Implementação de um Escalonador de Tarefas Distribuído para Redes de Estações de Trabalho

Marco Aurélio de Souza Mendes – corelio@dcc.ufmg.br[+]
Virgílio Augusto Fernandes Almeida – virgilio@dcc.ufmg.br[*]

Departamento de Ciência da Computação - UFMG
Caixa Postal 702 - Belo Horizonte - Minas Gerais - Brasil

Resumo

Redes de estações de trabalho são um ambiente adequado para processamento paralelo. O recente desenvolvimento tecnológico aliado ao crescimento do poder de processamento das estações de trabalho torna estas máquinas adequadas a uma ampla gama de usuários e aplicações, executando desde programas paralelos computacionalmente intensivos até tarefas interativas como edição de texto. Entretanto, problemas como ociosidade, má distribuição de carga e interferência entre classes de carga de trabalho distintas levam a problemas de desempenho das tarefas submetidas à estas máquinas. Baseado nestes problemas, este trabalho propõe um escalonador de tarefas para redes de estações de trabalho com os seguintes propósitos: reduzir a ociosidade em uma rede de estações através da redistribuição de carga dos nodos mais carregados para os nodos menos carregados, minimizar a interferência entre as diversas classes de carga de trabalho (jobs interativos, jobs batch e jobs paralelos) e utilizar a heterogeneidade normalmente presente numa rede de estações. Decorre deste trabalho que o escalonador proposto e implementado apresenta um bom desempenho para escalonamento de tarefas em redes de estações de trabalho. Ganhos sobre o PVM são obtidos, ao mesmo tempo que a interferência entre classes de carga de trabalho é minimizada.

Abstract

Networks of powerful workstations are an adequate means for parallel processing. The recent development in networks technologies as well as the enourmous growth in the processing power of workstations make these machines suitable for a widespread gamma of applications. However, problems like idleness, load imbalacing and competition among different classes of jobs decrease the performance in a network of workstations. Based on these facts, this work proposes a task scheduler for networks of workstations with the following goals: to reduce the idleness in the network redistributing the load among nodes; to minimize the competition among different classes of jobs (interactive, batch and parallel) and to use the heterogeneity found in this kind of environment. A new operating system scheme is proposed in order to minimize competition among diferent classes of jobs. Simulation and implementation results are given. As a consequence of this work, a good performance task scheduler is obtained. Gains over the plain PVM are achieved, as long as the competition among different classes of jobs is minimized.

[+] Parcialmente suportado pelo CNPq
[*] Parcialmente suportado pelo CNPq

1 Introdução

1.1 Motivação

Redes de estações de trabalho são um ambiente adequado para processamento paralelo. O desenvolvimento da tecnologia de redes (ATM, FDDI) aliado ao enorme crescimento do poder de processamento das estações de trabalho permitem a resolução de problemas antes só tratáveis em supercomputadores. Em verdade, o rápido crescimento do poder de processamento das estações de trabalho não acontece por acaso, mas devido a um alto investimento em engenharia e manufatura. Tais investimentos, entretanto, são amortizados pelo grande número de unidades vendidas. Por outro lado, supercomputadores e mainframes vendem poucas unidades. Isto provoca uma defasagem em avanços de desempenho além de provocar altos custos por unidade vendida. A razão preço/desempenho para estações de trabalho melhora 80% ao ano contra 25% para supercomputadores [Anderson95]. Estes e outros fatores estão mudando o perfil observado atualmente. Ao invés de pequenos computadores para uso interativo e grandes computadores para aplicações intensivas, observa-se o uso de redes de estações de trabalho para todas as necessidades dos usuários de computadores.

