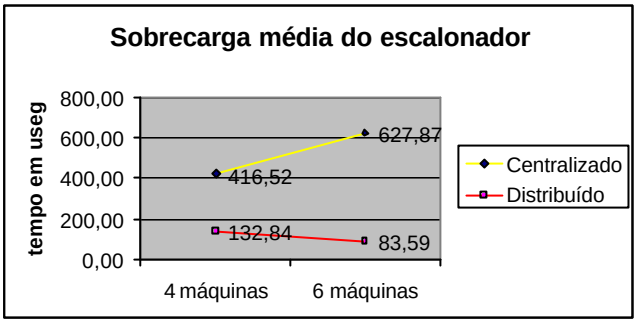

Figura 7. Sobrecarga média do escalonador com 6 nós.

Ao se fazer uma análise comparativa entre os dois algoritmos, observa-se que a sobrecarga média imposta pelo algoritmo Distribuído é aproximadamente 13,31% da sobrecarga imposta pelo Centralizado na execução com quatro nós e aproximadamente 31,89% na execução com seis nós. Diversos fatores contribuem para estes resultados:

- No algoritmo Centralizado qualquer necessidade de informação a respeito de outros nós é satisfeita através de uma requisição remota de informação feita para o servidor Centralizado. Isto sobrecarrega a comunicação ente os nós e o servidor centralizado que deve tratar um número grande de requisições.
- No algoritmo distribuído, as requisições de informação sobre a carga de outros nós é dividida em dois tipos de requisição: A intragrupo que é resolvida localmente, com um acesso à estrutura local de informação de carga e a entre-grupo que é mais demorada que a anterior, mas mais rápida que a do algoritmo Centralizado, por que é feita com o líder do grupo.
- O líder de um grupo no algoritmo Distribuído tem uma função análoga ao do servidor Centralizador no outro algoritmo, com a diferença do primeiro tratar a metade das requisições do segundo, já que o grupo possui 2 elementos na medição com quatro máquinas e 3 na outra medição.

5.3 Distribuição de carga

Para a análise de distribuição de carga, utilizou-se a execução com seis nós. A Tabela 5 mostra os índices de carga em cada nó ao final da execução no algoritmo Distribuído.

Tabela 5. Valores dos índices de carga no algoritmo Distribuído.

Máquinas / Medição	Nó 1	Nó 2	Nó 3	Nó 4	Nó 5	Nó 6	Carga Final Acumulada	Total *T_loop*	Carga Ótima
1.a medição	6,99	6,00	7,99	9,99	11,99	8,24	51,20	50	8,53
2.a medição	8,99	10,05	5,99	5,99	5,15	8,72	44,89	44	7,48
3.a medição	6,99	10,00	7,92	11,79	7,99	8,00	52,69	50	8,78
4.a medição	7,00	9,02	8,27	5,99	10,99	13,00	54,27	52	9,05
5.a medição	10,00	7,04	8,51	6,99	9,99	11,08	53,61	51	8,94

A Tabela 5 contém os valores do índice de carga ao final da execução em cada nó. A coluna carga final acumulada é a soma das cargas dos nós. A coluna Total *T_loop* é o total de *thread T_loop* criadas em todos os nós em cada medição. Pode-se observar que o valor final em todas as medições é aproximadamente o mesmo da coluna Total *T_loop*. A coluna Carga Ótima é o quociente entre a carga final acumulada e o número de nós.

Observa-se que a distribuição não é uniforme ao longo dos nós, existem alguns picos de carga. Atribui-se este comportamento, à inconsistência da informação de carga no momento da decisão tomada pelo algoritmo.

A Figura 8 mostra a comparação entre a distribuição de carga ótima e as efetuadas pelo algoritmo, estratificadas por medição. Observa-se que o pior caso ocorreu na quarta medição, onde a diferença foi de 44%.

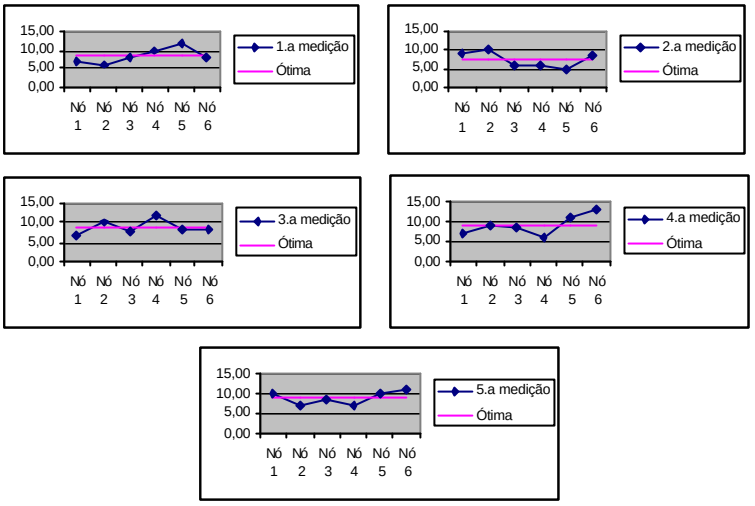

Figura 8. Distribuição de carga no algoritmo Distribuído.

