Um Esquema de Escalonamento em Dois Níveis para Jobs Paralelos em uma Rede de Estações de Trabalho

Virgílio Augusto Fernandes Almeida*
José Nagib Cotrim Árabe†
Adriana de Andrade Oliveira
Marco Aurélio de Souza Mendes

Resumo

Redes de estações de trabalho são um ambiente adequado para processamento paralelo. Ambientes de computação distribuída como o PVM provêm integração entre máquinas heterogêneas a fim de suportar a execução de vários jobs paralelos. Embora estes sistemas permitam que programas paralelos executem num conjunto de estações de trabalho, eles não tratam questões relativas ao gerenciamento e coordenação da distribuição do trabalho pela rede. Este trabalho trata o problema do escalonamento de jobs paralelos numa rede heterogênea de estações de trabalho. Heterogênea significa nesse contexto uma gama de arquiteturas de processadores e um amplo conjunto de jobs paralelos, com graus de paralelismo diferentes e mutáveis, co-existindo com jobs seqüenciais. O artigo consiste de duas partes. A primeira apresenta o projeto e a implementação de um escalonador distribuído para jobs paralelos, cujos objetivos são manter uma carga de trabalho balanceada e reduzir o tempo médio de execução dos jobs. A segunda parte analisa o impacto de jobs paralelos globais na performance de jobs interativos locais e propõe mecanismos para minimizar este impacto.

Abstract

Networks of powerful workstations are an adequate means for parallel processing. Distributed computing environments such as PVM provide the integration of multiple heterogeneous computing platforms to support execution of multiple parallel jobs. Although these systems allow an user to write parallel programs that run on a collection of workstations, they do not address a number of "system issues" that are necessary to manage and coordinate the distribution of work on the network. This work addresses the problem of scheduling parallel jobs on a heterogeneous network of workstations. By heterogeneous we mean a variety of processor architectures and a varying set of parallel jobs with different and changing degrees of parallelism, coexisting with sequential jobs (interactive and batch) at personal workstations. The paper has two parts. The first one presents the design and implementation of a distributed scheduler for parallel jobs with the objectives of maintaining a balanced system-wide workload and reducing the average execution time for the jobs. The second part analyzes the impact of global parallel jobs on the performance of local interactive users and proposes mechanisms to minimize that impact.

*Parcialmente fincanciado pelo CNPq

†Parcialmente suportado pelo CNPq. Atualmente, ele é um visitante escolar na School of Computer Science of Carnegie Mellon University (arabe@cs.cmu.edu)

1 Introdução

É um fato bem documentado na literatura que estações de trabalho, que representam uma parte significativa do poder de processamento disponível na redes atuais, permanecem ociosas a maior parte do tempo. Isto incentivou uma intensa pesquisa para a redução dos ciclos de máquina ociosos. Vários sistemas foram propostos para permitir o uso coordenado de máquinas diferentes como se estas fossem um único multicomputador. Neste aspecto, várias questões devem ser tratadas.

Este trabalho trata o problema do escalonamento de jobs paralelos numa rede heterogênea de estações de trabalho. Por heterogeneidade entende-se uma gama de arquiteturas de processadores e uma amplo conjunto de tarefas concorrentes com graus de paralelismos diferentes e mutáveis, co-existindo com jobs seqüenciais (interativos e batch) em estações de trabalho pessoais.

Primeiramente são descritos o projeto e a implementação de um escalonador distribuído para jobs paralelos com os objetivos de manter uma carga de trabalho balanceada e reduzir o tempo médio de execução dos jobs paralelos. Este escalonador não leva em conta a interação com processos locais em estações de trabalho individuais. A despeito disto, resultados satisfatórios foram obtidos em termos do tempo médio de resposta de processos paralelos. A seguir, são apresentados um modelo e resultados de simulação que tangem o problema da interação de tarefas paralelas geradas em máquinas remotas e processos gerados por usuários interativos. Ao final, são propostos mecanismos para minimizar este impacto em estações de trabalho privadas.

2 Escalonamento numa Rede de Estações de Trabalho

Vários sistemas já existentes utilizam grupos de estações de trabalho como máquinas paralelas virtuais. Entre os mais populares estão o PVM [9], Linda [15], Express [11], P4 [5] e, mais recentemente, MPI [14]. Embora tais sistemas permitam escrever programas paralelos que executem em várias estações de trabalho, não tratam várias questões relativas à coordenação e distribuição do trabalho pela rede. Uma importante questão é o escalonamento de tarefas geradas pelos processos paralelos entre as várias estações disponíveis. Relacionado ao escalonamento está o balanço de carga, desde que um dos objetivos de um bom escalonador é manter uma carga balanceada.