A despeito desta tendência, nota-se que estações de trabalho ficam ociosas a maior parte do tempo. Mesmo em horários de pico, observa-se a existência de máquinas livres. De fato, enquanto algumas poucas estações estão sobrecarregadas, várias outras estão subutilizadas. Tal fato pode ser explicado pela natureza estocástica da carga de trabalho normalmente presente em sistemas distribuídos [Livny82]. A figura 1 mostra a utilização de uma máquina SUN-SPARC 2 do DCC (Departamento de Ciência da Computação da UFMG) em um período de 24 horas. Tais valores, que são parte de um experimento que foi conduzido monitorando-se esta máquina durante um período de cinco dias, mostram uma baixa utilização do processador. Baseado nos dados coletados nos cinco dias, observa-se um fator de ociosidade de 50,04% da máquina monitorada, se considerarmos uma máquina ociosa como uma com carga abaixo de 5% de seu poder de processamento. O experimento mostra também que 80% do tempo a máquina está com carga abaixo de 20% de sua capacidade.

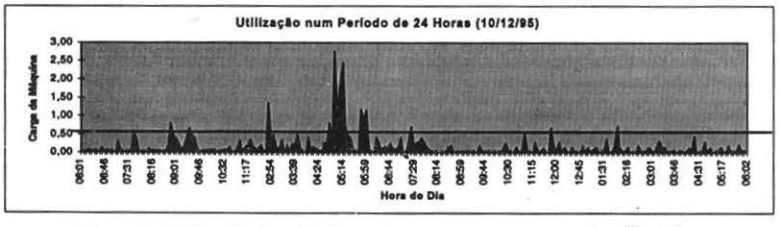

Figura 1: Utilização da máquina **mica** (SPARC 2) em um dia específico

Fatores como ociosidade e má distribuição da carga provocam alguns problemas. A ociosidade implica diretamente em um desperdício de recursos, visto que estes não estão sendo utilizados efetivamente. A má distribuição da carga leva em última análise a um aumento do tempo de resposta das tarefas submetidas na rede, pois tais tarefas utilizam mal os recursos da rede, aglutinando-se em algumas poucas estações enquanto deixam várias outras ociosas. Deste

modo, faz-se necessário realizar um escalonamento de tarefas adequado, de modo a minimizar tais problemas.

Atualmente, vários projetos tentam lidar com o problema do escalonamento de tarefas em termos mais gerais, como por exemplo o projeto NOW [Anderson95] ou o projeto DOME [Árabe94]. Tais projetos tentam prover desempenho de supercomputadores para aplicações paralelas enquanto buscam um melhor aproveitamento dos recursos do sistema - aí incluído balanço de carga. No contexto do DCC-UFMG, algumas pesquisas já foram realizadas para tentar tratar o problema do escalonamento de tarefas em sistemas distribuídos. O trabalho [Almeida92] procura utilizar informações das estrutura de um job paralelo a fim de se beneficiar da heterogeneidade normalmente presente em redes de estações. Um escalonador global distribuído para aplicações paralelas PVM já foi construído [Loures94]. Embora este escalonador apresente bons ganhos sobre aplicações PVM puras, vários aspectos do problema do escalonamento não são tratados, como a interação entre as diversas classes de carga de trabalho, como por exemplo jobs paralelos e jobs seqüenciais. Num trabalho mais recente [Almeida95] pesquisas foram realizadas no sentido de prover bom desempenho para programas paralelos ao mesmo tempo que mantivessem o tempo de resposta de programas seqüenciais interativos. Este trabalho, entretanto, fornece somente resultados obtidos por simulação.

A proposta de nosso trabalho, baseada numa continuação dos trabalhos supracitados, é expandir as pesquisas de escalonamento de tarefas em redes de estações de trabalho realizadas no DCC-UFMG. Isso inclui a implementação real de um escalonador distribuído de jobs com os seguintes objetivos: reduzir o tempo de execução de grandes jobs paralelos; minimizar os efeitos da execução de grandes jobs paralelos no desempenho de jobs interativos de usuários locais; e alocar as estações mais apropriadas a diferentes partes de um job paralelo; melhor combinando os recursos oferecidos pela rede e as necessidades de computação.

2 A Arquitetura de um Escalonador Distribuído

A figura 2 define o esquema lógico do escalonador em dois níveis. A função primeira é o escalonamento de jobs paralelos. O escalonador pode ser visto como uma entidade lógica única, embora fisicamente distribuída, que possui o conhecimento de todos os jobs paralelos submetidos ao sistema. Tal abordagem, logicamente centralizada, denominada escalonamento a nível de rede, é utilizada primordialmente para evitar conflitos entre várias aplicações paralelas.

