Um Escalonador Distribuído para Programas Paralelos numa Rede de Workstations

Gustavo S. Rímolo* Eduardo F. Loures*
Virgílio A. F. Almeida† José N. C. Árabe†

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-010 Belo Horizonte, MG, Brasil
e-mail: virgilio@dcc.ufmg.br

Resumo

Com o aumento da demanda computacional nas redes de trabalho, torna-se cada vez mais indispensável o bom uso dos recursos disponíveis. Porém, em sistemas distribuídos quase sempre se verifica que alguns recursos são usados de uma forma ineficiente, visto que cada usuário utiliza um nodo do sistema de uma forma independente, sem se preocupar com o sistema como um todo. O objetivo desse trabalho é desenvolver um método onde os usuários possam utilizar todo o sistema de uma forma mais racional, a partir de uma melhor distribuição de tarefas entre os nodos do sistema. Esse método consiste da criação de um escalonador de tarefas que atuará sobre o sistema verificando qual a máquina que fornecerá melhores resultados no instante da criação de um processo. Assim, a carga total do sistema será distribuída de uma forma mais homogênea entre os nodos que o formam.

Abstract

With the increasing computational demand on networks of workstations, we have a greater need for better utilization of the resources. However, in distributed systems, we see that servers are used inefficiently, since each user requests nodes of the system independently, without regard for the system as a whole. This work shows a method which achieves a better distribution of tasks among the nodes of the system. The method is implemented by a scheduler that keeps track of the load of the nodes and dispatches a new process to the more appropriate machine. By doing that, the overall load is homogeneously distributed among the nodes of the system, leading to a better performance.

*Parcialmente financiado pelo CNPq
†Parcialmente financiado pela FAPEMIG (contrato TEC-1113/90)

1 Introdução

Redes de *workstations* têm sido consideradas como uma alternativa aos supercomputadores para a execução de aplicações computacionalmente intensivas. A idéia básica é a utilização de várias *workstations* na execução simultânea de *tasks* que compõem um programa paralelo. Um dos problemas dessa proposta está no fato de que as redes de *workstations* são compartilhadas por vários usuários, com diferentes demandas computacionais. Algumas estações podem estar livres enquanto outras se encontram sobrecarregadas. Se a execução paralela de um programa inadvertidamente for alocada a *workstations* com alta utilização, o tempo de execução pode vir a ser muito longo.

Este artigo apresenta um escalonador distribuído que tem como objetivo otimizar o tempo de execução de programas paralelos numa rede de *workstations* compartilhada por varios usuários. Vários experimentos foram realizados com uma carga de trabalho real composta por diferentes tipos de programas paralelos. Em quase todas as situações foram obtidos ganhos com o uso do escalonador distribuído. A implementação desse escalonador foi feita sobre o PVM (*Parallel Virtual Machine*), que é o ambiente utilizado para a execução de programas paralelos.

O artigo começa com uma descrição geral do escalonador. A seção 3 apresenta as políticas de trabalho. A seção 4 descreve as aplicações paralelas que formam a carga de trabalho utilizada para avaliar o sistema proposto. A seção 5 apresenta o ambiente usado para a implementação e teste do escalonador distribuído. Os experimentos e resultados obtidos são discutidos na seção 6. Finalmente, a seção 7 conclui o artigo apresentando um balanço dos resultados alcançados e sugerindo pesquisa futuras.

2 Descrição do Escalonador Distribuído

Conceitualmente, o sistema distribuído em questão consiste de nodos processadores (*workstations* e *servidores*). A estratégia do algoritmo descentralizado é baseada na idéia de que qualquer tarefa de uma aplicação paralela, ao ser iniciada, compara a situação de carga do processador local com a situação de carga dos demais nodos que participam do sistema distribuído. A decisão sobre onde iniciar uma nova tarefa é baseada no estado do sistema. Cada nodo do sistema distribuído possui uma tarefa encarregada de coletar dados estatísticos sobre a utilização do processador. A carga de cada hospedeiro[1] é representada por um fator de carga, que é utilizado para as decisões de escalonamento. O hospedeiro conhece o fator de carga dos demais nodos por troca de mensagem entre os nodos que compõem o sistema distribuído. A utilização do algoritmo traz dois tipos de custos (*overhead*): o custo de comunicação e o custo de execução. O custo de comunicação é devido à troca de mensagens na rede. O custo de execução é devido às computações realizadas pelo algoritmo.

No balanço de carga para o ambiente distribuído, cada hospedeiro executa uma tarefa que é uma cópia idêntica do algoritmo de balanço de carga. Essas cópias executam assíncronamente e concorrentemente. O algoritmo de balanço de carga consiste de dois elementos básicos: um elemento de *informação* e um elemento de *controle*. O elemento de informação mantém dados sobre o estado do sistema distribuído, e é usado pelo elemento de controle para tomar as decisões de escalonamento. Estas informações caracterizam a carga dos outros hospedeiros no sistema. A carga de um hospedeiro é representada pelos seguintes critérios: comprimento da fila de tarefas *ready* do processador, fração de tempo livre da CPU no último minuto, e a velocidade do processador.