O sistema Condor [12] foi um dos primeiros a implementar um escalonador para processos batch numa rede de estações de trabalho. O sistema monitora as máquinas disponíveis num *pool* e atribui prioridade incondicional para os jobs interativos, i.e., qualquer atividade local remove esta máquina do pool de máquinas disponíveis e remove qualquer tarefa remota que porventura esteja utilizando este processador. Isto é alcançado pelo fornecimento de um mecanismo transparente de *checkpoints*, pelo qual processos são migrados e completados. Vários sistemas de gerenciamento foram costruídos posteriormente utilizando a estrutura fornecida pelo Condor. Uma comparação de sistemas similares é realizada em [10].

Ambientes paralelos como o PVM utilizam um escalonador local em cada nodo da rede (geralmente residentes no *kernel* do Unix) a fim de alocar as tarefas geradas por programas paralelos. Embora isto ofereça a vantagem de portabilidade desde que nenhum suporte de sistema é necessário, quedas inaceitáveis de performance para processos paralelos podem ocorrer bem como interferências indesejáveis em processos locais interativos. Por isto, esta questão foi

tratada por vários sistemas em diferentes níveis. A nível de aplicação, o trabalho apresentado em [7] descreve três métodos para migração de tarefas de aplicações PVM de modo a adaptar a variação de carga numa rede. Os dois primeiros, MPVM e UPVM, migram processos de modo transparente, enquanto que o terceiro, ADM, é uma metodologia que realiza distribuição de carga através de movimentação de dados. Também a nível de aplicação, o sistema DOME [4] provê uma biblioteca de objetos ditribuídos que se mantêm balanceados automaticamente sobre múltiplos computadores. Quando um programa executa, o sistema tenta manter a carga de trabalho balanceada entre as máquinas através da movimentação de dados entre os processadores. Outros sistemas tratam o balanço de carga a nível de compilador. Por exemplo, [17] descreve um compilador que gera código para transferir dinamicamente partes da carga de trabalho de uma aplicação entre processadores para aumentar a performance. Enquanto se esforçam para alcançar os objetivos propostos, tais técnicas não podem tratar o problema da competição entre cargas de trabalho paralelas e seqüenciais interativas numa rede. Em particular, também não tratam a questão de vários jobs paralelos simultâneos, em que cada qual tenta melhorar sua própria performance movendo-se pela rede. Não é evidente que algumas boas decisões locais aumentam a performance global. De fato, parece que o contrário é que seja verdade, devido ao tráfego inaceitável na rede e à troca de contexto entre processadores.

Outros trabalhos estão tratando o problema do escalonamento num nível mais global. O projeto *NOW* de Berkeley [3], por exemplo, trabalha no nível de rede, almejando por um sistema que forneça um performance de supercomputadores para processos paralelos sem interferir na performance dos jobs interativos. O trabalho aqui apresentado é, pois, uma contribuição nesta direção.

3 Um Escalonador Global Distribuído

A fim de estudar mais a fundo o problema de escalonamento de tarefas, foi construído um escalonador dinâmico distribuído para jobs paralelos numa rede de estações. Através de algoritmos *sender-intitiated* [8], o escalonador realiza atribuição global de tarefas aos nodos do sistema. Cada nodo processa uma cópia idêntica da função de escalonamento, sem um nodo mestre, como mostrado pela figura 1. O sistema foi construído sobre o PVM [9] e implementado através de dois *daemons* que rodam em cada estação da rede.

Uma função do escalonador é coletar a informação de carga de cada nodo e periodicamente trocar esta informação com os outros nodos. A outra função do escalonador é usar esta informação de carga global para escalonar uma nova tarefa ao nodo com menor carga. O status da carga de cada nodo é representado por um *índice de carga*, que é uma função do número de processos na fila *ready* de cada nodo, da fração do tempo ocioso no último minuto e da velocidade do processador (e.g., SPEC, MFLOPS). A troca de informação entre os nodos é em intervalos de tempo pré-definidos. A fim de reduzir o tráfego de mensagens na rede, um nodo somente envia sua informação de status quando sua carga varia significativamente. Em outras palavras, um intervalo de variação de carga é definido pelos nodos de forma tal que cada nodo só transmite uma mensagem quando sua variação de carga excede o intervalo estabelecido. A fim de se avaliar a performance do escalonador, quatro políticas diferentes foram investigadas.