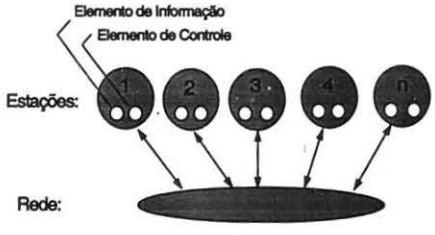

Figura 2: Esquema lógico do escalonador em dois níveis

A estratégia do algoritmo distribuído é baseada na idéia de que qualquer job recém-chegado ao sistema tem suas tarefas atribuídas aos nodos com menor índice de carga, através

da comparação da carga local de cada nodo do sistema distribuído. Isso é alcançado através de duas entidades presentes no sistema. A primeira entidade – Elemento de Informação – tem por função coletar o valor dos parâmetros relevantes ao cálculo da carga e transmitir esta informação aos demais nodos do sistema. A outra entidade, denominada Elemento de Controle, é responsável por atribuir as tarefas aos processadores, baseado em uma estrutura de dados local que armazena os valores da carga de cada nodo do sistema, calculados de acordo com os valores dos índices enviados pelos Elementos de Informação.

A entidade Elemento de Informação é implementada através de dois *daemons* presentes em cada nodo do sistema. O primeiro *daemon* é responsável pela coleta da carga local e envio desta informação aos demais nodos do sistema. O segundo *daemon* é responsável pelo recebimento das informações do primeiro *daemon* e pela atualização de uma tabela (Tabela de Carga) nas qual ficam armazenadas as informações de carga de cada nodo do sistema. A entidade Elemento de Controle é implementada na forma de uma biblioteca de funções ligada aos programas em tempo de execução. Este elemento consulta a Tabela de Carga do nodo onde a tarefa se originou antes de decidir qual a melhor estação para escalonar esta tarefa. Cada nodo do sistema mantém uma versão local da tabela de situação de carga residente em uma área de memória compartilhada que pode ser consultada por qualquer tarefa de um programa paralelo ao ser iniciada. É importante ressaltar que as tabelas de carga de cada nodo não possuem necessariamente a mesma informação, devido aos atrasos das mensagens enviadas pelos elementos de informação e pela freqüência das trocas de informação.

2.1 Competição entre Cargas de Trabalho Distintas

O esquema apresentado parece apropriado para jobs paralelos. No entanto, a carga de trabalho presente em redes de estações de trabalho é composta por três classes: grandes jobs paralelos, jobs seqüenciais interativos (e.g.: listagem de diretórios, edições de texto) e jobs batch. Ignorar este fato pode levar a uma indesejável interferência entre tais classes de carga de trabalho, com conseqüências graves. Por um lado, jobs paralelos podem interferir na execução de jobs interativos. Na outra ponta, existe a interferência entre jobs batch e jobs paralelos. No contexto do nosso trabalho, o 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 de programas paralelos, enquanto procure não degradar o tempo de resposta de jobs interativos e jobs batch. A escolha de índices de carga apropriados a um sistema qualquer é uma difícil tarefa. Em particular, vários trabalhos tratam especificamente da escolha de índices relevantes a funções de cômputo de carga em redes de estações de trabalho. Os trabalho de [Ferrari88] mostra que índices baseados no tamanho da fila apresentam bons ganhos. Como tais resultados são válidos somente para sistemas homogêneos, alguma informação de heterogeneidade das máquinas do sistema deve fazer parte também da função por nós utilizada. Baseada nestes critérios, a seguinte função é apresentada no escalonador proposto por [Loures94].

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

Esta função, adaptada das funções propostas em [Weinrib88] para escalonamento de lotes de jobs em sistemas de filas heterogêneas, usa os seguintes índices de carga:

- μ_i : Velocidade do servidor i em relação a um processador adimensional;
- f_i: fração de tempo livre do servidor i no último minuto;

- n_i: número de tarefas na fila ready no processador i.