O elemento de informação trabalha de uma maneira periódica. Após o período de observação, as informações coletadas são enviadas, via *broadcast*, aos demais elementos hospedeiros. A coleta de informações pode ser imaginada como se fosse uma fotografia instantânea, tirada pelo elemento de informação, do estado do seu hospedeiro. Além disso, o elemento de informação é também responsavel por receber e manter as fotografias dos demais nodos da rede. A freqüência na troca de informações é um ponto crítico do algoritmo.

[1] Nodo do sistema que aloja e atende às tarefas solicitadas.

Quando as fotografias são mais freqüentes, as informações de estado serão mais atualizadas. Isso ajuda o elemento de controle a tomar melhores decisões quanto ao escalonamento. Por outro lado, esta operação de coleta de dados é cara, e freqüentes fotografias implicam em um aumento do custo de comunicação.

O elemento de controle utiliza as informações fornecidas pelo elemento de informação para decidir onde iniciar uma nova tarefa. Toda tarefa de um programa paralelo ao ser iniciada, devido à dinâmica no paralelismo da aplicação, compara a situação de carga de seu hospedeiro com a situação de carga dos demais nodos. O nodo escolhido, que pode inclusive ser o nodo hospedeiro da tarefa solicitante, é aquele que fornece a situação de carga mais favorável em relação aos demais nodos do sistema. A ativação do elemento de controle é assíncrona, sem necessidade de um período de ativação do mesmo. O elemento é despertado ao iniciar-se uma nova tarefa da aplicação paralela. O elemento de controle não faz a migração de processos ativos de um nodo para outro do sistema distribuído. Ou seja, uma vez associada uma tarefa a um determinado nodo essa associação permanece até o fim de execução da tarefa. O trabalho de [EaLZ88] mostra que esta migração resulta em geral em um grande *overhead*. Não existe um ganho de desempenho significativo quando se utiliza a migração de processos ativos de um nodo para outro do sistema distribuído, em relação à execução de um balanço de carga sem esta migração.

3 Descrição das Políticas de Escalonamento

Quando um sistema entra em um determinado estado, a probabilidade e o custo dele sair desse estado não é fixa, mas vai depender de um conjunto finito de alternativas chamado de *políticas*. A política adotada procura selecionar para cada estado qual a operação do sistema que nos dá o menor custo ou a maior recompensa. Nosso objetivo é encontrar um conjunto de políticas heurísticas para o escalonamento de tarefas em múltiplos processadores com arquitetura heterogênea que minimize o tempo de resposta (tempo gasto no serviço e o tempo gasto nas filas) de um programa paralelo. Em cada caso, o algoritmo determina a decisão de alocação ótima para cada estado, isto é, em que fila as tarefas de uma aplicação paralela devem ser assinaladas para minimizar o seu tempo de resposta.

Nesta seção vamos discutir a definição e a implementação das políticas de alocação consideradas. Estas políticas são dinâmicas, no sentido de que o número de processadores alocados para cada aplicação pode variar em tempos arbitrários de acordo com o número de tarefas paralelas da aplicação e com as informações de carga dos diversos nodos do sistema distribuído. Mesmo um processador que ainda não foi utilizado por nenhuma tarefa da aplicação paralela, pode ser utilizado para dar início à uma nova tarefa. Uma descrição completa de políticas de escalonamento para ambientes paralelos heterogêneos encontra-se em [MSPA93].

O escalonamento das tarefas de um programa paralelo passa a ser executado por uma biblioteca em tempo de execução *(run time library)* ao invés de ser executado pelo sistema operacional. Esta biblioteca pode ser otimizada para uma semântica e um modelo de programação específicos, enquanto as primitivas do sistema operacional precisam ser suficientemente gerais para acomodar uma variedade de modelos e linguagens de programação [AnLL88, Humm88]. Para executar o escalanomento utilizando esta biblioteca, o usuário já deve ter iniciado com antecedência o elemento de informação em cada nodo do sistema distribuído. Esta tarefa, de dar início ao elemento de informação, poderá ser implementada para ser chamada diretamente pelo sistema operacional. Após iniciados os elementos de informação, cada nodo tem ativa uma única tarefa de avaliação. Cada tarefa de avaliação faz uma amostragem periódica dos critérios de desempenho adotados, envia os dados coletados aos demais nodos do sistema distribuído e atualiza uma área de memoria compartilhada entre todos os processos, que estão no mesmo domínio do processador avaliado. Essa área de memória compartilhada consiste de uma tabela com as informações, necessárias ao escalonamento, de todos os nodos configurados para participarem do ambiente de execução da aplicação. O *overhead* para acessar essa memória compartilhada é um fator determinante no tempo de criação de uma nova tarefa. Logo, é importante que essa região compartilhada, seja tanto quanto possível, livre de contenção. Quanto maior o número de processadores, maior é a degradação devido ao bloqueio. Essa área de memória compartilhada foi desenvolvida para permitir a atualização concorrente e simplificar a troca de informações entre processos

de um mesmo nodo. Utilizando os recursos do sistema operacional UNIX, faz-se com que a leitura dos dados desta tabela seja permitida a todas as tarefas em execução no nodo. A atualização é permitida somente à tarefa de avaliação associada ao nodo.

