O Conceito de Working Set de Processadores em Sistemas Paralelos com Multiprogramação

Luiz Chaimowicz* José Nagib Cotrim Árabe†

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Caixa Postal 702 – Belo Horizonte – MG – CEP 30.161-970

Resumo

Este trabalho realiza um estudo sobre o conceito de *Working Set* de Processadores em sistemas paralelos multiprogramados e discute a implementação de duas políticas de escalonamento baseadas nesse conceito. Resultados obtidos através de simulações mostram que uma dessas políticas tem um bom desempenho comparando-se a outras já existentes.

Abstract

This paper studies the concept of Processor Working Set in multiprogramed parallel systems and discusses the implementation of two scheduling policies based on it. Results obtained via simulation show that one of these policies has a good performance, when compared with others of the same class.

*Mestrando em Ciência da Computação. Áreas de Interesse: Arquitetura de Computadores, Computação Paralela e Sistemas Operacionais. E-mail: chaimo@dcc.ufmg.br

†*Ph.D. in Computer Science, UCLA*, Professor do Departamento de Ciência da Computação da UFMG. Áreas de Interesse: Arquitetura de Computadores, Supercomputação, Computação Paralela e Análise de Desempenho. E-mail: arabe@dcc.ufmg.br

1 Introdução

Ambientes paralelos multiprogramados estão se tornando cada vez mais freqüentes em computação. Nesses ambientes, diversos programas executam ao mesmo tempo nos diversos processadores existentes. Normalmente esses programas são compostos por tarefas que podem ser executadas em paralelo, melhorando muito a sua performance. Para que isso seja feito, é necessário realizar o escalonamento dos processadores entre as diversas tarefas, de forma que elas possam compartilhar os processadores de forma eficiente.

Existem basicamente dois tipos de políticas de escalonamento: estáticas e dinâmicas. Nas políticas estáticas o número de processadores alocados a um programa é fixo durante toda a sua execução. Já nas políticas dinâmicas, os processadores podem ser alocados e desalocados aos programas durante a sua execução de acordo com a necessidade. As políticas estáticas têm a vantagem de possuir um *overhead* de escalonamento menor, pois o número de realocações feitas nos processadores é menor. Em compensação, elas não acompanham a variação no paralelismo de cada programa, tornando menos eficiente a sua execução.

Existem diversos estudos sobre políticas de escalonamento em sistemas multiprocessadores com multiprogramação. Em [11] são comparadas duas políticas estáticas (*RTC - Run to Completion* e *Round-Robin*) com uma dinâmica (*Dynamic*). Nesse trabalho também é introduzida uma nova técnica chamada escalonamento em dois níveis (*two-level scheduling*). Os resultados mostram uma melhor performance da política dinâmica sobre as estáticas. Em [1] são comparadas algumas políticas dinâmicas de escalonamento mais freqüentemente usadas: *SNTF - Shortest Number of Tasks First*, *SCDF - Shortest Cumulative Demand First* (com versões preemptivas e não preemptivas), *LPF - Largest Parallelism First*, *SPF - Shortest Parallelism First* e *FCFS - First Come First Served*. A política baseada na demanda de trabalho dos programas (*SCDF*) mostrou um desempenho melhor. Em [8] são comparadas também as políticas *SNTF* e *SCDF* com algumas outras baseadas em *Round Robin*. Em [2, 3] são estudadas algumas políticas de escalonamento em ambientes com multiprocessadores heterogêneos.

Este trabalho sugere uma nova política baseada no conceito de *Working Set* de processadores. Esse conceito é relativamente novo e está relacionado com o número de processadores que alocados a um programa deixa ótima a sua execução. Foram implementadas duas políticas baseadas nesse conceito e, através de simulações, elas foram comparadas com outras já existentes.

O restante deste trabalho está organizado da seguinte maneira: na próxima seção é apresentado o conceito de *Working Set* de memória e de processadores. Na seção 3 são descritas as políticas de escalonamento e a estrutura dos programas paralelos utilizados nas simulações. A seção 4 descreve o ambiente que foi simulado, o simulador e as medidas de performance. Os resultados são apresentados na seção 5 e as conclusões na seção 6.