Ao ser iniciada uma tarefa, esta função de custo é avaliada para cada nodo e o nodo com menor valor da função de custo é eleito para processar a tarefa. O parâmetro f_i dá à função uma natureza adaptativa, i.e., embora a função de custo esteja ligada à aceitação de novas tarefas pelo servidor, o valor de f_i é atualizado para futuros cálculos. O termo adaptativo é utilizado para indicar que a função é explicitamente dependente de sua história (memória), ao invés de simplesmente computar as informações instantâneas de estado. O termo μ_i fornece a relação entre a velocidade (jobs/seg) do servidor i e a velocidade de um servidor padrão[1]. O termo n_i é um indicativo do número de tarefas na fila do processador.

Esta função, entretanto, não leva em conta a presença de outras classes de carga de trabalho. A seguinte função, então, é proposta, baseada neste fato.

$$F_i = \frac{\left(n_i + k1 \times g_i^2 + k2 \times b_i^3\right)}{\mu_i \times f_i} + \frac{1}{\mu_i}$$

Nesta função, os termos μ_i e f_i permanecem com o mesmo significado que na fórmula anterior. O termo n_i mede agora o número de jobs interativos na fila ready do processador. Dois novos parâmetros são adicionados, g_i e b_i. O primeiro é um indicativo do número de tarefas de jobs paralelos na fila ready do processador (tarefas paralelas) enquanto o segundo mede o número de jobs batch presentes na mesma fila. $k1$ e $k2$ são constantes associadas a estes dois novos parâmetros, utilizadas para indicar o peso relativo que cada tarefa de cada classe de carga de trabalho acarreta ao sistema. A inclusão destes dois novos parâmetros tem por objetivo desestimular o envio de tarefas paralelas para estações onde hajam usuários locais executando tarefas interativas e estações onde existam jobs batch, assim como os fatores quadrático e cúbico associados a g_i e b_i. Deste modo, usuários locais não são perturbados por programas paralelos nem estes têm sua execução influenciada pelos jobs batch. Valores muito altos de $k1$ e $k2$ aumentam o nível de conservadorismo do escalonador. Valores de $k1$ e $k2$ entre 2 e 10 são escolhas razoáveis, já conseguindo diferenciar as diversas classes de carga de trabalho. De qualquer modo, tais constantes devem ser ajustadas de modo empírico, até que consigam expressar o peso relativo que cada classe de tarefa deva representar no sistema em questão.

2.2 Escalonamento em Dois Níveis

Suponha a situação onde uma determinada máquina de uma rede de estações não possua usuário conectado. Nesta situação, esta máquina torna-se uma boa candidata a receber tarefas de jobs paralelos. Se uma determinada tarefa for escalonada a esta máquina e logo após um usuário local conectar-se a esta máquina, problemas ocorrerão. O usuário local sentirá a interferência de uma carga de trabalho estranha e qualquer tarefa escalonada por ele que precise de maior poder computacional competirá com a tarefa externa, degradando o desempenho de ambas. Neste ponto, é importante tomar uma decisão, visto que não só é importante escalonar bem as tarefas, mas tomar alguma atitude quando o proprietário da máquina retornar. Políticas conservadoras como a do Condor [Litzkow90] simplesmente suspendem e migram todas as tarefas remotas que porventura estejam executando. O custo de migrar tarefas é, entretanto, elevado e pode se tornar até mesmo inviável se máquinas muito heterogêneas estiverem

[1] Um servidor padrão é aquele utilizado como base de comparação entre todas as máquinas da rede de estações. Deste modo, uma tabela de velocidades é obtida, tabela esta usada como informação de heterogeneidade na função de custo.

fazendo parte da rede. Por outro lado, deixar tarefas paralelas executando nesta máquina pode causar perdas inaceitáveis no tempo de resposta das aplicações interativas dos usuários locais, o que pode levar inclusive à remoção desta do pool de máquinas disponíveis.