Do programa de aplicação, utilizando a nova biblioteca, o usuário invoca a política de escalonamento que desejar. De um mesmo programa podem ser chamados, em pontos distintos, políticas de escalonamento diferentes. Isto dá ao usuário a flexibilidade de poder ajustar a política de escalonamento às características de sua aplicação. A seguir, são citadas as diferentes políticas de escalonamento implementadas:

- **PVM:** Na política apresentada pelo PVM, as tarefas são iniciadas a partir de uma lista circular que contém a identificação de cada processador que faz parte da execução paralela da aplicação. Essa lista circular comtém um apontador único para o próximo processador a receber uma nova tarefa. Nessa política, o escalonador ignora as possíveis diferenças de arquitetura e velocidade dos processadores que compõem o sistema distribuído. Além disso, é também ignorada a situação de carga destes processadores no momento da inicialização de uma nova tarefa. Na ocorrência da inicialização concorrente de mais de uma tarefa, o apontador da lista circular pode vir a ser um ponto de estrangulamento do sistema distribuído.

- **Fi - Fotografia instantânea:** Nesta política as tarefas são iniciadas a partir de valores calculados pela função de custo:

$$Fi(servidor_i) = \frac{n_i}{\mu_i \cdot f_i} + \frac{1}{\mu_i},$$

onde:

n_i : quantidade de tarefas na fila *ready* do servidor i.

μ_i : taxa de serviço relativa do servidor i em $jobs/seg$.

f_i : fração de tempo livre do servidor i no útimo minuto.

Quando uma nova tarefa necessita ser iniciada, a função de custo é avaliada para cada nodo. O nodo que fornece o menor custo é selecionado para executar a tarefa. Por exemplo, se existe mais de um nodo livre quando chega uma nova tarefa, a função indica o menor custo para aquele que tem maior velocidade, pois a primeira parcela de sua expressão é igual a *zero*. Essa função de custo é uma adaptação da função proposta por [WeSh88, ShWe88] para o escalonamento de lotes de *jobs* em um sistema de filas heterogêneas. Pela medida de f_i pode-se observar a natureza adaptativa dessa função: ela utiliza o valor f_i para determinar sua função de custo. A função de custo está ligada à aceitação de novas tarefas pelo servidor que, por sua vez, determina o futuro valor de f_i. Desta maneira, a política pode adaptar-se ao sistema e pode facilmente acomodar-se às mudanças na taxa de chegada e configuração do sistema.

- **Fr - Fotografia instantânea com retoque:** A política Fi não considera modificações na fotografia devido ao início simultâneo de mais de uma tarefa. Isto porque, o momento de reavaliação periódica do sistema é independente do momento de início de uma nova tarefa. O início de uma nova tarefa para o sistema distribuído só depende da aplicação, sem nenhum tipo de sincronismo com as tarefas de avaliação dos nodos. Se entre duas avaliações sucessivas for dado início a mais de uma tarefa, o escalonador vai tomar como base para o cálculo de sua função de custo a mesma fotografia. Para aplicações paralelas que necessitam iniciar um número muito grande de tarefas simultaneamente, esse efeito pode provocar uma concentração de tarefas em um mesmo nodo do sistema distribuído. Para minimizar esse efeito, a política Fr propõe fazer um retoque na fotografia após o início de cada tarefa. Isso é feito incrementando o número de processos na fila do nodo selecionado.

- **Políticas com a utilização de prioridade:** As políticas Fi e Fr podem ser aplicadas com a utilização de um novo parâmetro: uma prioriade fornecida pelo usuário a cada tarefa. Com base nos dados fornecidos pelas tarefas de avaliação dos nodos, a carga média de utilização do sistema distribuído é calculado. Se essa carga estiver acima de certo valor (80% na implementação usada nos experimentos desse artigo), as novas tarefas que chegam vão para uma fila de prioridades. Todas as tarefas dessa

fila são iniciadas periodicamente (50 milisegundos nessa implementação) ou quando a carga média for menor do que o valor pré-determinado. Essa prioridade representa a importância dessa tarefa em relação às demais tarefas que ainda não foram iniciadas. Isto pode ser útil na execução de um programa paralelo, porque dá-se ao usuário a possibilidade de definir diferentes prioridades entre as tarefas da aplicação paralela, podendo dar maior prioridade às tarefas que podem se constituir em um gargalo do grafo de tarefas.