2 O Conceito de Working Set

2.1 Working Set de Memória

O conceito de *Working Set* (*WS*) de Memória para ambientes multiprogramados já é conhecido desde o fim da década de 60 [4, 5]. O *Working Set* se refere ao conjunto de páginas da memória virtual mais recentemente utilizadas por um programa. A idéia é se examinar as últimas θ referências que um programa faz à memória. É como se existisse uma *janela* móvel de tamanho θ que fosse guardando as últimas referências à memória. O conjunto de páginas presentes nessa *janela* é o *Working Set* do programa. A figura 1 mostra uma seqüência de referências à memória e uma *janela* de tamanho cinco. No tempo $t1$ o *WS* é composto pelas páginas 1, 2, 3, 4, e no tempo $t2$ pelas páginas 3 e 4. Devido ao princípio da localidade (espacial e temporal) em programas seqüenciais, escolhendo-se um tamanho adeqüado para essa *janela*, o *WS* conterá as páginas que estão sendo mais utilizadas pelo programa em um determinado momento.

O *Working Set* é usado em uma estratégia de alocação de páginas cujo objetivo principal é tentar otimizar a utilização do processador. Além disso, essa estratégia diminui o *thrashing*,

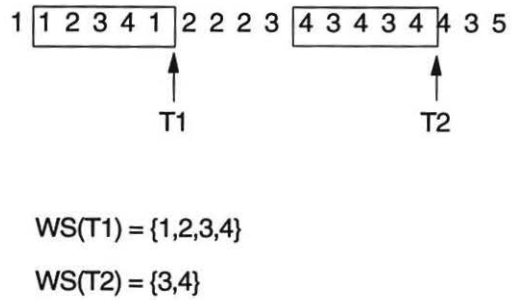

Figura 1: Working Set de Memória

procurando manter elevado o grau de multiprogramação [10]. Ela tem o seguinte princípio: um programa só pode estar em execução se todas as páginas de seu *WS* estiverem na memória. Caso o número de páginas do *WS* de um programa aumente de forma que a soma dos tamanhos dos *WS* de todos os programas em execução ultrapasse o número de molduras disponíveis, esse programa será interrompido e suas páginas realocadas para algum outro.

2.2 Working Set de Processadores

O *Working Set* de Processadores (*PWS*) segue o mesmo princípio mas, ao invés de lidar com alocação de páginas da memória para diversos programas, trata da alocação de processadores para os programas em um ambiente multiprocessador.

O conceito de *PWS* estático foi introduzido em [7], aonde é definida uma função de custo que envolve o número de processadores e a *speedup* atingida para esses processadores:

$$C(p) = \frac{p}{S(p)}, \quad S(p) = \frac{T(1)}{T(n)}$$

Essa função de custo é interessante pois se a *speedup* for linear ($S(p) = p$) então o custo para se utilizar mais processadores é pequeno (igual a 1), enquanto se a *speedup* for pequena, esse custo será alto. Por exemplo, para uma *speedup* igual a 1, o custo vai ser igual ao número de processadores já utilizados. Com base nessa função de custo é definida a eficácia do sistema:

$$\eta(p) = \frac{S(p)}{C(p)} = \frac{S^2(p)}{p}$$

O *PWS* estático de um programa é então definido como o menor número de processadores que maximiza a eficácia. Se o número de processadores alocados a um programa for menor que o seu *PWS*, a alocação de mais processadores resultará em um ganho na *speedup* em relação ao custo associado. Por outro lado, se o número de processadores alocados a um programa for maior que o seu *PWS*, o ganho com a alocação de mais processadores será menor que o custo associado. A definição de *knee* em [6] é muito similar a essa. O *Knee* é o número de processadores onde o benefício por unidade de custo é maximizado. Portanto, o *PWS* fornece o número de processadores que devem ser alocados a um programa de modo que ele tenha uma ótima relação de custo-benefício em sua execução.

O *PWS* estático é baseado no paralelismo médio de um programa, mas não leva em consideração as variações do paralelismo a cada momento. O paralelismo de um programa pode ser visualizado pelo seu grafo de precedência, que mostra, a cada momento, as diversas tarefas executadas pelo programa bem como a relação de precedência entre elas. O conceito de *PWS* dinâmico está relacionado com o grafo de precedência de um programa. O *PWS* é o número de tarefas que podem ser executadas simultaneamente pelo programa em um determinado momento como é mostrado na figura 2. Portanto, o *PWS* dinâmico nos fornece o número de processadores que alocados a um programa em um determinado momento torna ótima a sua execução.

