NBSP: Uma política de escalonamento *network-bound* para aplicações paralelas distribuídas

Renato Porfírio Ishii

Orientador: *Prof. Dr. Marcos José Santana*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC–USP — como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Novembro de 2004

Candidato: Renato Porfírio Ishii

A Comissão Julgadora:

Prof. Dr. Marcos José Santana _____

Prof. Dr. Rodrigo Fernandes de Mello _____

Prof. Dr. Gonzalo Travieso _____

A minha família

Agradecimentos

Em primeiro lugar agradeço a Deus por me conceder o dom mais importante: a Vida.

Agradeço ao professor Marcos José Santana, pela orientação, confiança e paciência durante as diversas fases do trabalho.

À professora Regina pelas críticas construtivas, por sua disponibilidade e opiniões sempre seguras.

Ao professor Rodrigo e ao Luciano pelo apoio direto no trabalho.

Aos demais professores e funcionários do Instituto, pelo profissionalismo e disponibilidade. Representantes dessa instituição que aprendi a respeitar e compreender o significado: USP, educação para o Brasil.

A minha mãe Iracema, minha irmã Regina, meus sobrinhos Luiz Fernando e João Paulo, meu cunhado Beto e todos os outros familiares que sempre demonstraram total apoio e compreensão na minha ausência. Sem esquecer daquele que deixou saudades: meu pai Fumio.

A todos os amigos e amigas. Para não cometer o equívoco de esquecer algum, não listarei todos os nomes. Mas fiquem certos que sua amizade foi de suma importância para a conclusão deste trabalho e, como se não bastasse, terão que me aturar daqui para frente!

Aos amigos e colegas do grupo LaSDPC. Aos amigos de república: Douglas, Vinicius, Adriano, Rodrigo, Juliano e Michel. Amigos de outros grupos, galera do volei e do futebol.

À CAPES, pelo apoio financeiro.

Resumo

Este trabalho apresenta uma nova política de escalonamento para aplicações paralelas *Network-Bound* baseada no impacto do processamento causado pela comunicação entre processos. O modelo utilizado quantifica o volume de tráfego imposto sobre a rede de comunicação por meio dos parâmetros latência e sobrecarga. Tais parâmetros representam a carga que cada processo impõe sobre a rede e o atraso sobre a CPU devido às operações na rede. Esse atraso é representado na política por meio da métrica *slowdown*. Equações matemáticas são definidas para a quantificação dos custos envolvidos no processamento e na troca de mensagens, do mesmo modo, são propostas equações para determinar a largura máxima de banda (*bandwidth*) utilizadas nas tomadas de decisões de escalonamento. Outra característica importante da política é a definição de uma constante k, que delimita a utilização máxima permitida da rede de comunicação. O valor de k define a adoção de duas possíveis técnicas de escalonamento: escalonamento em grupo, ou por intermédio da rede de comunicação. As técnicas propostas são incorporadas à política de escalonamento DPWP (originalmente *CPU-Bound*) gerando uma extensão *Network-Bound*. Resultados experimentais e de simulação confirmam o aumento de desempenho de aplicações paralelas sob supervisão da política DPWP estendida, denominada NBSP, quando comparadas às execuções supervisionadas pela DPWP original.

Abstract

This work presents a new scheduling policy for *Network-Bound* parallel applications based on impacts of the processing operations resulting from the communication among processes. The model adopted quantifies the traffic volume imposed on the communication network by means of the latency and the overhead parameters. Such parameters represent the load that each process imposes over the network and the delay on the CPU, as a consequence of the network operations. The delay is represented on the model by mean of metric measurements *slowdown*. The mathematics equations that quantify the costs involved in the processing operation and message exchange are defined. In the same way, equations to determine the maximum network bandwidth are used on the decision-making scheduling. Another important feature of the policy is the definition of a constant k that delimitates the communication network maximum allowed usage. The k value defines the adoption of two possible scheduling techniques: group scheduling or through communication network. The proposed techniques are incorporated to the DPWP scheduling policy (oriniginally *CPU-Bound*), generating an extension *Network-Bound*. Experimental and simulation results confirm the performance enhancement of parallel applications under supervision of the extended DPWP policy, denominated NBSP, when compared to the executions supervised by the original DPWP.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO

1

Introdução

O problema do escalonamento eficiente de processos é um tema relevante em sistemas computacionais distribuídos. Para resolver esse problema várias políticas de escalonamento foram propostas (OH; HA, 1996; PARK; CHOE, 2002; RANAWEERA; AGRAWAL, 2000; J.Y.COLIN; P.CHRÉTIENNE, 1991; DARBHA; AGRAWAL, 1998; TOPCUOGLU et al., 1999; CHOE; PARK, 2002; RADULESCU et al., 1999). Essas políticas distribuem processos, que compõem aplicações paralelas distribuídas, sobre os elementos de processamento (EPs) disponíveis, visando objetivos tais como: balanceamento de carga, diminuição do tempo de resposta das aplicações e melhor utilização de recursos.