Uma possível abordagem consiste na idéia de que tarefas remotas tenham, efetivamente, prioridades menores do que tarefas geradas localmente (esquema denominado Prioridade Remota Menor). A política de serviço do processador do UNIX™ é mantida (*round robin with multilevel feedback*), i.e., o *kernel* aloca a um processo um pequeno *quantum* do processador, preeempta o processo quando o tempo expira, e o coloca de novo em uma das filas de prioridade. No UNIX™, a prioridade de um processo é determinada pela seguinte fórmula:

priority = – [("recent CPU usage"/constant) + (prioridade base) + (valor nice)]

O valor mais alto de prioridade para um processo no modo usuário é zero. Um processo se torna menos prioritário, de acordo com esta fórmula, se ele utilizou muita CPU recentemente, se a sua prioridade base é alta ou através do comando nice[2]. A idéia consiste, então, em atribuir uma prioridade base elevada para processos remotos e penalizar a utilização recente de CPU em fatores mais altos dos que aqueles aplicados a processos comuns. Espera-se, com isso, que processos remotos não degradem os usuários locais devido a sua prioridade reduzida, mas tenham completo acesso à CPU da máquina quando esta estiver ociosa.

Em resumo, o esquema proposto de escalonamento é realizado em dois níveis (política aqui denominada Escalonamento em Dois Níveis – EDN). Em um primeiro nível, o escalonador global escalona tarefas de jobs paralelos às estações de acordo com a função de custo apresentada anteriormente. No nível de estação, o escalonador local é alterado de modo que o processador priorize tarefas locais via a adoção de prioridades diferenciadas a processos locais e processos remotos (Prioridade Remota Menor), garantindo, deste modo, a não degradação do tempo de execução das tarefas locais.

3 Implementação do Escalonador em Dois Níveis

Para implementar o *daemon* responsável pela coleta e troca de informações, um processo é criado em cada estação assim que o escalonador é inicializado. Tais processos (denominados *balance*) se comunicam via uma memória compartilhada presente em cada estação, mantendo desta forma as informações relevantes ao escalonador (número de usuários locais na estação, velocidade do processador, número de processos na fila ready do processador e fração livre do processador no último minuto). Para inicializar o escalonador, portanto, basta executar um processo denominado *init_twolevel*. Este processo é responsável por criar as memórias compartilhadas nas estações da máquina e disparar os processos *balance* em cada uma destas máquinas. Os processos *balance*, então, trocam informações em intervalos de tempo pré-determinados. Tais intervalos não podem ser muito grandes pois a informação não refletiria o estado atual de cada estação. Por outro lado, intervalos muito pequenos causariam um grande *overhead* ao sistema. O valor utilizado efetivamente como intervalo de tempo para troca de informações foi de 300 ms. Tal valor, puramente empírico, obtido de [Loures94], fornece uma boa relação custo benefício na rede do DCC-UFMG, onde o escalonador foi implementado.

O escalonador é construído sobre o PVM versão 3.1. O PVM foi escolhido por se tratar de um ambiente amplamente difundido no DCC-UFMG e de utilização relativamente fácil. Deste modo, aplicações PVM já existentes podem ser facilmente portadas para o escalonador em dois

[2] O valor do comando nice deve sempre ser positivo para usuários comuns. Entretanto, o superusuário pode atribuir valores negativos ao nice, de modo a tornar um processo mais prioritário.

níveis. As modificações necessárias para portar aplicações PVM para aplicações construídas no escalonador proposto são simples. No código fonte (arquivos .c), basta trocar a função de criação de processos *pvm_spawn* pela nova função de criação de processos *sch_spawn*. Trocada esta função, basta ligar os programas objetos (arquivos .o) com a biblioteca *lib_twolevel.a* para gerar um programa controlado pelo escalonador em dois níveis. Esta biblioteca (*lib_twolevel.a*) é responsável por inserir nos programas que utilizem a função *sch_spawn* o elemento de controle. Quando o programa é instanciado, qualquer chamada à função *sch_spawn* faz com que o elemento de controle atue, consultando as informações mantidas pelos elementos de informação de modo a escolher a estação menos carregada para executar uma nova tarefa.