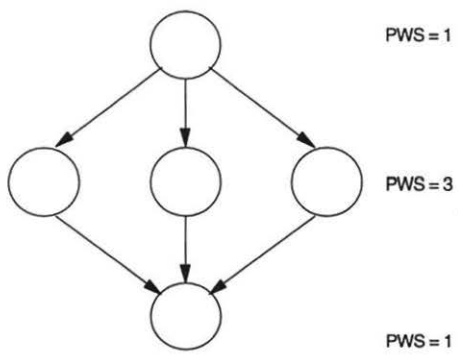

Figura 2: Grafo de Precedência e o *PWS*

Com base nesse conceito pode-se definir uma estratégia de alocação que segue o mesmo princípio da estratégia de alocação de páginas para o *Working Set* de memória: um programa só pode estar em execução se o número de processadores alocados a ele for igual ao seu *PWS* naquele momento. Se num determinado momento, devido ao aumento do paralelismo de algum dos programas, a soma do *PWS* de todos os programas em execução ultrapassar o número de processadores disponíveis, um dos programas deve ser interrompido e os processadores alocados a ele devem ser realocados.

3 Descrição do Problema

3.1 Políticas de Escalonamento

Com base na estratégia de alocação descrita para o *PWS* dinâmico, foram implementadas duas políticas de escalonamento, uma denominada *pura* e outra *modificada*. Ambas seguem as seguintes regras:

1. Quando um novo programa chega para execução, se o número de processadores livres for maior ou igual ao seu *PWS* ele é colocado em execução com um número de processadores igual ao seu *PWS*, senão ele é colocado em uma fila de espera. Essa fila é organizada de modo que os programas com *PWS* menor tenham prioridade.
2. Quando um programa termina sua execução, ou é interrompido, faz-se uma pesquisa na fila para ver se existem programas esperando cujo *PWS* seja menor ou igual ao número de processadores liberados. Se existir, os processadores são alocados ao programa.

3. Quando um programa aumenta seu paralelismo e conseqüentemente seu *PWS*, necessitando de mais p processadores podem ocorrem as seguintes situações:
   (a) Se existirem p ou mais processadores livres, p processadores são alocados para o programa.
   (b) Se não existirem p processadores livres, ou seja, a soma do *PWS* de todos os programas em execução ultrapassa o número de processadores disponíveis, o programa é interrompido e suas tarefas colocadas na fila.

A diferença entre as políticas pura e modificada está no procedimento realizado quando existem processadores livres, mas o *PWS* de todos os programas da fila é maior que o número de processadores disponíveis:

- A política pura (*PWSP*) deixa os processadores ociosos, ou seja, segue fielmente o princípio que diz que o programa só pode estar em execução se o número de processadores alocados a ele for igual ao seu *PWS*.
- A política modificada (*PWSM*) aloca os processadores livres ao primeiro programa da fila, diminuindo o seu *PWS*. Esse programa, ao qual foram alocados os processadores, é considerado *particionado* e passa a ter prioridade menor que um programa com o mesmo *PWS* mas que não foi particionado.

Além dessas, foram utilizadas para comparação mais duas políticas de escalonamento:

- *FCFS - First Come First Served*
  Essa política não leva em consideração as características do programa. Um processador que estiver livre é alocado à primeira tarefa que o requisita.
- *SNTF - Shortest Number of Tasks First*
  Essa política privilegia os programas que tem um número menor de tarefas. Um processador livre é alocado à tarefa que pertence ao programa com o menor número de tarefas que ainda não foram executadas.

3.2 Estrutura dos Programas Paralelos

Um programa paralelo pode ser representado pelo seu grafo de precedência, que nos mostra as tarefas a serem executadas pelo programa bem como a relação de precedência entre elas. Uma tarefa pode ser identificada como um fragmento do programa que deve ser executado seqüencialmente. É importante salientar que apenas uma tarefa pode estar em execução em um processador num determinado momento. Nesse estudo, os programas paralelos têm a estrutura do tipo *Multiple Fork-and-Join (MFJ)*. Essa estrutura representa o comportamento de diversos programas científicos tais como a multiplicação de matrizes.

A principal característica dos programas *MFJ* é que eles sofrem mudanças repentinas em seu paralelismo. Os programas *MFJ* fazem uma série de iterações idênticas, compostas por um período serial e um paralelo, como pode ser visto na figura 3. Quando entra em seu período paralelo o programa realiza uma operação de *fork*, gerando um número determinado de filhos. Quando todos os filhos terminam sua execução uma operação de *join* é realizada retornando o programa à fase serial. O *MFJ* pode ser descrito por dois parâmetros básicos: o número de iterações ou número de *forks* (*NF*) e o grau de paralelismo máximo ou número de filhos por fork (*GP*).