O PVM padrão aloca as tarefas aos processadores de maneira circular e não leva em conta a carga de cada nodo. A primeira política proposta, chamada Fotografia Instantânea (FI),

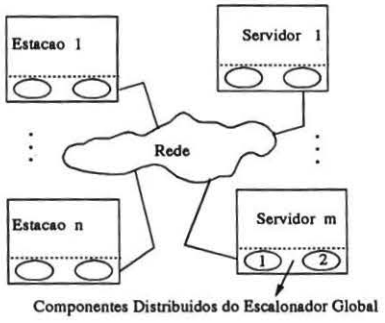

Figura 1: Esquema do Escalonador Global Distribuído

aloca o nodo com a função mínima de carga, dada por:

$$\psi_i = \frac{n_i}{\mu_i \times f_i} + \frac{1}{\mu_i}$$

onde n_i é o número de tarefas na fila ready do procesador, f_i é a fração do tempo livre do processador no último minuto, e μ_i é a velocidade do processador i. "Instantânea" deve ser aqui interpretada num sentido menos rigoroso, desde que não é possível obter uma fotografia real sem um relógio global, que não existe numa rede de estações de trabalho. Esta função, adaptada do algoritmo proposto em [18], lida com o fator heterogeneidade. Logo, quando uma nova tarefa é atribuída, o escalonador pega o nodo com o menor valor de ψ. Se existem vários nodos disponíveis, o escalonador utilizará o mais rápido (i.e., o nodo com o maior indicador de velocidade), que minimiza ψ. Quando o sistema está ocupado, o escalonador escolhe o nodo que oferece a melhor combinação de carga mínima (representada por n_i e f_i) e maior velocidade (μ_i).

A segunda política (Fotografia Instantânea Modificada FIM) é uma modificação da primeira que tenta evitar uma sobrecarga de um processador no caso de uma chegada maciça de tarefas entre trocas de informações de carga. Neste cenário, o nodo com menor carga pode rapidamente se tornar o mais carregado. Este problema é evitado pelo atualização da versão local da tabela global de status de carga, sem a transmissão da modificação (*broadcasting*), o que poderia ocasionar um alto *overhead* de comunicação. Assim, novas decisões de escalonamento neste nodo levam em conta as alocações anteriores, independente da chegada de novas informaçoes de outros nodos. Finalmente, a terceira e a quarta políticas permitem ao usuário especificar prioridades às tarefas. Tarefas prioritárias são colocadas em filas especiais de cada nodo, que são servidas primeiro pelo escalonador. Deve-se notar que uma carga baixa no sistema leva ao não uso dessas filas pelo escalonador, já que o mesmo provavelmente conseguirá uma alocação imediata num nodo ocioso. Entretanto, quando o sistema se torna mais carregado, as chances de uma alocação imediata diminuem. Neste caso, as tarefas são enfileiradas até que um processador se torne disponível. Vale notar que as políticas propostas não migram tarefas não concluídas.

O escalonador foi avaliado numa rede de 12 estações heterogêneas e servidores (SUN, SGI, DEC) interconectados por uma rede Ethernet de 10 Mbits/sec. Um conjunto de aplicações

paralelas foi desenvolvido utilizando o PVM para compor a carga de trabalho paralela. As aplicações — Multiplicação de Matrizes (bloco-particionada), equação de Bessel e série de harmônicos — possuem estruturas de paralelismo diferentes, com um *mix* de números de operações de ponto flutuante e troca de mensagens altos e baixos. Vários experimentos foram realizados, com nodos carregados de maneira variada a fim de se determinar o intervalo ótimo de troca de informação de carga e o impacto de cada política na execução da carga de trabalho paralela. Uma descrição detalhada da implementação do escalonador, com resultados

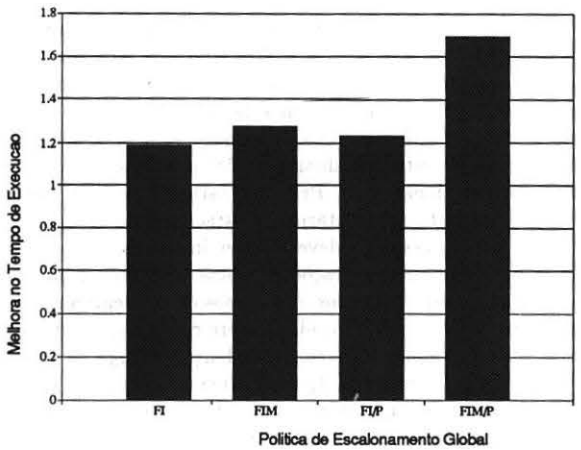

Figura 2: Melhora no Tempo de Execução × Política de Escalonamento

adicionais de performance pode ser encontrada em [2].