Segundo Feitelson e Rudolph (1998), as políticas podem considerar vários parâmetros que afetam o desempenho, incluindo a carga de ocupação da CPU, a quantidade de memória disponível, entrada/saída e comunicação, a fim de minimizar o tempo de execução dos processos. Pode-se estender esses parâmetros considerando-se diferentes classes de aplicações. Essas aplicações podem ser subdivididas de acordo com a utilização dos recursos, em duas classes: *CPU-Bound* e *I/O-Bound*.

Aplicações da classe *CPU-Bound* são caracterizadas pela alta demanda por processamento (utilização de CPU), enquanto as da classe *I/O-Bound* são caracterizadas pelo intenso acesso aos dispositivos de entrada e saída (SILVA; SCHERSON, 2000; SENGER et al., 2004; SENGER et al., 2003).

A classe de aplicações *I/O-Bound* pode ser subdividida, de acordo com os recursos de entrada e saída mais utilizados pela aplicação, em: *Disk-Bound* e *Network-Bound*. Aplicações da classe *Disk-Bound* realizam muitos acessos à memória secundária. Aplicações da classe *Network-Bound* realizam muitas operações de envio e recebimento de mensagens através da rede de comunicação.

A utilização e o desenvolvimento de aplicações da classe *Network-Bound* têm-se destacado, principalmente, com a evolução contínua das redes de computadores (BERMAN; WOLSKI, 1996; GRAMA et al., 2003) e com a consolidação dos sistemas computacionais distribuídos: *clusters*, NOW's (*Network of Workstations*) (TINETTI; BARBIERI, 2003; ANGLANO, 2000) e Grids (LIU et al., 2002). Exemplos de aplicações que utilizam esses tipos de sistemas são: sistemas corporativos, dinâmica de fluidos, aplicações para previsão do tempo e processamento de imagens (RANAWEERA; AGRAWAL, 2000).

Sistemas distribuídos são caracterizados por um conjunto de recursos compartilhados através de uma rede de comunicação e visam características tais como: transparência no acesso aos recursos, tolerância a falhas, possibilidade de crescimento em escala e flexibilidade. Essas características, aliadas à possibilidade de baixo custo na aquisição e manutenção, tornam atrativa a utilização de sistemas computacionais distribuídos em vários domínios (TANENBAUM; STEEN, 2002).

Um dos domínios que se beneficiam das vantagens dos sistemas distribuídos é a computação paralela, que consiste na exploração de eventos concorrentes no processo computacional, por intermédio de EPs que cooperam e comunicam entre si (ALMASI; GOTTLIEB, 1994). Antes restrita aos domínios compostos de computadores específicos e com processamento paralelo, a computação paralela, em busca de melhor relação custo/benefício, tem sido constantemente empregada em sistemas computacionais distribuídos. Essa abordagem permite associar as vantagens dos sistemas distribuídos com os objetivos da área de computação paralela. Dessa maneira, o sistema distribuído é visto pelas aplicações paralelas, como uma máquina paralela virtual, cujos EPs estão espalhados pela rede de comunicação.

Quando existem diversos EPs disponíveis e vários usuários interessados em executar aplicações nesses EPs, é necessário que haja um software que controle o acesso a esses elementos. Portanto, deve haver um componente do sistema computacional responsável pelo escalonamento dos processos.

Decisões comuns a serem tomadas pelo software de escalonamento são quais aplicações terão acesso aos recursos computacionais, a quantidade e a localização dos recursos que serão destinados às aplicações. Essas decisões são afetadas por diferentes fatores, tais como a carga de trabalho imposta ao sistema, a heterogeneidade dos recursos e as diferentes necessidades dos usuários. Além disso, o escalonador deve tratar de maneira especializada aplicações paralelas com diferentes comportamentos em relação à utilização dos recursos computacionais (FEITELSON; RUDOLPH, 1998).