4 Aplicações Paralelas

O artigo realiza uma avaliação experimental de desempenho de programas paralelos em um ambiente de computação distribuído e heterogêneo. Para compararmos os resultados obtidos utilizamos um *mix* de aplicações paralelas, que são iniciadas em um mesmo instante, mas com diferentes políticas de alocação das tarefas aos processadores. No ambiente distribuído, por não possuir mecanismos para a utilização de memória compartilhada entre os processadores, a comunicação entre as tarefas é muito cara. O fator limitante de um bom desempenho deixa de ser a habilidade de encontrar suficiente paralelismo na aplicação e passa a ser a paralelização do algoritmo visando minimizar a comunicação entre tarefas [LoSA92]. Isso nos leva à obtenção de tarefas de alta granularidade (*coarse grain*), típicas de programas paralelos em ambientes de computação distribuído.

Esta seção descreve sucintamente as aplicações utilizadas nos experimentos: Cálculo dos Harmônicos, Problema da Equação de Bessel e Multiplicação de Matrizes. O perfil do paralelismo dessas aplicações é mostrado na figura 1, que apresenta o tempo de execução de cada aplicação em função do número de *workstations*. A primeira caracteriza-se pela grande quantidade de cálculos numéricos e a última pela alta troca de mensagens entre as tarefas. Já o cálculo dos harmônicos, se situa numa posição intermediária, apresentando tanto cálculos numéricos quanto trocas de mensagens de uma forma equilibrada. Na descrição de cada aplicação, inclui-se uma representação gráfica em forma de grafo direcionado, onde os nodos do grafo representam os processos e os arcos representam a dependencia entre eles.

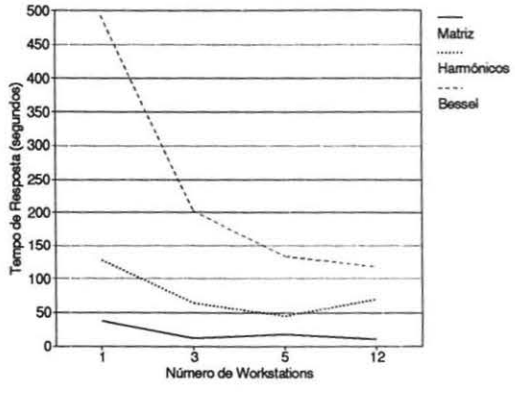

Figura 1: Perfil do paralelismo das aplicações

4.1 Cálculo dos Harmônicos

O algoritmo paralelo proposto é do tipo *master-slave*. Nessa estrutura o processo principal envia tarefas a serem realizadas por outros elementos de processamento (processos filhos), coleta os resultados e toma decisões dos próximos passos a seguir. A alocação de tarefas e o balanceamento do sistema está sob controle do programa. A contenção é reduzida nesse algoritmo, pois toda a troca de mensagens ocorre entre o processo principal e os processos filhos.

O programa faz a transformação gradual de uma onda através da soma de harmônicos utilizando um ambiente de computação distribuído. O grafo da aplicação é mostrado pela figura 2. O processo *pai* é crítico para o desempenho desta aplicacação e é um bom candidato à política de prioridades.

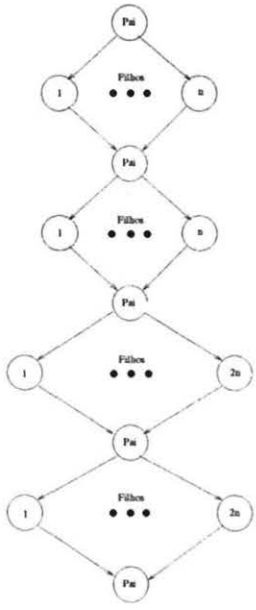

Figura 2: Grafo da aplicação do cálculo dos harmônicos

4.2 Problema da Equação de Bessel

A equação de Bessel dada por:

$$x^2y'' + xy' + (x^2 - n^2)y = 0 \tag{1}$$

É uma equação diferencial onde n é uma constante e x uma variável independente que aparece em diversas aplicações matemáticas e científicas envolvendo simetria circular.

Para se conseguir a solução dessa equação diferencial não linear tem-se a seguinte expressão :

$$y = J_n(x) \tag{2}$$

onde J é a função de Bessel de ordem n do primeiro tipo.