A figura 2 mostra o impacto das políticas de escalonamento propostas na execução de aplicações paralelas reais em sistemas altamente carregados (utilização global do sistema maior que 70%). A métrica de performance usada é o tempo de execução normalizado, definido como o tempo médio de execução do sistema com o PVM padrão sobre o tempo médio de execução do sistema com as políticas propostas. Nota-se a partir do gráfico da figura 2 que todas as políticas de escalonamento propostas melhoram a performance global das cargas de trabalho paralelas que rodam sobre o PVM. Os tempos de execução normalizados são todos maiores que um. A FIM/P exibe a melhor performance. A razão advém do fato que esta leva em conta informações a respeito da estrutura do job (e.g. tarefas críticas) a fim de tomar as melhores decisões. É importante mencionar a característica adaptativa desta política, desde que ela usa algoritmos distintos de acordo com a utilização global do sistema.

Embora tais resultados sejam realmente bons, algumas questões importantes são ignoradas pelo escalonador, principalmente no que concerne à competição entre tarefas remotas de aplicações paralelas e carga de trabalho seqüencial normal originada localmente em cada estação.

Não há interação entre o escalonador global (implementado no espaço de endereçamento

do usuário) e o escalonador do sistema operacional nativo de cada nodo. Isto significa que processos gerados remotamente competem pelos recursos locais igualmente com as tarefas locais de usuários interativos. Usuários locais se sentem arredios à idéia de compartilhar sua máquina com outros, já que isto provoca um aumento no tempo de resposta de suas tarefas. Portanto, o estudo é aqui direcionado na busca de mecanismos que minimizem este tipo de problema, enquanto utilizem os recursos ociosos. A próxima seção propõe modificações no escalonador original a fim de lidar com esta situação.

4 Um Esquema de Escalonamento em Dois Níveis

Pode-se considerar que a carga de trabalho de uma rede de estações consiste de três classes: 1) grandes jobs paralelos que são executados utilizando-se os recursos globais da rede; 2) jobs locais interativos que compreendem as atividades de processamento normal (e.g: edição, compilação, etc.) submetidos pelo proprietário da estação, e 3) jobs locais batch que são submetidos pelo proprietário da estação para execução local, com uma prioridade menor (e.g.: prioridade *nice*). Numa rede de estações, duas questões devem ser consideradas para guiar o projeto do mecanismo de escalonamento. Primeiro, estações pessoais permanecem ociosas a maior parte do tempo. Segundo, proprietários de estações (aqueles que compram equipamentos para propósitos particulares) não deveriam ser incomodados pela execução de jobs paralelos submetidos a partir de outras estações remotas. Em outras palavras, mecanismos de escalonamento deveriam tentar usar todos os recursos ociosos embora devessem preservar os direitos do proprietário, dando a eles prioridade sobre computações globais. É importante notar que estes dois objetivos mudam o conceito de balanço de carga de um sistema, conceito esse adotado na maioria dos escalonadores. Quando uma tarefa for ser executada em algum outro ponto da rede, o fator mais importante na escolha da estação será a possibilidade de obter um menor tempo de execução, e não a estação menos utilizada, desde que prioridade será dada a jobs seqüenciais locais. Assim, o esquema de escalonamento em dois níveis tem três objetivos principais:

- Reduzir o tempo de execução de jobs paralelos globais.
- Minimizar os efeitos da execução de jobs paralelos globais na performance de jobs locais em estações privadas.
- Alocar as estações mais apropriadas a diferentes partes do job paralelo, de modo a obter um melhor casamento entre os recuros oferecidos e as necessidades computacionais.

O mecanismo de escalonamento em dois níveis ajuda o escalonamento destas três classes de jobs numa rede de estações. A nível de rede, o escalonador distribuído global é responsável pelo escalonamento de tarefas de jobs paralelos para as estações da rede, de acordo com a função que considera a carga global de cada estação, o fator de heterogeneidade das estações e a existência de uma carga de trabalho seqüencial local. Esta função é implementanda fora do kernel, como um *deamon*. Em tempo de escalonamento, o componente local do escalonador global aloca a primeira tarefa à estação com o mínimo custo, dado por:

$$\phi_i = \frac{(ng_i + K_1 nl_i^2 + K_2 nb_i^3)}{\mu_i \times f_i} + \frac{1}{\mu_i}$$

onde ng_i é o número de jobs globais na fila de ready do processadore i, nl_i é o número de jobs locais interativos, nb_i é o número de jobs locais batch, f_i é a fração do tempo livre do processador i no último minuto, μ_i é a velocidade do processador i e K_1, K_2 são constantes. O propósito dos fatores quadrático e cúbico na expressão acima é desestimular o escalonador global no envio de tarefas paralelas a estações com carga de trabalho local. Entretanto, se a carga local não demanda muito tempo de processamento (geralmente, jobs interativos são I/O *bound*), o fator f_i é grande e contribui para reduzir o custo da estação. A combinação dos fatores f, n_g, n_l, and n_b oferece um bom indicador da carga de uma estação. Por exemplo, se o tamanho da fila é pequeno e f é pequeno, provavelmente a estação teria jobs *processor bound*. Neste caso, o custo da estação seria alto e o escalonador da rede não enviaria tarefas paralelas para ela.