4 Modelagem do Sistema

4.1 Ambiente Multiprocessador

O ambiente multiprocessador simulado consiste de n processadores, cada um capaz de executar uma tarefa de cada vez. A memória é compartilhada entre os diversos processadores, e o tempo para acessá-la é igual para todos. Existe uma fila única para todos os processadores, como é mostrado na figura 4, e todas as tarefas que chegam para ser executadas

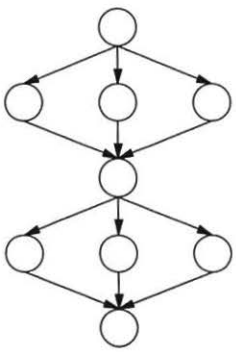

Figura 3: Grafo *Multiple Fork-and-Join* (*NF=2, GP=3*)

são colocadas nessa fila de acordo com cada política de escalonamento. Para os experimentos realizados, o número de processadores escolhido foi 20, o que representa um sistema multiprocessador de média escala.

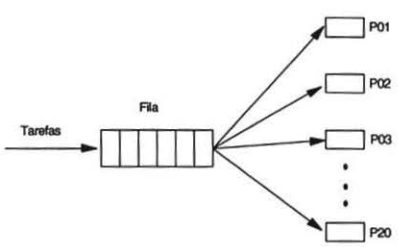

Figura 4: Ambiente Multiprocessador

O tempo para executar a política de escalonamento (*scheduling overhead*) é desprezado, pois uma política de escalonamento eficiente deve ser rápida. O tempo para um processador terminar a execução de uma tarefa e começar a execução de outra (*processor alocation overhead*) também é desprezado. Isso porque, para esse modelo específico de simulação, apenas com programas do tipo *MFJ*, o *overhead* é o mesmo para todas as políticas pois nenhuma tarefa é interrompida tendo de ser realocada posteriormente.

4.2 Modelagem da Carga

A carga do sistema consiste de uma seqüência de programas paralelos que chegam ao sistema para serem processados. A chegada dos programas é descrita por uma distribuição exponencial, com um intervalo de chegada médio dado por λ. Como foi dito, os programas são do tipo *MFJ* descritos por seus dois parâmetros (*NF* e *GP*). Cada uma das tarefas tem sua demanda de processamento dada por D. Essa demanda é o tempo que cada tarefa gastará em um processador para ser executada. Esses valores também são dados por uma distribuição exponencial. Um outro parâmetro é o tempo de simulação *TS*, que indica para quantos

programas será realizada a simulação. Resumindo, os parâmetros das simulações são dados na tabela 1. Todos, exceto *TS* são dados de acordo com uma distribuição exponencial.

λ	Intervalo de chegada dos programas
NF	Número de forks do programa
GP	Grau de paralelismo máximo do programa
D	Demanda de cada tarefa
TS	Tempo de simulação

Tabela 1: Parâmetros de Carga das Simulações

4.3 Simulador

O simulador utilizado foi o SMPL. O SMPL é um ambiente de simulação baseado em eventos discretos. Ele é composto de diversas funções escritas na linguagem C com as quais é modelada a simulação desejada [9]. Os programas das simulações foram escritos na linguagem C, ligados com a biblioteca do SMPL, e executados em uma *Sun Sparc Station 2*.

4.4 Medidas de Performance

Basicamente, foram usadas duas medidas de performance para se comparar as políticas de escalonamento:

- **Tempo de Resposta: é o tempo passado entre o instante que o programa chega na fila até o término da execução de sua última tarefa.**
- **Utilização dos Processadores: é a fração do tempo em que os processadores ficam** ocupados durante a simulação.

Além dessas duas medidas, também foi medida a fração de jobs que são particionados durante a execução da política *PWS Modificada*. Essa fração representa a principal diferença entre a *PWS Pura* e a *PWS Modificada*.

5 Resultados

5.1 Variando-se a Taxa de Chegada

Os primeiros testes foram realizados variando-se a taxa de chegada dos programas no sistema (λ). O número médio de forks (*NF*) e o número médio de filhos (*GP*) foram mantidos constantes e iguais a 4. A demanda média de cada tarefa *D* foi igual a 2, e a simulação foi realizada para 5000 programas, ou seja, $TS = 5000$. Para cada um dos pontos foram executadas dez simulações, obtendo-se um intervalo de confiança de 90%.O gráfico da figura 5 mostra o tempo de resposta, o da figura 6 mostra a utilização do sistema, e no gráfico da figura 7 tem-se a fração de programas particionados pela política *PWSM*.