As políticas de escalonamento podem adaptar-se às diferentes classes de aplicações a serem escalonadas. Entretanto, poucos trabalhos têm visado o desenvolvimento de políticas voltadas para aplicações da classe *I/O-Bound* (SILVA; SCHERSON, 2000; MACHE et al., 1999; CHASE et al., 1999). Esses trabalhos avaliam tanto aplicações *Disk-Bound* quanto *Network-Bound*

que manipulam grandes volumes de dados (*data intensive*). Nessas avaliações fica evidente que tais aplicações requerem muita largura de banda (*bandwidth*) e baixa latência, tanto para o acesso à memória secundária, quanto à rede de comunicação. A rede de comunicação apresenta-se como o principal gargalo de desempenho na execução dessas aplicações, dadas suas limitações de largura de banda e latência.

Aplicações da classe *Network-Bound* podem disputar a utilização dos recursos da rede de comunicação. Essa disputa, conhecida como *network contention*, afeta o tempo de transferência das informações (latência) e, conseqüentemente, prolonga o tempo de execução dos processos. Esse fato motiva o desenvolvimento de pesquisas sobre avaliações e análises do estado da carga de ocupação dos recursos envolvidos no subsistema de comunicação.

Essa motivação conduziu este trabalho a um estudo criterioso do escalonamento de aplicações paralelas, que realizam constantes transferências de informações em rede, gerando uma extensão de uma política de escalonamento, denominada DPWP (ARAÚJO et al., 1999) previamente desenvolvida. Esse estudo considera parâmetros de sobrecarga de comunicação, latência e volume transmitido de mensagens, que em conjunto com os parâmetros de ocupação da CPU, permitem definir índices de carga. Esses índices quantificam o estado dos recursos do sistema e, dessa maneira, contribuem nas tomadas de decisões de escalonamento. Decisões fundamentadas por meio desses parâmetros, agregam melhorias na utilização dos recursos do sistema e melhoram o tempo de execução das aplicações.

Utilizando-se esses índices, baseados nas ocupações de CPU e da rede de comunicação, são definidas duas regras de decisão de escalonamento: alocação de processos em grupo, no mesmo EP, ou distribuição de processos entre os EPs disponíveis, ampliando o uso da rede. Observou-se que essa estratégia, em determinadas situações de carga, aumenta o desempenho das aplicações de comunicação intensiva. Isso ocorre porque, alocando todos os processos num único elemento de processamento, diminui-se o atraso na sincronização dos processos que comunicam entre si. Dessa maneira, o tempo gasto em comunicação restringe-se à capacidade dos recursos (barramentos e interfaces locais de comunicação) do EP receptor, deixando a rede livre para outros escalonamentos e, conseqüentemente, minimizando o tempo de resposta das aplicações que apresentam como gargalo a rede de comunicação.

A rede de comunicação é utilizada sempre que a transmissão dos processos não prejudique o desempenho da aplicação, isto é, que a ocupação da rede não cause grandes atrasos. Isso permite, conseqüentemente, melhor exploração do paralelismo presente nas aplicações executadas simultaneamente, nos diferentes EPs do sistema.

O estudo do comportamento da carga dos processos sobre a rede de comunicação e o estudo do escalonamento de aplicações paralelas, baseado no volume de comunicação entre processos, são apresentados nesta dissertação. A partir desses estudos, são definidos um simulador e uma extensão da política DPWP. Tal extensão auxilia na avaliação de desempenho do

escalonamento de aplicações paralelas *Network-Bound*.

Esta dissertação está organizada através dos seguintes capítulos:

No capítulo 2 são apresentados os conceitos sobre computação paralela distribuída. São discutidos conceitos gerais da computação paralela, principais arquiteturas, bem como uma classificação para a área. Também são apresentados conceitos de sistemas distribuídos, softwares de suporte à computação distribuída e a convergência entre essas duas áreas, computação paralela e sistemas distribuídos.

No capítulo 3 são apresentadas as principais técnicas para avaliar o desempenho de sistemas computacionais. Tais técnicas são subdivididas em modelagem e aferição. Também busca-se demonstrar quais dessas técnicas podem ser aplicadas na avaliação do objeto de estudo desta dissertação, que compreendem um ambiente paralelo virtual (computação paralela sobre sistemas distribuídos) e o escalonamento de processos.

No capítulo 4 é apresentada uma revisão da literatura sobre escalonamento de processos em sistemas distribuídos, incluindo conceitos básicos e técnicas de escalonamento de processos. Essa revisão busca enfatizar as formas de compartilhamento da potência computacional entre as aplicações, assim como exemplos de softwares de escalonamento são apresentados.