A Tabela 6 mostra os índices de carga em cada nó ao final da execução no algoritmo Centralizado. Da mesma forma que a Tabela 5, a Tabela 6 contém os valores do índice de carga, carga final acumulada, total de *thread T_loop* e a carga ótima. Pode-se observar também na execução do algoritmo centralizado, uma aproximação dos valores Total *T_loop* e a carga final acumulada.

Tabela 6. Valores dos índices de carga no algoritmo Centralizado.

Máquinas / Medição	Nó 1	Nó 2	Nó 3	Nó 4	Nó 5	Nó 6	Carga Final Acumulada	Total *T_loop*	Carga Ótima
1.a medição	8,00	8,99	8,99	9,99	9,00	7,01	51,98	50	8,66
2.a medição	5,99	8,00	12,03	5,99	7,99	7,89	47,89	47	7,98
3.a medição	6,00	9,00	8,04	7,99	10,10	8,99	50,12	49	8,35
4.a medição	5,99	10,99	10,00	7,99	6,35	8,03	49,35	48	8,23
5.a medição	6,99	9,00	9,02	8,00	6,05	8,09	47,15	46	7,86

A Figura 9 mostra a comparação entre a distribuição de carga ótima à efetuada pelo algoritmo.

Pode-se observar através das Figuras 8 e 9 que no algoritmo centralizado a distribuição de carga foi realizada de forma mais homogênea que no algoritmo distribuído. Os valores de carga em cada nó estão mais próximos da carga ideal. Observa-se que o pior caso ocorreu na segunda medição, onde a diferença foi de 51%.

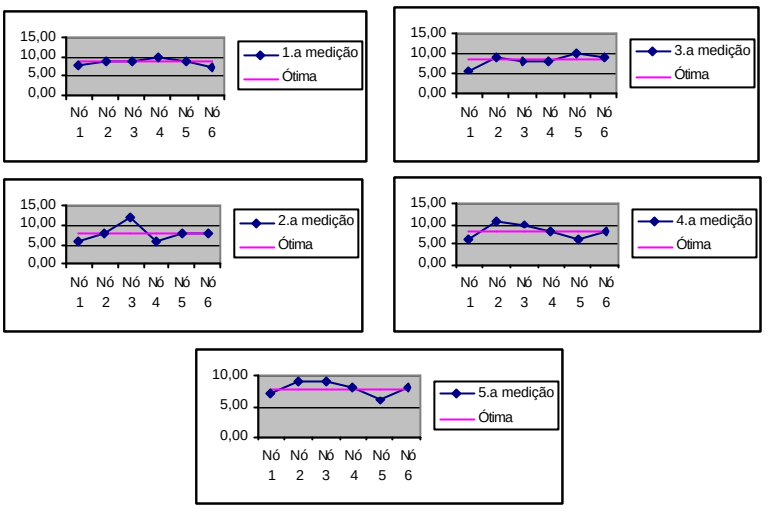

Figura 9. Distribuição de carga no algoritmo Centralizado.

Atribui-se este comportamento à distorção de valores de índice de carga que a política de informação mantém armazenada nas estruturas de dados e o índice de carga real do nó. Esta distorção é encontrada nos dois algoritmos devido a frequência de atualização desta informação. No algoritmo distribuído esta distorção é ainda maior, pois existem dois níveis de atualização de informação: a atualização intragrupo em que todos os nós difundem suas cargas, e a atualização entre-grupos é realizada entre os líderes de grupo.

6. Conclusão

Este artigo apresentou a concepção e implementação de dois algoritmos de balanceamento de carga no MDX. Os algoritmos são dinâmicos, globais e cooperativos. O primeiro algoritmo é um algoritmo centralizado desenvolvido utilizando um enfoque de simplicidade e eficiência. O segundo, é um algoritmo distribuído, que foi desenvolvido com o objetivo de obter uma maior eficácia no balanceamento de carga. A complexidade no modelo distribuído é maior que o modelo centralizado. Os algoritmos são configuráveis com relação a um considerável número de variáveis que definem seus comportamentos, possibilitando uma grande flexibilidade nas suas utilizações.