No gráfico da figura 5, a política *PWSP* é a que tem o pior desempenho, principalmente quando a carga do sistema é média. Quando a carga é muito alta (um programa a cada duas unidades de tempo) a política *FCFS* é a pior de todas, mas logo depois ela passa acompanhar o desempenho das outras duas políticas. A *SNTF* e a *PWSM* tem um desempenho muito parecido, com uma ligeira vantagem para a segunda quando a carga do sistema é média (entre 3 e 6 ut/jobs). O gráfico da figura 6 mostra porque a política *PWSP* tem o pior desempenho. Enquanto todas as outras políticas utilizam os processadores o máximo possível, a *PWSP* deixa processadores ociosos mesmo quando há tarefas esperando na fila, pois o número de processadores disponíveis é menor que o *Working Set* dos programas que estão na fila. A *PWSM* não deixa processadores ociosos, pois particiona um grande número de programas principalmente quando a carga está alta (figura 7), obtendo um desempenho muito melhor.

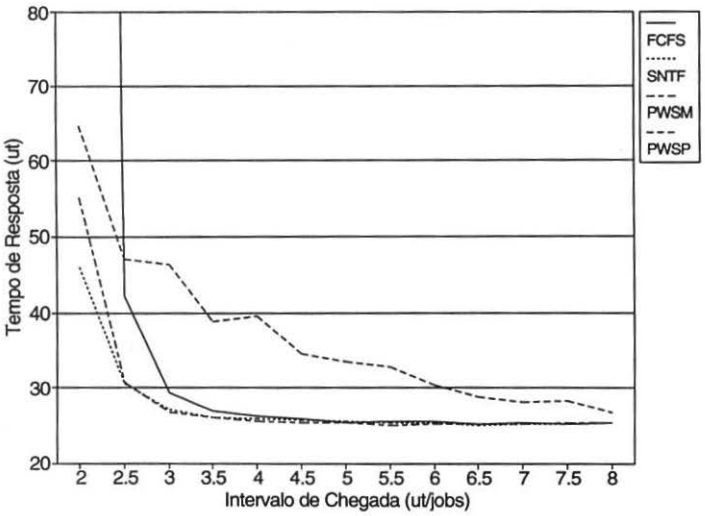

Figura 5: Tempo de Resposta × Intervalo de Chegada

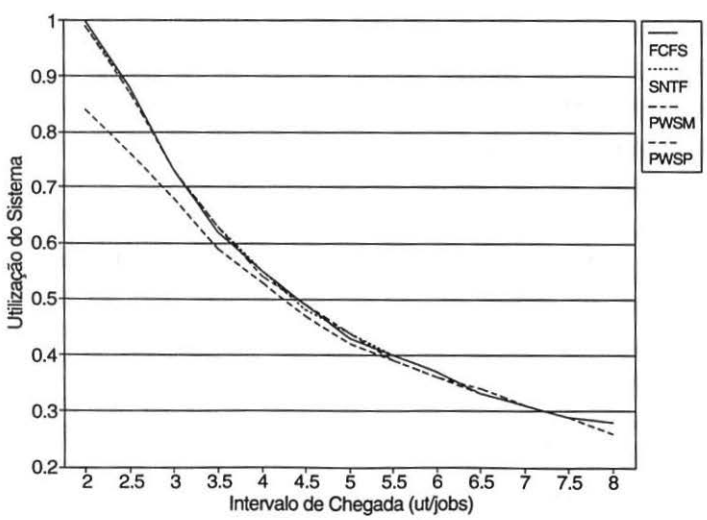

Figura 6: Utilização do Sistema × Intervalo de Chegada

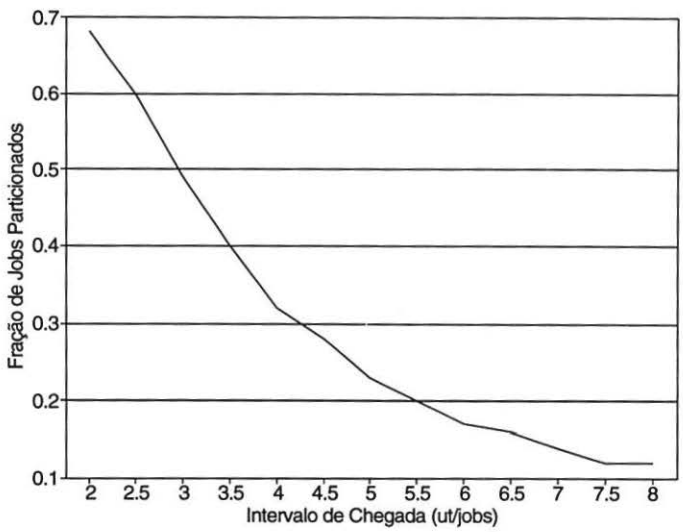

Figura 7: Fração de Jobs Particionados × Intervalo de Chegada

5.2 Variando-se o Grau de Paralelismo

Outros testes foram realizados variando-se o grau de paralelismo médio dos programas. Os parâmetros de carga foram mantidos os mesmos, com a diferença que o taxa de chegada (λ) foi mantida constante e igual a 4 ut/jobs e o que variou foi o grau de paralelismo ou número médio de filhos por *fork* (*GP*). O gráfico da figura 8 mostra o tempo de resposta médio, o da figura 9 mostra a utilização do sistema, e no gráfico da figura 10 tem-se a fração de programas particionados pela política *PWSM*.