No capítulo 5 é apresentada uma revisão bibliográfica sobre políticas de escalonamento de processos. Essa revisão compreende trabalhos sobre políticas de escalonamento, estratégias para diminuição da sobrecarga de comunicação e técnicas para modelagem do tráfego gerado sobre a rede de comunicação.

No capítulo 6 é proposta a política de escalonamento NBSP (*Network-Bound Scheduling Policy*) baseada na avaliação do impacto da comunicação entre processos. Dessa maneira, são apresentados o modelo de comunicação e a extensão de uma política de escalonamento previamente desenvolvida, denominada DPWP.

No capítulo 7 é apresentada a avaliação de desempenho decorrente dessa nova política de escalonamento de processos. Essa avaliação envolveu experimentos sobre um ambiente real, a extensão de um simulador e a extensão da política DPWP, denominada NBSP. Também são apresentados os resultados obtidos com o uso da política NBSP em comparação à DPWP.

Finalmente, no capítulo 8 são apresentadas as conclusões do trabalho acompanhada das principais contribuições e de propostas para trabalhos futuros.

CAPÍTULO

2

Computação paralela distribuída

2.1 Considerações iniciais

A computação paralela caracteriza-se, basicamente, pela utilização de elementos de processamento (EPs) que cooperam e comunicam entre si. Essa abordagem visa a resolução de problemas de maneira mais eficiente, se comparada à solução seqüencial para o mesmo problema (ALMASI; GOTTLIEB, 1994). Seu modelo arquitetural consiste de uma evolução da máquina de von Neumann para computação seqüencial, com a vantagem de superar suas limitações de hardware. Isso pode ser constatado por meio de duas características: a primeira é oferecer maior capacidade de processamento, que proporciona maior desempenho e segundo, por apresenta-se potencialmente, mais tolerante a falhas. Além disso, caracteriza-se pela capacidade de resolver eficientemente, aplicações que são naturalmente paralelas (HWANG; XU, 1998).

Um dos objetivos principais da computação paralela é a busca por maior desempenho em aplicações que necessitam de maior potência computacional. No entanto, isso leva ao surgimento de novos problemas, inexistentes na computação seqüencial. Um exemplo disso é o gerenciamento e a organização dos processadores, que em um sistema computacional paralelo e mais complexo.

Para dar suporte à computação paralela, diferentes sistemas computacionais paralelos podem ser empregados. Aplicações que demandam muitos recursos computacionais têm sido executadas em sistemas computacionais paralelos com memória distribuída. Isso permite a obtenção satisfatória de maior desempenho, possibilitando ainda crescimento em escala da potência computacional. Esses computadores podem ser máquinas com processamento maciçamente paralelo (MPP - *Massive Parallel Processing*), que são constituídos de inúmeros EPs conectados por uma rede de alta velocidade.

Outra maneira de se obter um sistema computacional paralelo é por meio da utilização de sistemas distribuídos. Tais sistemas são formados por EPs interligados através de uma rede de comunicação, um sistema operacional de rede (Linux, Solaris, etc) e um software especial (PVM, MPI, etc), que juntos atuam como uma máquina paralela virtual (ANDERSON et al., 1995).

Apesar da computação paralela ter a vantagem de proporcionar alto desempenho, fatores como o alto custo de aquisição e manutenção, e a dependência ao fabricante têm dificultado sua utilização em computadores paralelos. Entretanto, a boa relação custo/benefício e a grande flexibilidade presente em sistemas distribuídos motiva seu emprego como uma plataforma para execução de programas paralelos (ARPACI et al., 1995).

Técnicas e mecanismos antes utilizados para construção, partição e mapeamento de programas paralelos, nem sempre se adaptam aos sistemas distribuídos. Fatores característicos de sistemas distribuídos, tais como a heterogeneidade, o fraco acoplamento dos recursos computacionais, a existência de aplicações com diferentes requisitos e a própria satisfação do usuário do sistema geram dificuldades para o software paralelo (CORBALAN et al., 2001).