As definições dos algoritmos foram feitas a partir de heurísticas de um bom desempenho do escalonador, tais como: o número de mensagens entre os nós deve ser reduzido, as informações de carga devem ser atualizadas, a informação deve ser disponibilizada de alguma forma para todos os nós do ambiente e as decisões do escalonador devem ser executadas em poucos ciclos de CPU.

Os algoritmos desenvolvidos foram comparados entre si sob dois enfoques: o primeiro enfoque foi dado ao tempo médio gasto nas decisões de escalonamento. O segundo enfoque foi avaliar como foi feita a distribuição de carga entre os nós do sistema durante a execução do programa de testes.

Com relação ao tempo médio consumido em uma decisão, o algoritmo centralizado é mais lento que o distribuído. Isto é justificado pela própria estratégia de implementação utilizada, isto é, no centralizado um único nó atende a todas as requisições dos demais nós do sistema, caracterizando um ponto de sobrecarga no sistema. A distribuição de carga ao longo dos nós é feita de forma homogênea nos dois modelos, no modelo centralizado observa-se que as distribuições realizadas são mais uniformes. Isto é devido ao modelo centralizado armazenar de forma mais consistente que o distribuído as informações de carga de todo o sistema. O pior caso de distribuição ocorreu no algoritmo centralizado.

Pode-se considerar que o algoritmo distribuído é melhor que o algoritmo centralizado já que suas decisões são mais rápidas e a distribuição de carga ao longo dos nós é homogênea, o que comprova que o uso do conceito de grupo de processadores é adequado aos algoritmos de escalonamento distribuídos.

Pode-se sugerir outras avaliações para os algoritmos: avaliação das decisões considerando várias configurações de índices de carga e freqüência de atualização; avaliação de distribuição de carga considerando limites de carga diferentes dos propostos e utilizados nas medições deste trabalho; avaliação de s*peedup* de uma aplicação irregular quanto ao número de *threads* criadas.

Referências

[1] CASAVANT, T.L., KHUL, J.G. A taxonomy of schedulling in general-purpose distributed computing systems. IEEE Trans. Software Eng., v.l, SE-14, n.2, p. 141-154, fev. 1988.

[2] YAMIN, Adenauer C.; Um Ambiente Para Exploração de Paralelismo na Programação em Lógica. Porto Alegre: CPGCC da UFRGS, 1994. 204p. Dissertação de Mestrado.

[3] A.S. Tanenbaum , "Distributed Operating Systems", New Jersey: Prentice-Hall, Inc., 1995.

[4] KUNZ, T. The influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme, IEEE Trans. Software Eng., p.725-730, julho de 1991.

[5] NOGUEIRA, Mauro Lúcio B. Robin Hood: Um Ambiente para a Avaliação de Políticas de Balanceamento de Carga. Porto Alegre, CPGCC-UFRGS, 1997. 128p. (Dissertação de Mestrado).

[6] COSTA, Cristiano A. Uma Proposta de Escalonamento Distribuído para exploração do Paralelismo na Programação em Lógica. Porto Alegre: CPGCC-UFRGS, 1998. 104 p. Dissertação de Mestrado.

[7] MAXWELL, S. Linux Core Kernel Commentary, Scottsdale: Coriolis Group , 1999,575p.

[8] BECK, M. et al , Linux Kernel Internals. Addison-Wesley Longman. 1998, 480p.

[9] E. Preuss, "MDX: Um ambiente de programação paralela baseado em memória virtual distribuída", Dissertação de Mestrado, PUCRS, Porto Alegre, 1998

[10] ARAÚJO, A.P.F.; SANTANA, M.J.; SANTANA, R.H.C.; SOUZA, P.S.L. DPWP - *A New Load Balancing Algorithm, 5th International Conference on Information Systems Analysis and Synthesis* - ISAS'99, Orlando, U.S.A., 31 de julho a 4 de agosto de 1999.

[11] ARAÚJO, A.P.F.; SANTANA, M.J.; SANTANA, R.H.C.; SOUZA, P.S.L. *A New Dynamical Scheduling Algorithm, International Conference on Parallel and Distributed Processing Techniques and Applications* - PDPTA'99, Las Vegas, Nevada, U.S.A., 28 de junho a 1 de julho de 1999.

[12] SOUZA, P. S.; SANTANA, M. J.; SANTANA, R. H. C. *AMIGO – A Dynamical Flexible Scheduling Environment, Proceedings of the 5th International Conference on Information Systmas, Analysis and Synthesis: ISAS'99*, Orlando, Florida, junho de 1999.