No gráfico da figura 8, quando o grau de paralelismo é baixo, as políticas têm um desempenho similar. À medida que *GP* vai aumentando, a política *PWSP* vai tendo um desempenho cada vez pior. Isso ocorre porque o *PWS* médio dos programas vai se tornando cada vez maior, e fica díficil arranjar um número de processadores disponíveis para suprir o *PWS* dos programas que estão na fila. Portanto, à medida que o grau de paralelismo aumenta a utilização do sistema pela política *PWSP* torna-se menor em relação às outras políticas (figura 9), e conseqüentemente o tempo de resposta médio torna-se pior. A política *PWSM*, como particiona os programas principalmente quando o paralelismo é alto (figura 10), não sofre esse problema e tem um desempenho equivalente ao da *SNTF*.

Outro ponto que deve ser levado em consideração é que as políticas baseadas no *PWS* priorizam os programas com um grau de paralelismo menor. Isso acontece devido à maneira como a fila é organizada, dando prioridade a quem tem um *PWS* menor. Isso pode causar o fenômeno conhecido como *starvation* no qual os programas com paralelismo maior ficariam esperando na fila por tempo indeterminado. Esse fenômeno é mais acentuado na política pura, que deixa processadores ociosos quando o paralelismo dos programas é maior que o número de processadores livres. Esse fenômeno também ocorre com a *SNTF*, que prioriza os programas com menor número de tarefas para execução.