Para valores de x menores do que aproximadamente 15 as funções de Bessel podem ser calculadas através da série:

$$J_n(x) = \sum_{k=0}^{n} \frac{-1^k}{k!\Gamma_i(n+k+1)}(x/2)^{n+2k} \tag{3}$$

onde

$$\Gamma_i(x) = \sqrt{\frac{2\pi}{x}}x^x e^y \tag{4}$$

$$y = \frac{1}{12x} - \frac{1}{360x^3} - x \tag{5}$$

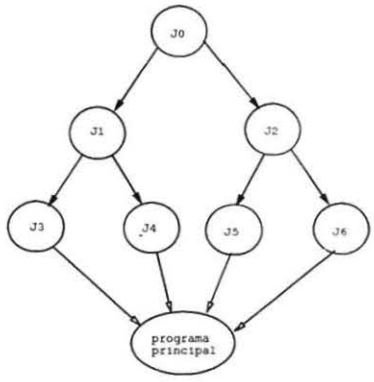

Figura 3: Grafo da aplicação do cálculo da equação de Bessel

Basicamente a aplicação paraleliza o cálculo de cada função de Bessel de ordem n. Essa paralelização gera um grafo de tarefas em forma de uma árvore binária completa (figura 3). Em particular, não existe comunicação entre processos filhos, apenas entre o pai e os fillhos e entre os nodos-folhas e o programa principal que criou a raiz.

4.3 Multiplicação de Matrizes

O algoritmo da multiplicação de matrizes em paralelo utilizado nesse artigo foi proposto por [Fox88]. O algoritmo consiste na divisão das matrizes em sub-blocos e na distribuição desses sub-blocos em processos distintos. Assim, parte da multiplicação é feita por cada processo isoladamente e em paralelo com os demais

processos, minimizando o tempo de resposta da aplicação. A implementação consiste em dois processos distintos:

- Processo Pai: Inicia e coordena os demais processos.
- Processo Filho: Multiplica as matrizes de entrada (realizada em paralelo pelas várias instâncias deste processo).

A representação gráfica dessa aplicação é mostrada na figura-4. O algoritmo proposto se caracteriza pelo grande número de troca de dados entre o processo pai e os processos filhos, e destes entre si.

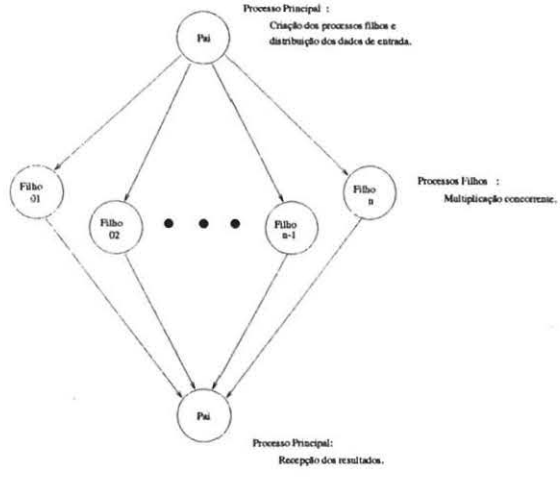

Figura 4: Grafo da aplicação de multiplicação de matrizes

5 Ambiente de Testes

5.1 Rede SUN/UNIX DCC-UFMG

A rede do Departamento de Ciência da Computação — UFMG é composta por um conjunto de *workstations* e servidores interconectados por uma rede *Ethernet* de 10 Mbits. A rede utiliza o protocolo TCP/IP, onde os nodos processadores estão conectados através de uma topologia de barramento, isto é, cada nodo está ligado à um único barramento e a rede utilizada possui dois barramentos interligados por um *gateway*. Nos experimentos citados nesse artigo utilizou-se as *workstations* apresentadas na tabela 1.

Workstation	MIPS	MFLOPS
SLC	17.4	2.1
SPARC-1	17.4	2.1
SPARC-2	28.5	4.2
IPX	28.5	4.2
SPARC-630	2×28.5	2×4.2

Tabela 1: *Workstations* e seus desempenhos nominais

5.2 PVM - Parallel Virtual Machine

O complemento do ambiente de testes na rede DCC-UFMG é o **PVM** [Sund90], um *software* que permite a execução de programas paralelos num ambiente heterogêneo. Esse software atua interligando várias máquinas da rede, formando um ambiente pré-estabelecido pelo usuário. Esse ambiente passa a ser visto de uma forma transparente, como se fosse um único nodo do sistema.

Enquanto, ao nível do usuário, pacotes de gerenciamento de um ambiente de computação distribuído diferem em alguns detalhes de implementação, os conceitos básicos são os mesmos em todos os casos. Ao iniciar-se sua execução, um processo núcleo (*kernel*) de gerenciamento desse ambiente é iniciado para cada processador físico. Estes processos provêem um ambiente genérico onde as aplicações irão executar.

5.3 Integração: PVM/Escalonador

Em cada nodo do sistema distribuído existe um processo *daemon* que atua sobre o PVM coletando periodicamente informações sobre a carga do sistema. As aplicações continuam tendo seus processos escalonados transparentemente, porém, ao invés de utilizar as primitivas de inicialização da biblioteca PVM [BDGM91], utiliza-se as primitivas da nova biblioteca. O ambiente formado pelas aplicações paralelas, processos *daemons* e o PVM é mostrado na figura 5.