3.1 Desempenho para Aplicações Paralelas

– Múltiplos Fork and Join

Tal aplicação é interessante por modelar muitas aplicações reais (e.g.: segmentação de imagens, multiplicação de matrizes, cálculo de harmônicos). Considerando que cada tarefa de uma aplicação MFJ seja computacionalmente semelhante às outras, dois aspectos principais definem a aplicação: o número de níveis (profundidade) e a número de tarefas geradas em cada nível (largura). Para os experimentos realizados, foram modeladas aplicações MFJ de maior profundidade com diferentes números de tarefas geradas por nível. Cada tarefa da aplicação MFJ consome um tempo considerável de CPU, modelando aplicações de granularidade ditas grão grosso. A figura 3 mostra os resultados obtidos para aplicações MFJ de profundidade 16 e largura variável (1 a 8 tarefas por nível). As condições de experimento foram as mais próximas do possível para as aplicações PVM padrão e as aplicações com o escalonador em dois níveis. Os testes foram realizados em horários que existiam usuários locais conectados às máquinas. Os resultados mostram um aumento do tempo de resposta à medida que o número de tarefas por nível aumenta para ambas as aplicações. Tal comportamento pode ser explicado pela conjunção de dois fatores: a profundidade alta do grafo MFJ e a demanda alta de cada tarefa.

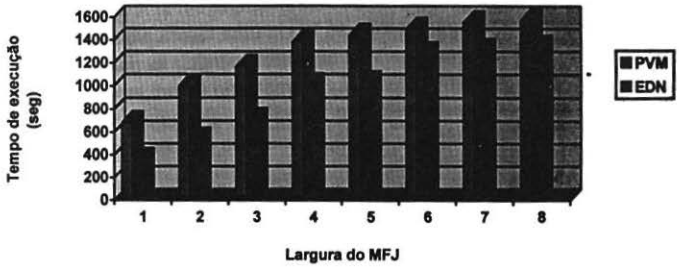

Figura 3: Tempo de execução de uma aplicação MFJ

Desta forma, quanto maior o número de tarefas por nível, maior a probabilidade de que uma carga externa afete o tempo de execução de uma tarefa e assim o tempo total de execução da aplicação. Observa-se um comportamento superior (no que diz respeito ao tempo de execução) para o escalonador EDN. Observa-se também um crescimento maior do tempo à medida que aumenta o número de tarefas por nível para o escalonador EDN. Tais fatos podem ser explicados devido ao fator heterogeneidade. Quando existem poucas tarefas por nível, o escalonador sempre aloca uma máquina mais rápida para executar esta tarefa enquanto o PVM

utiliza uma lista circular para escolher onde executar a próxima tarefa. Isso determina fatores de ganho de quase 100% para poucas tarefas (largura = 1) do EDN sobre o PVM padrão. À medida que o número de tarefas alcança o número de máquinas disponíveis (oito máquinas), a influência do fator heterogeneidade diminui. Dessa forma, os ganhos do EDN sobre o PVM são menores para um número maior de tarefas por nível.

3.2 Impacto nos Usuários Locais - PVM

A figura 4 mostra o impacto sofrido por usuários locais quando havia uma carga de trabalho paralela sendo executada com o PVM puro. O percentual de degradação representa a razão entre o tempo de execução de uma tarefa interativa na presença de jobs paralelos e o tempo de execução desta mesma tarefa na ausência de uma carga paralela. Algumas atividades típicas de usuários de jobs interativos foram monitoradas, como por exemplo listagem de diretórios (ls), formatação de textos (tex) e cópia de arquivos (cp). Os testes foram realizados em uma SPARC-SLC com 32 MB de memória e foram repetidos de 30 a 100 vezes para efeitos de maior confiança estatística.