A nível de estação, o escalonador local, implementado dentro do kernel, alocaria tempo de processador aos processos, de acordo com as regras de prioridade que privilegiem os jobs locais seqüenciais. Jobs locais interativos têm maior prioridade que jobs locais batch que, por sua vez, têm maior prioridade que jobs globais paralelos. As questões a serem investigadas aqui são, então, a performance deste escalonador em dois níveis e a interação entre a carga de trabalho paralela e a seqüencial. A próxima seção descreve o modelo de simulação usado para avaliar a performance do mecanismo de escalonamento proposto.

5 Modelo de Simulação

Um job paralelo pode ser visto como uma coleção de tarefas, quando da análise de sua performance. Tarefas são fragmentos computacionais que rodam seqüencialmente até o fim sem considerar o estado de outras tarefas. Um grafo de tarefas é uma representação do paralelismo de um job. Um atributo de uma tarefa que é relevante à sua performance é a sua *demanda de processador*, que é o tempo que um processador demora para executar a tarefa. Assim, uma carga de trabalho paralela pode ser caracterizada pelo seguinte conjunto de parâmetros: a estrutura do job paralelo, representada por um grafo de tarefas; as características do job, representada pela demanda de processador de cada tarefa; e a taxa de chegada dos processos. Esta caracterização de uma carga de trabalho paralela tem sido usada por muitos autores para a análise de diversos aspectos da performance de sistemas paralelos [1, 16, 19]. Há o problema associado com a utilização de grafos de tarefas com uma representação de jobs paralelos. Para algumas aplicações, o grafo de tarefas pode ser dependente dos dados e pode não ser conhecido a priori. O algoritmo *quicksort* paralelo é um bom exemplo deste tipo de aplicação. Enquanto o melhor caso do quicksort paralelo leva a um grafo totalmente balanceado, o pior caso é representado por um grafo degenerado.

Com o objetivo de representar uma ampla família de aplicações paralelas ou a execução da mesma aplicação para diferentes dados de entrada, é necessário introduzir alguma informação estatística que concerne às relações de precedências e às demandas de serviços das tarefas. Seja A a matriz de incidência do grafo de tarefas com elementos $a(i,j)$, onde i e j denotam tarefas. Quando $a(i,j) = 1$, $T_i \prec T_j$, que indica que a tarefa T_i deve ser finalizada antes que a execução de T_j possa ser iniciada. Por outro lado, $a(i,j) = 0$ indica que as tarefas i e j são independentes entre si. Como proposto em [1], $a(i,j)$ é variável aleatória independente definida como:

$$P\left[a\left(i,j\right)=1\right] = \pi \quad for \quad 1 \leq i < j \leq N \tag{1}$$

$$P\left[a\left(i,j\right)=0\right] = 1-\pi \quad for \quad 1 \leq i < j \leq N \tag{2}$$

$$P\,[a\,(i,j) = 0] = 1 \quad if \quad i \geq j \tag{3}$$

O parâmetro π indica a probabilidade de que exista uma dependência entre a tarefa i e a tarefa j, i.e., $a(i,j) = 1$. Considere um grafo de tarefas com N nodos no qual um arco do nodo i, $(i = 1, 2, ..., N)$ para o nodo j, $(j = 1, 2, ..., N)$ existe com probabilidade π. Assim, a estrutura do grafo de tarefas se torna arbitrária e acíclica (veja a equação (3)) e a densidade das relações de precedência entre tarefas é determinada por π.

Seja $\tilde{\mathbf{D}} = (\tilde{d_1}, \tilde{d_2}, \ldots, \tilde{d_N})$, onde $\tilde{d_i}$ é uma variável aleatória com média $\bar{d_i}$, que representa a demanda de processador da tarefa i. A taxa de chegada de processos paralelos ao sistema é caracterizada por um processo de Poisson com taxa de chegada λ. Portanto, uma tupla $(N, \pi, \tilde{\mathbf{D}}, \lambda)$ é usada para caracterizar uma carga de trabalho de jobs paralelos para diferentes aplicações. Por exemplo, famílias de tarefas com muito paralelismo podem ser representadas com valores baixos de π. Por outro lado, jobs com uma grande fração seqüencial têm um valor de π próximo de 1.