Com o objetivo de oferecer o embasamento geral para o desenvolvimento deste projeto de pesquisa, neste capítulo são apresentadas as principais características da computação paralela distribuída. Isso é abordado destacando-se a utilização de sistemas computacionais distribuídos como plataforma de execução de aplicações paralelas. Dessa maneira, na seção 2.2 são apresentados alguns conceitos gerais de computação paralela, na seção 2.3 são apresentadas as arquiteturas paralelas, na seção 2.4 são apresentados conceitos sobre sistemas distribuídos, na seção 2.5 é apresentada a convergência entre as duas áreas: computação paralela e sistemas distribuídos, destacando os principais softwares de apoio à computação paralela distribuída.

2.2 Conceitos gerais

Na computação paralela é comum o emprego dos conceitos concorrência e paralelismo. O primeiro, ocorre sempre que dois ou mais processos iniciam suas execuções e, por motivos de disputa na utilização do recurso, ainda não terminaram. Desse modo, tanto em sistemas uniprocessados como multiprocessados, pode ocorrer concorrência entre os diversos processos submetidos à execução. O segundo, apenas pode existir na presença de dois ou mais processadores, isto é, mais de um processo é executado em um determinado intervalo de tempo (ALMASI; GOTTLIEB, 1994).

Um outro conceito importante no contexto da computação paralela refere-se ao grau de paralelismo ou granulação (granulosidade), que define o tamanho da unidade de trabalho destinada aos processadores. A granulosidade está diretamente relacionada ao hardware, medida em número de instruções e classificada em fina, média e grossa.

A granulosidade fina ou paralelismo em nível de operações, existe sempre que os proces-

sos componentes são tão pequenos quanto às operações ou instruções de máquina. Esse tipo de granulosidade é implementada, geralmente, em hardware (HWANG; XU, 1998). Esse tipo de paralelismo é utilizado com sucesso em sistemas computacionais que apresentam múltiplas unidades funcionais.

Na granulosidade média, o tamanho dos processos é definido em blocos de instruções. Como exemplo, pode-se citar um paralelismo em nível de *loops*, em que um *loop* cria um número de processos, cada qual executando uma interação constituída de um conjunto de instruções de atribuição. Geralmente, a exploração do paralelismo por meio desse nível exige suporte de uma linguagem de programação paralela (HWANG; XU, 1998).

Na granulosidade grossa, divide-se o problema em várias tarefas executado-as paralelamente nos processadores. Dessa maneira, o tamanho dos processos é dado pela quantidade de código das tarefas, ou seja, o tamanho dos processos são os próprios programas executados simultaneamente (ALMASI; GOTTLIEB, 1994).

Segundo Hwang e Xu (1998), aplicações paralelas têm grau de complexidade maior que as aplicações seqüenciais, em relação às suas diferenças na construção dos programas. Dessa maneira, o desenvolvimento de aplicações paralelas segue dois modelos básicos: o paralelismo de dados e o paralelismo funcional.

No paralelismo de dados, também conhecido como modelo SPMD (*Single Program Multiple Data*), cada EP executa a mesma instrução sobre dados diferentes. Já no paralelismo funcional, também conhecido como MPMD (*Multiple Program Multiple Data*), tarefas de uma aplicação têm diferentes instruções que podem ou não operar sobre o mesmo conjunto de dados.

A decisão de qual modelo utilizar deve ser baseada na melhor adequação do modelo ao problema em questão. Outras características, que afetam a construção de aplicações e a granulosidade do paralelismo empregado, são os modos de execução, organização da memória, heterogeneidade dos recursos computacionais e o desempenho da rede de comunicação.

Além dessas características, é importante para a computação paralela avaliar de forma coerente o desempenho de um sistema computacional. Para isso, medidas de desempenho consistentes devem ser empregadas. Tudo isso, geralmente, com o objetivo de demonstrar a vantagem da utilização da computação paralela em relação à execução seqüencial de um determinado algoritmo.

Essas medidas de desempenho estão relacionadas, principalmente, com os objetivos da avaliação, ou seja, para cada situação que se deseja avaliar, determinadas medidas são mais adequadas para os objetivos esperados. A seguir, são descritas algumas medidas de desempenho comumente empregadas na área da computação paralela.

Speedup: mostra o ganho de velocidade na execução de uma aplicação paralela utilizando P elementos de processamento, em relação à execução dessa mesma aplicação em um

único elemento de processamento. Pode ser obtido por meio da expressão:

$$S(P) = \frac{T_1}{T_p} \tag{2.1}$$

onde T_1 é o tempo de execução em um único EP e T_p o tempo de execução em p EPs. Essa definição se aplica a sistemas homogêneos, nos quais todos os processadores são idênticos (capacidade de processamento e memória). Por exemplo, uma aplicação seqüencial que gasta 200 unidades de tempo para ser executada em um único EP e 100 unidades de tempo na execução de sua versão paralela em 4 EPs, o ***Speedup*** é $S(4) = 2$.