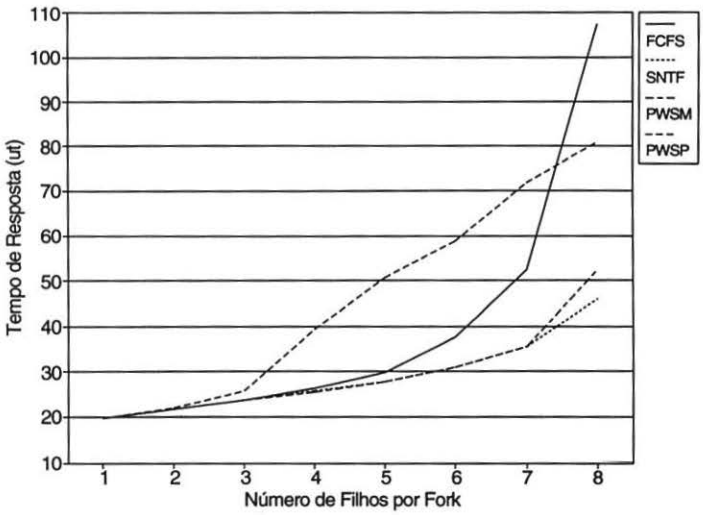

Figura 8: Tempo de Resposta × Grau de Paralelismo

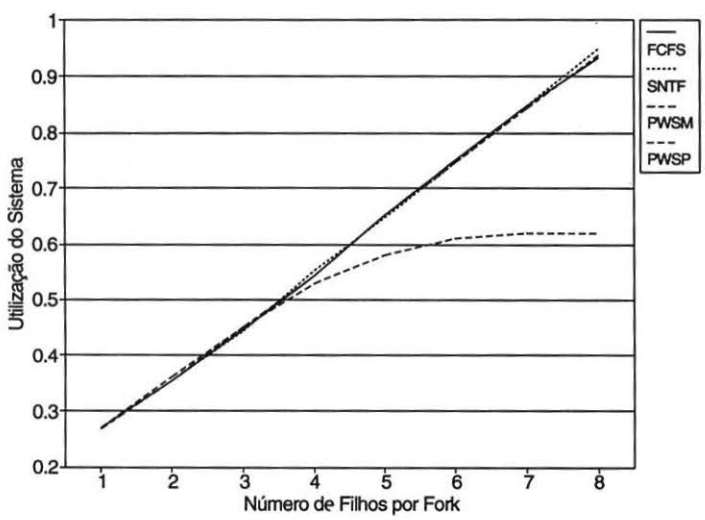

Figura 9: Utilização do Sistema × Grau de Paralelismo

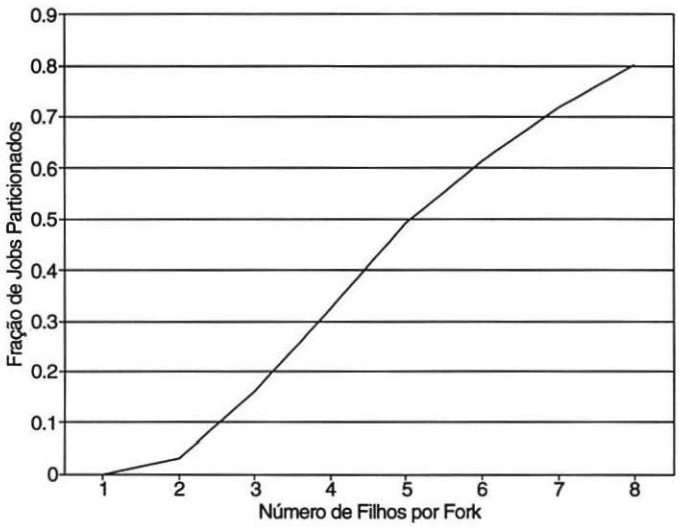

Figura 10: Fração de Jobs Particionados × Grau de Paralelismo

5.3 Variando-se o Tempo de Simulação

Um último teste foi realizado com o objetivo de mostrar como o desempenho da política *PWSP* é extremamente sensível ao número de programas utilizados na simulação. No gráfico da figura 11, foi variado o número de programas para o qual foi feita a simulação, mantendo-se os outros parâmetros constantes. Como o tempo de resposta é calculado tirando-se a média do tempo de resposta de todos os programas, era de se esperar que ele permanecesse aproximadamente constante para qualquer que fosse o tempo de simulação. Isso ocorre para todas as políticas, menos para a *PWSP*. Como a *PWSP* não utiliza todo o poder de processamento disponível, o processamento torna-se um *gargalo* e a fila com as tarefas esperando fica cada vez maior. Quanto maior o tempo de simulação, maior será o tamanho dessa fila, e o tempo de resposta médio torna-se cada vez pior. Isso na verdade ocorreria para todas as políticas, pois dependendo da carga, sempre existirá um ponto em que o sistema fica sobrecarregado. Esse gráfico mostra que para a política *PWSP* isso ocorre rapidamente, muito antes das outras políticas.

6 Conclusões

Esse trabalho discutiu o conceito de *Working Set* de Processadores para sistemas paralelos multiprogramados. Com base nesse conceito foram implementadas duas políticas de escalonamento, *PWSP* e *PWSM*, que foram comparadas com outras através de simulações considerando um ambiente paralelo com 20 processadores e programas do tipo *Multiple Fork-and-Join*. Através dos resultados das simulações chegou-se às seguintes conclusões:

- A política *PWSP* não apresentou bons resultados para praticamente todas as situações simuladas. Isso ocorre porque ela não se aproveita de todo o poder de processamento existente, deixando processadores ociosos e conseqüentemente piorando muito a sua performance.