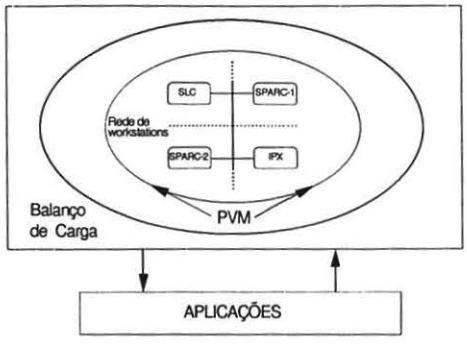

Figura 5: Ambiente PVM + Balanço de Carga

6 Experimentos e Resultados

Nessa seção discute-se uma série de experimentos realizados para mostrar o desempenho das aplicações no ambiente de testes com o uso do escalonador.

6.1 Período de Coleta da Carga do Sistema

O escalonador faz uma coleta de dados do sistema periodicamente, atualizando a memória compartilhada onde são guardadas as informações sobre a carga dos nodos do sistema. Esse período de coleta deve ser um valor que minimize o tempo de resposta das aplicações. Com o objetivo de obter o período de varredura ideal para o escalonador, realizou-se uma série de experimentos com a aplicação dos harmônicos. Os experimentos foram realizados para um ambiente formado por 4 nodos: 2 *workstations* SPARC-2 e 2 *workstations* SLC. Os experimentos foram divididos em dois casos: para uma carga baixa (20% de utilização média do sistema) e para uma carga alta (80% de utilização média do sistema). Os resultados são apresentados pela figura 6.

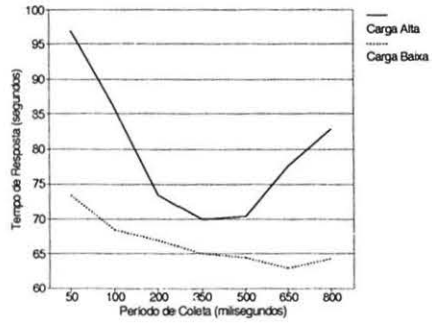

Figura 6: Tempo de resposta × Período de coleta

Para o caso da carga alta, verifica-se que para períodos curtos, apesar de se ter atualizações mais recentes do estado do sistema, o *overhead*[2] é muito alto, degradando assim o desempenho das aplicações. A medida que o período aumenta, o desempenho melhora consideravelmente, até um certo ponto onde, as informações já não estão muito atualizadas, levando os processos para máquinas que não são adequadas no momento do escalonamento, ou seja, gerando piores resultados.

Para a curva obtida com a carga baixa do sistema, verifica-se um comportamento semelhante à curva de carga alta. Porém, para esta situação, a carga se mantém constatemente baixa, visto que a carga de cada nodo do sistema não se modifica com muita freqüência. Assim, para um maior intervalo de valores, o *overhead* é superior ao ganho obtido pelas tarefas através do escalonador. Portanto, como visto por esta curva, o período de coleta que minimiza o tempo de resposta das aplicações é um valor maior do que para o caso da carga alta.

[2] Custo de comunicação e processamento.

6.2 Speedup

A fim de obter o ganho de desempenho das aplicações sendo executadas sobre o escalonador, uma série de experimentos foram feitos com vários conjuntos de *workstations* e uma carga média global do sistema de 50%. O conjunto de *workstations* é mostrado a seguir:

- 3 *workstations*: SLCs(3).
- 6 *workstations*: SPARC-1(1), SPARC-2(1), SLCs(4).
- 9 *workstations*: SPARC-1(1), SPARC-2(1), IPX(1), SLCs(6).
- 12 *workstations*: SPARC-1(1), SPARC-2(2), IPX(1), SPARC-630(1), SLCs(7).

A medida utilizada para representar o ganho devido ao uso do escalonador distribuído é denominada *speedup*, definido assim:

$$speedup(política[i]) = \frac{Tempo\ de\ Resposta(política[i])}{Tempo\ de\ Resposta(política[PVM])},$$

onde: *Tempo de Resposta(política[i])* é o tempo de resposta obtido pela execução da aplicação utilizando a política i e *Tempo de Resposta(política[PVM])* é o tempo de resposta da aplicação utilizando a biblioteca PVM, isto é, sem a presença do escalonador distribuído. Vale a pena ressaltar que esse *speedup* não se refere ao ganho do paralelismo sobre o processamento seqüêncial, e sim do ganho do escalonador sobre o PVM puro.

A figura 7 mostra o *speedup* médio obtido entre todas as políticas para cada aplicação. O *speedup* médio foi calculado pela fórmula:

$$\overline{speedup} = \frac{\sum_{i=1}^{n} speedup\ (política[i])}{n},$$

onde n é o número de políticas utilizadas ($n = 4$: *Fi e Fr, com e sem prioridade*).