Em sistemas heterogêneos (uma rede formada por estações de trabalho ou computadores pessoais com arquiteturas de máquinas diferentes umas das outras) é necessário aplicar uma variação dessa definição.

Dados k diferentes tipos de EPs, $\Sigma EP = EP_1 + ... + EP_k$ o número total de EPs, $T_1(i)$ o tempo para executar o algoritmo seqüencial em um elemento de processamento do tipo i e T_p o tempo para executar o algoritmo paralelo no conjunto de EPs ΣEP (NEARY; CAPPELLO, 2000). O tempo médio ponderado para a execução do algoritmo em um EP T_1, e ***Speedup*** S, são calculados segundo as equações:

$$T_1(EP) = \frac{EP_1 T_1(1) + ... + EP_k T_1(k)}{\Sigma EP} \tag{2.2}$$

$$S = \frac{T_1(EP)}{T_p(EP)} \tag{2.3}$$

A título de ilustração, se uma aplicação seqüencial gasta 100 unidades de tempo para ser executada no EP_1, 200 unidades de tempo no EP_2, 100 unidades de tempo no EP_3 e 200 unidades de tempo no EP_4, gastando 50 unidades de tempo na execução de sua versão paralela com 4 EPs, o seu ***Speedup*** é de $S(4) = 3$.

Eficiência: reflete a taxa de utilização dos processadores obtida com o processamento paralelo utilizando P EPs e pode ser calculada por:

$$E(P) = \frac{S(P)}{P} \tag{2.4}$$

O ***Speedup*** ideal é aquele igual ao número de EPs P e o valor ótimo para a ***Eficiência*** é $E = 1$. Geralmente, o ***Speedup*** tem valores menores ao ideal devido a fatores tais como a sobrecarga de comunicação entre os EPs, as porções de código que devem ser executadas obrigatoriamente em seqüência e os tempos de processamento gastos na criação e sincronização de tarefas. Aplicando os exemplos mencionados anteriormente para o cálculo do ***Speedup*** em sistemas homogêneos ($S = 2$) a ***Eficiência*** é de $0,5$ (50%), e no exemplo com sistemas heterogêneo ($S = 3$) a ***Eficiência*** é de $0,75$ (75%).

2.3 Arquiteturas paralelas

A taxonomia de Flynn (FLYNN; RUDD, 1996) é uma classificação amplamente usada para arquiteturas de máquinas, tanto paralelas quanto seqüenciais. A base de sua classificação é a adoção de dois conceitos, o fluxo de dados e o fluxo de instruções. Fluxo de instruções são seqüências de instruções executadas por um determinado computador e fluxo de dados é a seqüência de dados manipulada pelo fluxo de instruções (QUINN, 1994).

Dessa maneira, as arquiteturas podem ser classificadas segundo os fluxos de dados e instruções em quatro categorias: SISD, SIMD, MISD, MIMD.

- SISD (*Single Instruction Single Data stream*): fazem parte dessa categoria as máquinas de von Neumann, cuja execução é seqüencial. Ainda que instruções possam ser sobrepostas (técnicas de *pipeline* [1]), esses computadores, geralmente, decodificam uma única instrução num determinado intervalo de tempo. Essas máquinas podem ter múltiplas unidades funcionais, porém estão ligadas a uma única unidade de controle. Dessa maneira, elas são caracterizadas por ter um único fluxo de instruções e de dados;

- SIMD (*Single Instruction Multiple Data stream*): vários EPs executam um único fluxo de instruções, mas atuam sob múltiplos dados. Podem ter várias unidades lógicas aritméticas, cada uma busca e manipula um conjunto próprio de dados. Dessa maneira, em determinado instante, tem-se uma única operação no mesmo estado de execução em múltiplas unidades de processamento. Nesse caso, tais unidades de processamento são gerenciadas por uma única unidade de controle;

- MISD (*Multiple Instruction Single Data stream*): essa categoria apresenta múltiplos fluxos de instruções atuando em um único fluxo de dados. Na literatura não existem exemplos precisos de máquinas MISD que seguem essa classificação, porém, alguns autores (ALMASI; GOTTLIEB, 1994; NAVAUX, 1989) consideram a técnica de *pipeline* como representante dessa categoria.