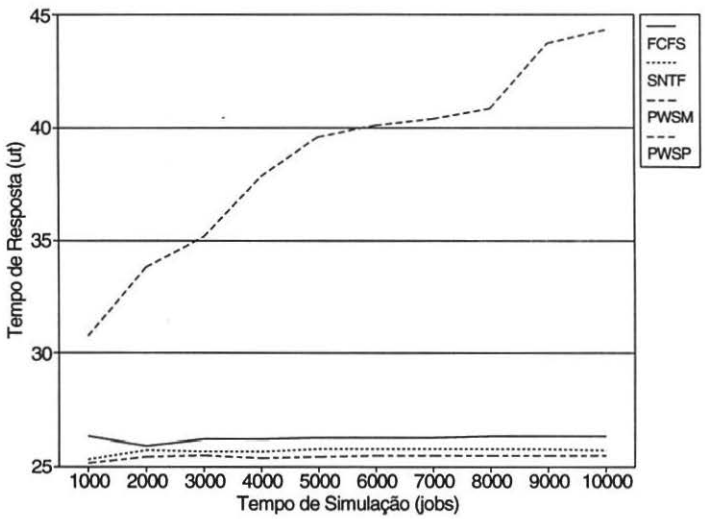

Figura 11: Tempo de Resposta × Tempo de Simulação

- Tanto as políticas baseadas no *PWS* quanto a *SNTF* podem sofrer o problema de *starvation*, ou seja, os programas com alto grau de paralelismo ou com grande número de tarefas ficarem esperando por um tempo indeterminado na fila porque têm menor prioridade. Uma solução para isso seria ir aumentando a prioridade dos programas de acordo com o tempo que eles estão esperando.

- As políticas *PWSM* e *SNTF* foram as que apresentaram melhor desempenho em praticamente todas as situações. Numericamente o resultado das duas foi praticamente igual, mas a política *STNF* leva desvantagem em um ponto básico: a necessidade que ela tem de um conhecimento completo do grafo de execução do programa, para que ela possa selecionar qual tem o menor número de tarefas. A *PWSM* calcula o *PWS* a cada momento, sem a necessidade de um conhecimento anterior do grafo de precedência. Isso torna a *PWSM* uma política mais viável para ser implementada.

Esse trabalho deixa em aberto alguns tópicos para pesquisas futuras. Um deles é a realização de simulações para outros tipos de programas paralelos diferentes do *MFJ*. Grafos do tipo *MFJ* têm características próprias que podem favorecer um tipo específico de política. Portanto, a utilização de outros tipos de grafos como o *MVA* [8] ou grafos aleatórios [3] seria interessante. Outra sugestão seria a implementação de novas políticas baseadas no *PWS*. Por exemplo uma política que deixa os processadores ociosos esperando durante um intervalo δ de tempo. Ela seria algo entre a *PWSP* que deixa os processadores ociosos até que um programa com *PWS* menor apareça ($\delta = \infty$) e a *PWSM* que nunca deixa os processadores ociosos ($\delta = 0$). Essa política tentaria melhorar o tempo de resposta particionando um menor número de programas. Por fim, deve ser feita a implementação real dessa política em algum ambiente paralelo, para observar o seu desempenho em situações que não sejam de simulação.

Referências

[1] Almeida, V. A. F. and Vasconcelos I. M. M. "A Simulation Study of Processor Scheduling Policies in Multiprogrammed Parallel Systems" *Proceedings of the 1991 Summer Computer Simulation Conference*, Baltimore, Maryland, July 22–24 1991, pp. 276–281.

[2] Almeida, V. A. F., Vasconcelos I. M. M. and Árabe, J. N. C. "The Effect of Heterogeneity on the Performance of Multiprogrammed Parallel Systems" *International Paralell Processing Symposium, Workshop on Heterogeneous Processing*, Beverly Hills, CA, March 23–26, 1992, pp.23–31.

[3] Almeida, V. A. F., Vasconcelos I. M. 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 Supercomputing 92*, Minneapolis, November 16–20 1992, pp. 683-691

[4] Denning, P. J. "Virtual Memory" *Computing Surveys*, Vol. 2, N. 3, September 1970, pp. 153–159.

[5] Denning, P. J. "Working Sets Past and Present" *IEEE Transactions on Software Engeneering*, Vol. SE-6, N. 1, January 1980, pp. 64–84.

[6] Eager, D. L., Zahorjan, J. and Lazowska, E. D. "Speedup Versus Efficiency in Parallel Systems" *IEEE Transactions on Computers*, Vol. 38, N. 3, March 1989, pp. 408–423.

[7] Ghosal, D., Serazzi, G. and Tripathi, S. K. "The Processor Working Set and Its Use in Scheduling Multiprocessor Systems" *IEEE Transactions on Software Engeneering*, Vol. 17, N. 5, May 1991, pp. 408–423.

[8] Leutenegger, S. T. and Vernon, M. K. "The Performance of Multiprogrammed Multiprocessor Scheduling Policies" Computer Sciences Technical Report N. 913, Computer Sciences Department, University of Wisconsin, Madison, February 1990.

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

[10] Peterson, J. L. and Silberschatz, A. *"Operating Systems Concepts"* Addison-Wesley Publishing Company, Reading, 1985.

[11] Zahorjan J. and McCann, C. "Processor Scheduling in Shared Memory Multiprocessors" Technical Report 89-09-17, Department of Computer Science, University of Washington September 1989.