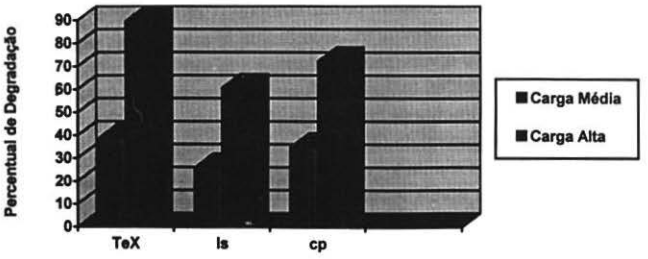

Figura 4: Impacto em Usuários Locais

A carga gerada através do PVM consistia de uma aplicação MFJ. A aplicação PVM foi testada para duas situações de carga: quando só havia uma tarefa paralela na máquina monitorada (carga média) e quando havia mais de uma tarefa paralela na máquina monitorada (carga alta). A primeira aplicação interativa foi a formatação de um arquivo de 5K pelo formatador de textos TEX. A segunda aplicação foi a listagem do conteúdo do diretório /usr/tmp da máquina monitorada. A terceira aplicação consistia da cópia de uma arquivo de 200K entre diretórios da máquina monitorada.

Os resultados obtidos mostram uma degradação do desempenho das tarefas interativas quando aplicações paralelas estão presentes. É importante notar que a natureza destes resultados é mais qualitativa do que quantitativa visto que o percentual de degradação para as aplicações interativas é altamente variável, dependendo de inúmeros aspectos como o tamanho dos arquivos manipulados, a configuração da máquina (memória, cache, processador), o tipo de aplicação interativa, entre outros.

3.3 Impacto nos Usuários Locais - Escalonador em Dois Níveis

Nos testes realizados procurou-se determinar o impacto do escalonador em dois níveis nos usuários locais. Para isso, a carga das oito máquinas do pool foi monitorada durante a execução específica de uma aplicação MFJ. Nesta aplicação um usuário local foi inserido em uma máquina, enquanto nas demais máquinas não haviam usuários conectados. Dessa forma, foi

possível avaliar o impacto sofrido por este usuário quando da execução de uma aplicação paralela. Dois gráficos de carga foram obtidos, então. O primeiro foi obtido quando a aplicação estava sobre domínio do PVM padrão. O segundo gráfico foi obtido executando-se a aplicação com o escalonador em dois níveis (EDN).

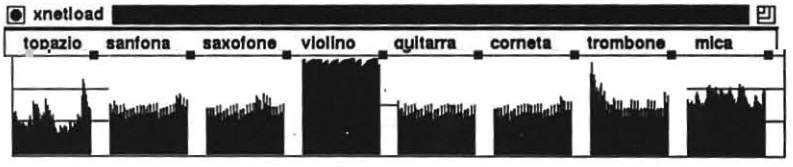

Figura 5: Amostragem de carga com o PVM

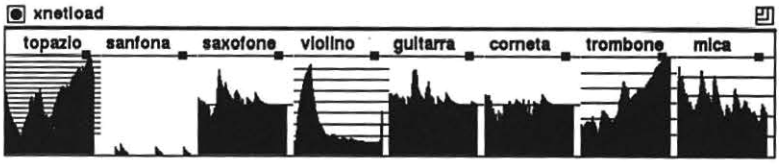

Figura 6: Amostragem de carga com o EDN

A máquina **sanfona** possuía um usuário conectado enquanto as demais não possuiam nenhum usuário conectado. Durante a amostragem de carga mostrada nos dois gráficos acima, nota-se que o escalonador em dois níveis (EDN) evita a máquina **sanfona**. Isso leva a baixos valores para a carga local desta, não degradando, desta forma, o desempenho das tarefas interativas.

Um outro fator interessante observado é que o PVM, baseado em sua política de lista circular para decidir onde escalonar a próxima tarefa, provê um carga igualmente distribuída entre as estações (figura 5). Já o escalonador EDN usa as estações mais rápidas mais vezes devido à sua função de custo. Tal fato pode ser visto na amostragem da máquina **topazio** (figura 6), uma máquina mais poderosa (SPARC Server 630), cuja utilização é maior do que a das demais máquinas.

4 Conclusões

Este trabalho apresentou o projeto e arquitetura de um escalonador distribuído e cooperativo para uma rede de estações de trabalho. O escalonador proposto tem por objetivo prover ganhos para programas paralelos ao mesmo tempo que não degrade o tempo de resposta de usuários locais executando jobs interativos.