- MIMD (*Multiple Instruction Multiple Data stream*): nessa categoria múltiplos fluxos de instruções operam sobre múltiplos fluxos de dados, que englobam a maioria dos computadores paralelos.

Arquiteturas MIMD apresentam algumas diferenças em relação às SIMD, por exemplo menor número de processadores, porém com maior potência computacional. Arquiteturas MIMD têm capacidade de aumento em escala, isto é, a potência computacional do sistema como um todo, pode ser adequada segundo as necessidades do usuário. MIMD são mais flexíveis e por

[1] A técnica *pipeline* representa a execução de eventos sobrepostos no tempo. Uma tarefa é subdividida em partes e cada parte é executada num estágio de hardware especializado

isso são muito utilizadas na execução de aplicações paralelas de propósito geral. A maneira como a memória é organizada pode ser tanto compartilhada entre os EPs da máquina paralela, quanto distribuída. Um exemplo de arquitetura MIMD com memória distribuída são os sistemas distribuídos.

2.4 Sistemas distribuídos

Os sistemas computacionais distribuídos foram desenvolvidos para atender à demanda por processamento das aplicações e à necessidade de compartilhamento de recursos. O desenvolvimento desses sistemas teve mudanças significativas a partir da década de 1980, quando novos conceitos começaram a ser empregados. Esses conceitos estão relacionados ao uso de redes de computadores e à aplicação de técnicas de paralelismo para obtenção de maior desempenho.

A pesquisa e o desenvolvimento de sistemas distribuídos foram direcionados segundo alguns fatores, tais como a evolução das redes de computadores, o aumento da largura de banda dessas redes, o desenvolvimento de novos protocolos de comunicação, a facilidade de interconexão de computadores em redes, a evolução dos microprocessadores e, conseqüentemente, o aumento da capacidade de processamento e o alto custo na aquisição de máquinas paralelas.

Esses sistemas visam a transparência de operações, tolerância a falhas, possibilidade de crescimento em escala e desempenho. Tais características, aliadas à boa relação custo benefício, tornam viáveis a utilização de sistemas distribuídos como plataforma para execução de aplicações paralelas (TANENBAUM; STEEN, 2002; COULOURIS et al., 2001; MULLENDER, 1993).

Sistemas distribuídos são constituídos por computadores autônomos que trabalham cooperativamente, na execução de tarefas dos usuários. Para esses usuários, o sistema deve se apresentar de forma transparente como se fosse um único recurso de processamento (ou máquina paralela virtual). Duas vantagens importantes podem ser obtidas com a utilização dessa abordagem. A primeira é a facilidade na integração de diferentes aplicações, que executam em computadores distintos, a segunda é relacionada à fase de desenvolvimento, onde é possível projetar o ambiente distribuído para atender aplicações específicas.

Essas vantagens agregam custo e complexidade ao software distribuído, podendo degradar o desempenho e diminuir a segurança. Entretanto, existe considerável interesse na construção e instalação de sistemas distribuídos em diversas áreas, tais como comércio eletrônico, sistemas acadêmicos, laboratórios de pesquisa, etc.

Sistemas distribuídos, geralmente, são construídos adicionando-se uma camada de software sobre um sistema operacional de rede (figura 2.1 (TANENBAUM; STEEN, 2002)). Essa camada, chamada *middleware*, é projetada para ocultar a heterogeneidade e a localização distribuída dos computadores. Os *middlewares* podem adotar um modelo específico para a distribuição e comunicação dos processos submetidos ao sistema. Esses modelos são baseados em chamadas