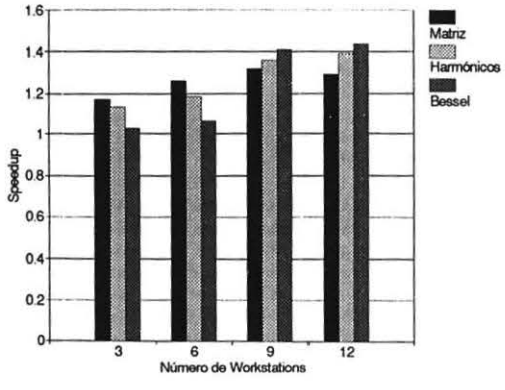

Figura 7: *Speedup* médio × Número de *workstations*

Pela figura 7, observa-se que o desempenho médio obtido pela execução de aplicações paralelas com a utilização do escalonador fornece resultados superiores aos obtidos pela execução das mesmas aplicações com a utilização da política de escalonamento do PVM.

A fim de obter dados comparativos do *speedup* para diferentes números de *workstations*, uma nova série de experimentos foi realizada para o mesmo conjunto de *workstations* com a política de trabalho Fi. A figura 8 mostra o *speedup* obtido pelo mix das aplicações:

$$speedup(mix) = \frac{\sum_{i=1}^{k} speedup(aplicação[k])}{k},$$

onde k é o número de aplicações utilizadas ($k = 3$: *harmônicos, matriz e bessel*).

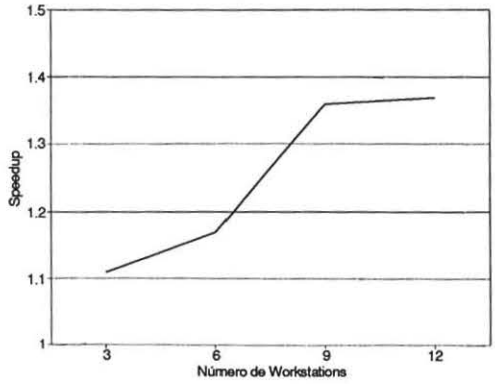

Figura 8: *Speedup(mix)* × Número de workstations

As figuras 7 e 8 mostram que o *speedup* é tanto maior quanto maior for o número de *workstations*. Esse valor tende a um limite, a partir do qual, o aumento do número de *workstations* não ocasiona um aumento linear no *speedup*.

6.3 Carga Baixa × Carga Alta

O gráfico da figura 9 apresenta o *speedup* médio do *mix* das aplicações obtido durante uma nova série de experimentos realizados (12 *workstations*) para dois tipos de carga média do sistema: carga baixa (40% de utilização) e carga alta (90% de utilização). Por este gráfico, pode-se observar que o ganho de desempenho com a utilização das políticas de escalonamento propostas é tanto maior quanto maior a carga média do sistema. Isto porque, quando a carga do sistema é alta, as políticas de escalonamento que procuram se beneficiar da heterogeneidade do ambiente distribuído são mais eficientes, pois as tarefas que trabalham com a política de escalonamento do PVM têm uma maior probabilidade de serem escalonadas para nodos inadequados (nodos com carga mais alta do que a carga média global) do sistema.

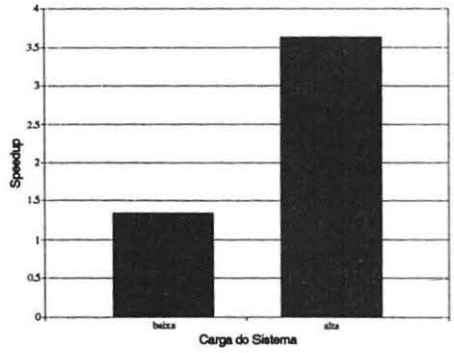

Figura 9: Speedup: carga baixa × carga alta

6.4 Políticas

Aproveitando os experimentos realizados na comparação dos *speedups* entre carga baixa e alta, obteve-se o gráfico da figura 10. Este gráfico foi baseado no conjunto de resultados obtidos, com carga baixa, para o *speedup* médio de todas as aplicações. O gráfico compara as políticas de trabalho descritas na seção 3. As políticas Fi/P e Fr/P referem-se respectivamente às políticas Fi e Fr com a utilização de prioridade.

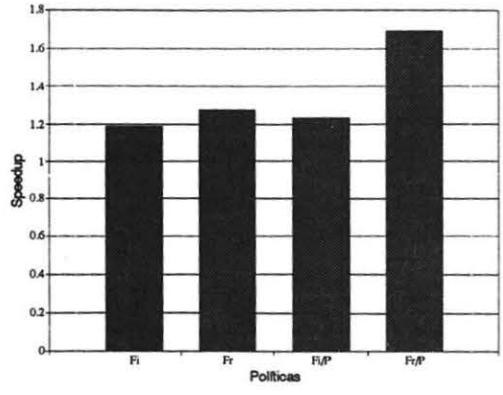

Figura 10: Speedup para diferentes políticas de trabalho

Pela figura 10 observa-se que:

- **Fi × Fr:** A política *Fr* comporta-se melhor no escalonamento de aplicações paralelas em relação à política *Fi*. Isto porque a política *Fr* após iniciar uma tarefa, provoca um *retoque* na *fotografia*, deixando-a mais preparada para o início de uma nova tarefa.
- **Prioridade:** As políticas com prioridade mostraram melhores resultados do que as respectivas políticas sem prioridade. Isto porque às tarefas *gargalo* das aplicações paralelas foram destinados os processadores com maior capacidade de processamento no momento solicitado.