O modelo de sistema utilizado como base para os resultados de performance do esquema de escalonamento em dois níveis é composto por 20 diferentes estações heterogêneas conectadas por uma rede e uma de carga de trabalho composta por três tipos de jobs: grandes jobs paralelos, jobs locais interativos e jobs locais batch. Essas três classes são definidas em termos de demanda de processador. Na média, grande jobs paralelos demandam 1000 unidades de tempo, jobs locais batch demandam 100 e jobs locais interativos demandam 1 unidade de tempo. Assume-se que as variáveis aleatórias que representam a demanda de cada processador são distribuídas exponencialmente. Quando uma tarefa é alocada a uma estação, sua demanda é ajustada de modo a representar a velocidade do processador específico. Nas simulações realizadas, N recebeu o valor 20 e π o valor 0.5. A intensidade da carga de trabalho, λ, é obtida pela definição de uma valor que impõe uma carga específica ρ ao sistema.

O modelo de simulação consiste de um conjunto de programas C implementados utilizando-se a ferramenta de simulação SMPL [13]. O número de simulações efetuadas foi o necessário para obter um confiança de 90% com valores do intervalo de confiança dentro de 5% da média.

6 Resultados Numéricos

Esta seção analisa o impacto do mecanismo de escalonamento em dois níveis no que concerne à interação entre carga de trabalho seqüencial local e paralela global. Todos os resultados foram obtidos dentro do modelo descrito acima. A política de escalonamento usada como referência para propósitos de comparação foi a Fotografia Instantânea Modificada (FIM), implementada e testada no escalonador global distribuído da Seção 3. Os resultados obtidos por simulação do esquema em dois níveis foram então comparados com outros resultados de simulação da política FIM para a mesma carga de trabalho.

6.1 Performance da Carga de Trabalho Interativa

O primeiro caso consiste de um modelo com duas classes de carga de tabalho: jobs interativos seqüenciais e jobs paralelos. O fator aqui destacado é o impacto da execução de jobs paralelos globais na performance dos jobs interativos submetidos pelos proprietários das máquinas. A métrica de performance deste impacto é o tempo de execução normalizado, definido como o tempo médio de resposta de uma carga de trabalho interativa em conjunto com as aplicações paralelas e o tempo médio de resposta dos jobs interativos sem a presença de jobs paralelos.

A figura 3 mostra o tempo médio de execução normalizado versus a utilização do sistema (i.e.,

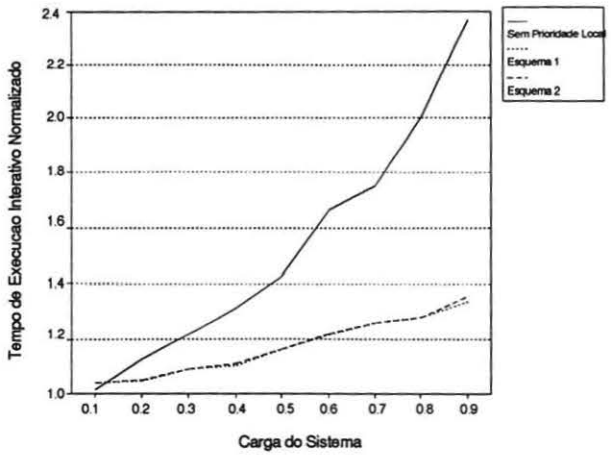

Figura 3: Tempo Médio de Execução Interativo Normalizado × Carga do Sistema

carga), para três diferentes mecanismos de escalonamento. O primeiro mecanismo corresponde ao escalonador global FIM. Neste caso, jobs interativos e paralelos competem por tempo de processador. Como esperado, a performance dos jobs interativos se deteriora rapidamente quando a carga do sistema aumenta, fato este que reflete o impacto que o proprietário de uma máquina sofre. Quando o sistema se torna carregado, existe uma contenção por processadores e a carga de trabalho paralela aumenta o tempo de resposta interativo aumenta por um fator maior que 100%. As outras duas curvas mostram os resultados obitidos utilizando o mecanismo de escalonamento em dois níveis. A nível de estação, jobs locais têm prioridade sobre jobs paralelos. A nível de rede, o escalonador distribuído aloca tarefas paralelas para estações. No esquema 1, a política FIM é usada. O esquema número 2 usa a nova função de escalonamento, que tem um fator quadrático que desencoraja jobs paralelos de irem para estações onde vários jobs locais estão em execução. Como pode ser visto pela figura 3, a melhoria da performance dos jobs locais é significativa. Em situações de carga alta, o fator de deterioração no tempo de resposta de jobs interativos não alcança 40%. Quase não há diferencas entre os resultados obtidos pelos dois esquemas. O fator prioridade local parece ser dominante no comportamento dos dois mecanismos de escalonamento.