Numa avaliação geral do escalonador EDN, percebe-se uma bom desempenho sobre aplicações PVM. Os ganhos médios para o escalonador EDN foram em torno de 20% para o *mix* de aplicações que fizeram parte do experimento. O escalonador em dois níveis apresentou ganhos mais expressivos quando o número de tarefas no sistema é menor. Isso é devido à maior probabilidade de existirem usuários de jobs interativos executando aplicações que possam degradar o desempenho de tarefas paralelas quando o número de tarefas paralelas no sistema é maior. Além disso, nota-se que uma variação mais abrupta da chegada de tarefas ao sistema fornece melhores resultados para o escalonador em dois níveis (e.g.: aplicação MFJ).

Escalonadores que utilizem funções de custo com índices baseados em heterogeneidade de estações reagem melhor a estas variações abruptas de carga pois as tarefas são de alguma forma aglutinadas nas estações mais rápidas. Políticas como as usadas pelo PVM (lista circular) distribuem eqüitativamente as tarefas entre as estações, o que faz o desempenho geral ser dependente da máquina mais lenta do pool. Já aplicações reais trazem vários outros aspectos para análise. Problemas como a fração seqüencial da aplicação (Lei de Amdhal) e o overhead de comunicação entre estações limitam o *speed-up* da aplicação. Dessa forma, o ganho do EDN sobre aplicações PVM deve ser medido para a parte paralelizável da aplicação. Baseado neste fato, observamos bons ganhos para esta aplicação quando o EDN é usado. O EDN tenta alocar as estações mais rápidas ou menos utilizadas recentemente na esperança de obter um menor tempo de resposta. Tal abordagem mostra-se aplicável, o que depõe a favor da política e mecanismos introduzidos pelo EDN.

Referências

[Almeida95] Almeida, V.A.F., Árabe, J.N.C, Mendes, M.A.S and Oliveira, A.A., *"A Two Level Scheduling Scheme for Paralell Jobs on a Networks of Workstations"*, XV International Conference of the Chilean Society in Information Science, Arica, Chile, 1995.

[Almeida92] Almeida, V.A.F., Árabe, J.N.C, Vasconcelos, I.M. e Menascé, D.A., *"Using Random Task Graphs to Investigate the Potential Benefits of Heterogeineity in Parallel Systems"*, Proceedings of the IEEE/ACM SuperComputing'92, Minneapolis, MN, 1992.

[Anderson95] Anderson, T.E., Culler, D.E., Patterson, D.A., e the NOW Team, *"A Case for NOW (Networks of Workstations)"*, IEEE Micro, February 1995, pp 54-64.

[Árabe94] Árabe, J.N.C., Beguelin, A., Lowekamp, B., Seligman, E., Starkey, M. and Stephan, P., *"Dome: Parallel Programming in a Heterogeneous Multi-User Environment"*, Technical Report CMU–CS–95–137, School of Computer Science, Carnegie Mellon University, Pittsburgh, April 1995.

[Ferrari88] Ferrari, D. and Zhou, S., *"An Empirical Investigation of Load Indices for Load Balancing Applications"*, Proceedings of Performance '87, The 12th International Symposium on Computer Performance Modeling, Measurement and Evaluation, North Holland Publishers, Amsterdam, 1988, pp. 515-528.

[Litzkow90] Litzkow, M. and Livny, M., *"Experience with the Condor Distributed Batch System"*, IEEE Workshop on Experimental Distributed Systems, Huntsville, AL, October 1990.

[Loures94] Loures, E.F., *"Balanço de Carga Para Programas Paralelos em Redes Heterogêneas de Workstations"*, Tese de Mestrado do Departamento de Ciência da Computação da UFMG, Março de 1994.

[Livny82] Livny, M., and Melman, M., *"Load Balancing in Homogeneous Broadcasting Distributed Systems"*, Performance Evaluation Review, Vol.11, no. 1, pp. 47-55, April 1982.

[Weinrib88] Weinrib, A., Shenker, S., *"Greed Is Not Enough: Adaptive Load Sharing in Large Heterogeneous Systems"*, Proceedings IEEE INFOCOM'88, pp 986-994, 1988.