7 Conclusões

O desempenho de programas paralelos em um ambiente de computação distribuído e heterogêneo pode ser fortemente melhorado utilizando-se de um escalonador que explore essa heterogeneidade na alocação das tarefas aos processadores. Nesse mesmo ambiente, quanto maior a carga média dos nodos hospedeiros, maior o ganho de desempenho das aplicações paralelas que utilizam um escalonador determinístico, em relação ao desempenho de aplicações paralelas que não fazem uso de um escalonador *inteligente*. A freqüência da troca de informações entre os nodos deve ser um valor balanceado: não muito alto, para evitar um grande *overhead*, e nem muito baixo, para se evitar o resultado de uma decisão de escalonamento mais pobre. Conclui-se finalmente que o a utilização do escalonador em sessões de trabalho com carga alta traz um ganho de desempenho superior àquele obtido em seções cuja carga do sistema é baixa. Os experimentos paralelos realizados e reportados neste trabalho confirmam as conclusões apresentadas.

Esse trabalho pode ser estendido na criação de novas políticas heurísticas de escalonamento visando obter resultados ainda mais significativos no escalonamento de programas paralelos. Uma outra extensão ao trabalho seria o desenvolvimento de um escalonador tolerante a falhas, de modo que, as tarefas de uma aplicação paralela possam ser iniciadas normalmente mesmo que ocorram problemas em algum nodo do sistema distribuído. Finalmente, o projeto prevê a adaptação deste escalonador para outros ambientes de programação paralela/distribuída, como LINDA e Express.

Referências

[AnLL88] Anderson, P. E., Lazowska, E. D. and Levy, H. M. "The Performance Implications of Thread Management Alternatives for Shared Memory Multiprocessors", *Technical Report 88-09-04*, University of Washington, Seattle. September, 1988.

[BDGM91] Benguelin, A., Dongarra, J., Geist, A., Manchek, R. and Sunderam, V. "PVM: A User's Guide to PVM - Parallel Virtual Machine", *Technical Report ORNL/TM-11826*. Oak Ridge National Laboratory. December, 1991.

[EaLZ88] Eager, D. L., Lazowska, E. D. and Zahorjan, J. "The Limited Performance Benefits of Migrating Active Processes for Load Sharing", *Proc. 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Performance of Evaluation Review*. May, 1988.

[Fox88] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K. and Walker, D. W. *"Solving Problems on Concurrent Processors, Volume I: General Techniques on Concurrent Processors"* , Prentice-Hall International, Inc. 1988.

[GuTU86] Gupta, A., Tucker, A. and Urushibara, S. "The Impact of Operating System Scheduling Policies and Synchronization Methods on the Performance of Parallel Aplications" *Proc. 1991 ACM SIGMETRICS Conf. on Meas. and Mod. of Comp. Sys.*. May, 1986.

[HaJi87] Háç A. and Jin, X. "Dynamic Load Balancing in a Distributed System Using a Decentralized Algotithm", *The 7th International Conference on Distributed Computing Systems*, pp 170-177. September, 1987. 1987 *IEEE*.

[Humm88] Hummel, S. F. "SMARTS Shared Memory Multiprocessor Ada Run Time Supervisor", *Ph. D. Thesis*, New York University, New York. 1988.

[LoSA92] Loures, E. F., Silva, G. P. Jr. e Almeida, V. A. F. "Análise de Desempenho de Programas Paralelos em Redes de Workstations", *IV Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho*. Outubro, 1992

[MSPA93] Mensacé, D. A., Saha, D., Porto, S. C. S., Almeida, V. A. F. e Tripathi, S. K. "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures", a ser publicada no Journal of Parallel and Distributed Systems.

[ShWe88] Shenker, S. and Weinrib, A. "Asymptotic Analysis of Large Heterogeneous Queueing Systems", *Proceedings of the ACM SIGMETRICS Conference*, pp 56-62. May, 1988.

[Sund90] Sunderam, V. S. "PVM: A Framework for Parallel Distributed Computing", *Concurrency: Pratice & Experience, Vol. 2 No. 4*. December, 1990.

[WeSh88] Weinrib, A. and Shenker, S. "Greed is not enough: Adaptive load sharing in large heterogeneous systems", *Proceedings IEEE INFOCOM'88, pages 986-994*. 1988.