6.2 Performance da Carga de Trabalho Paralela

O outro lado da moeda é o efeito do mecanismo de escalonamento em dois níveis na performance da carga de trabalho paralela. A figura 4 mostra o tempo de execução normalizado para os dois esquemas, que é definido como o tempo médio de execução da carga de trabalho paralela sem a presenção do mecanismo de dois níveis pelo tempo médio de execução para a mesma carga de trabalho com o mecanismo de escalonamento em dois níveis (i.e., com

prioridade para jobs locais). A boa surpresa é que a prioridade para os jobs locais não piora

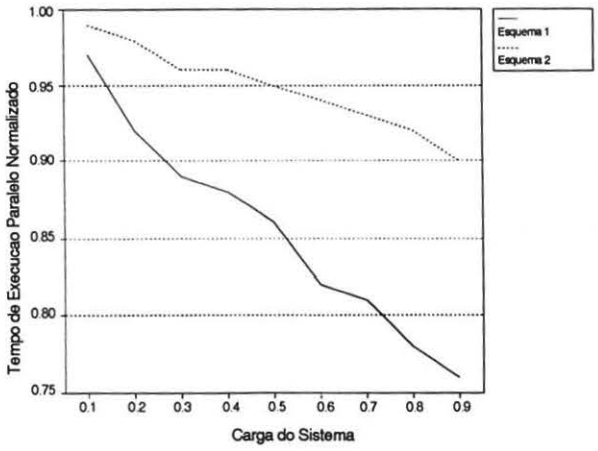

Figura 4: Tempo Médio de Execução Paralelo Normalizado × Carga do Sistema

muito o tempo da execução de jobs paralelos. A razão vem do fato que jobs interativos são normalmente pequenos e além disso demandam mais tempo de I/O que tempo de processador. No esquema 1, representado pela curva sólida da figura 4, foi utilizada a política FIM a nível de rede. Esta mostra uma degradação de 25% no tempo de execução numa situação de carga alta. A nova função de escalonamento melhora este resultado, como mostrado pela linha pontilhada do mesmo gráfico. A nova função "enxerga" a distribuição de jobs locais na rede e tenta enviar as tarefas para nodos que tem melhores chances de uma execução mais rápida. Neste esquema, a degeneração foi em torno de 10%.

6.3 Impacto na Carga de Trabalho Batch

É considerado agora o modelo com três tipos de carga de trabalho: interativa, batch, e paralela. É examinada aqui o impacto causado no sistema pela introdução de jobs batch locais que têm prioridade sobre jobs paralelos. Ao contrário de jobs interativos, a carga de trabalho batch demanda um fator considerável de tempo do processador. A figura 5 exibe o tempo de execução normalizado como função da carga do sistema. A normalização é definida pela divisão do tempo médio de execução do classe do job com o mecanismo de escalonamento em dois níveis sobre o tempo de execução da mesma carga de trabalho sem o mecanismo de escalonamento em dois níveis. Os jobs batch sofrem um impacto muito maior da interferência de jobs paralelos que os jobs interativos. Um job batch requer muito mais tempo de processador e, como conseqüência, a contenção por tempo de processador com jobs paralelos é grande. Para ambas as classes de jobs locais, o uso de um esquema de prioridade local favorece a performance total do proprietário da máquina.

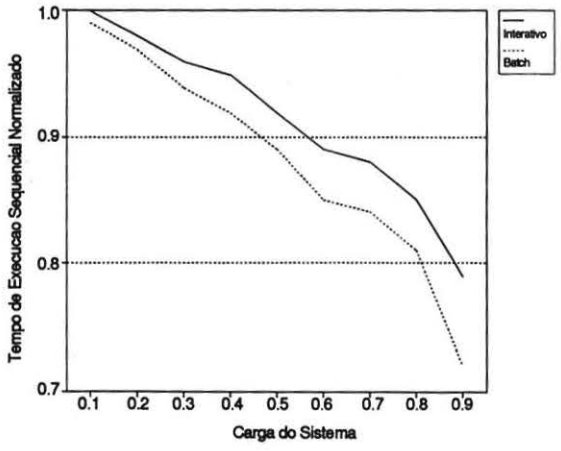

Figura 5: Tempo Médio de Execução Seqüencial Normalizado × Carga do Sistema

7 Conclusão

Este artigo tratou o problema do escalonamento de jobs paralelos numa rede heterogênea de estações. Mecanismos de escalonamento para redes de estações de trabalho são responsáveis pelo gerencimanto de recursos, alocação e balanço de carga. Primeiro, foram apresentados resultados de um escalonador distribuído executado num ambiente real. Foi obtida uma redução substancial do tempo de execução de um mix de aplicações paralelas reais. Embora tais resultados fossem satisfatórios, a degradação do tempo de resposta dos jobs interativos causou reclamações dos proprietários da estações. Foi então proposto um mecanismo de escalonamento em dois níveis com dois objetivos: reduzir o tempo de execução das aplicações paralelas e minimizar os efeitos da carga de trabalho paralela nos jobs locais executados pelos prorietários da máquinas. Como uma política de escalonamento em dois níveis requereria modificações no kernel de vários sistemas operacionais, sua implementação se tornaria não realística. Assim, foi construído um modelo de simulação da rede de estações e do mecanismo de escalonamento.

O artigo fez uma análise, por simulação, da performance deste mecanismo de escalonamento. Foi observado que o esquema minimizou os efeitos da resposta de jobs interativos sem impor muitas penalidades na performance de grandes aplicações paralelas. O escalonador em dois níveis que atribui prioridade para jobs locais é crítico para a performance total. Ele acelera a execução de jobs interativos, mas não atrasa a execução de jobs maiores. Em suma, foi mostrado que o mecanismo de escalonamento proposto melhora a performance da rede de estações e reduz os conflitos causados pela interação entre jobs interativos e batch locais e jobs paralelos globais.

Referências

[1] Almeida, V.A.F., Vasconcelos, I.M., Árabe, J.N.C. and Menascé, D.A., "Using Random Task Graphs to Investigate the Potential Benefits of Heterogeneity in Parallel Systems", *Proceedings of the IEEE/ACM Supercomputing'92 Conference*, Minneapolis, MN, 1992.

[2] Almeida, V.A.F., Árabe, J.N.C., Loures, E.F. and Rímolo, G.S., "Scheduling Parallel Jobs on a Cluster of Heterogeneous Workstations", *Proceedings of the High Performance Computing Conference '94*, Singapore, September 1994, pp. 103-108 .

[3] Anderson, T.E., Culler, D.E., Patterson, D.A., and the NOW team, "A Case for NOW (Networks of Workstations)", *IEEE Micro*, to appear, 1995.

[4] Beguelin, A., Seligman, E. and Starkey, M., "Dome: Distributed Object Migration Environment", Technical Report CMU-CS-94-153, School of Computer Science, Carnegie Mellon University, May 1994.

[5] Butler, R. and Lusk, E., "User's Guide to the P4 Programming System", Technical Report ANL–92/17, Argonne National Laboratory, 1992.

[6] Carriero, N. and Gelernter, D., "How to Write Parallel Programs: A Guide to the Perplexed", *ACM Computing Surveys*, pp. 323–357, September 1989.

[7] Casas, J., Konuru, R., Otto, S.W., Prouty, R. and Walpole, J., "Adaptive Load Migration Systems for PVM", *Supercomputing '94*, Washington DC, November 1994, pp. 390–399.

[8] Eager, D., Lazowska, E. and Zahorjan J., "A Comparison of Receiver Initiated and Sender Initiated Dynamic Load Sharing", *Performance Evaluation*, Vol. 6, No. 1, April 1986.

[9] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., *PVM: Parallel Virtual Machine — A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, 1994.

[10] Kaplan, J.A. and Nelson, M.L., "A Comparison of Queueing, Cluster and Distributed Computing Systems", Technical Report NASA TM 109025 (Revision 1), NASA Langley Research Center, June 1994, pp. 50.

[11] Kolawa, A., "The Express Programming Environment", *Workshop on Heterogeneous Network-Based Concurrent Computing*, Tallahassee, FL, October 1991.

[12] Litzkow, M. and Livny, M., "Experience with the Condor Distributed Batch System", *IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, October 1990.

[13] MacDougall, M.H., *Simulating Computer Systems: Techniques and Tools*, The MIT Press, 1987.

[14] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", Technical Report CS–94–230, Computer Science Department, University of Tennessee, Knoxville, TN, April 1994

[15] "C-Linda Reference Manual", Scientific Computing Associates, Inc., 1992.

[16] Sevcik K., "Application Scheduling and Processor Allocation in Multiprogrammed Paralell Systems", TR CSRI-282, University of Toronto, 1993.

[17] Siegell, B. and Steenkiste, P., "Automatic Generation of Parallel Programs with Dynamic Load Balancing", *Proceedings of the Third International Symposium on High-Performance Distributed Computing*, IEEE, San Fransisco, August 1994.

[18] Weinrib A. and Shenker S., "Greed is not enough: adaptive load sharing in large heterogeneous systems", *Proceedings of the IEEE INFOCOM*, 1988.

[19] Zahorjan, J. and McCann, C., "Processor Scheduling in Shared Memory Multiprocessors", Proceedings of ACM SIGMETRICS Conference, May 1990.