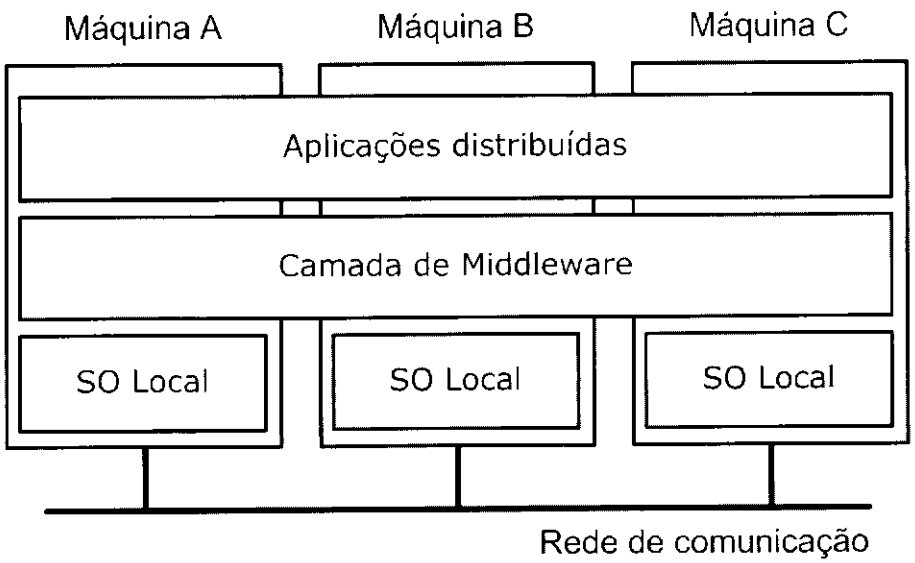

Figura 2.1: Exemplo de um sistema distribuído

de procedimento remoto (RPC), objetos, sistemas de arquivos e documentos distribuídos.

2.5 Computação paralela distribuída

Os sistemas computacionais paralelos constituídos por estações de trabalho e computadores pessoais, voltados para aplicação em computação paralela distribuída são também conhecidos como NOWs (*Network of Workstations*) (ANDERSON et al., 1995; ARPACI et al., 1995) e COWs (*Cluster of Workstations*) (ANASTASIADIS; SEVCIK, 1997; CHIOLA; CIACCIO, 2000; ZHANG et al., 2000). Nesses sistemas nota-se o acréscimo de fatores que necessitam de cuidados especiais para não comprometer o desempenho global. Esses fatores se referem à heterogeneidade dos recursos e a interatividade do sistema.

A heterogeneidade existe quando os EPs apresentam diferentes arquiteturas, com diferentes formas de representação de dados e potência computacional (heterogeneidade arquitetural). Entretanto, caso existam EPs com a mesma arquitetura, mas com diferentes potências computacionais, quantidade de memória e diferentes atrasos na rede de comunicação, a heterogeneidade é denominada configuracional. Utilizando-se computadores paralelos, pessoais e estações de trabalho para se obter um grande sistema computacional de alto desempenho, tem-se à partir disso um ambiente heterogêneo.

O ponto em questão é a utilização da computação paralela sobre sistemas heterogêneos distribuídos e, para isso, é necessária toda uma infraestrutura de software paralelo com mecanismos eficazes, que permitam a distribuição de tarefas e a gerência de recursos de forma transparente.

Com a utilização de diversas arquiteturas de máquinas, junto a grande variação de potência computacional, agregam-se novas dificuldades que necessitam ser gerenciadas pelo software

paralelo. Aspectos de arquitetura e configuração, potencialmente distintos, dos EPs que compõem o sistema computacional requerem mecanismos eficientes para o gerenciamento.

Uma solução empregada e que torna possível a utilização da computação paralela em sistemas distribuídos heterogêneos é a utilização de softwares que permitem a distribuição de tarefas e a gerência de recursos de maneira transparente. Esses softwares permitem o emprego de várias arquiteturas constituindo um único sistema, formado por computadores pessoais, estações de trabalho e máquinas paralelas, interconectadas por meio de tecnologias diferentes de comunicação. Para o desenvolvimento de aplicações paralelas utilizam-se amplamente o PVM e o MPI, que são discutidos nas subseções 2.6.1 e 2.6.2, respectivamente.

2.6 Softwares de suporte à computação paralela distribuída

A programação paralela é fundamental no suporte à construção de aplicações paralelas por oferecer recursos, que podem ser utilizados visando maior desempenho e melhor utilização do hardware paralelo disponível (QUINN, 1994).

Na programação seqüencial, esses recursos (instrução que especificam operações aritméticas, endereço de dados a serem lidos da memória e endereço da próxima instrução que será executada) são utilizados por meio de uma linguagem de montagem, ou, de uma maneira mais usual, por linguagens de alto nível como C, Pascal ou Fortran.

A programação paralela necessita de recursos não disponíveis diretamente nessas linguagens. Portanto, são necessários mecanismos para definir quais tarefas serão executadas paralelamente, mecanismos para a ativação e finalização dessas tarefas e métodos para coordenar a interação entre elas.

Quinn (1994) sugere três maneiras para construção de um algoritmo paralelo com o objetivo de melhorar o desempenho e a utilização do hardware disponível:

- Detectar e explorar algum paralelismo inerente a um algoritmo seqüencial existente. Amplamente utilizada, porém essa estratégia resulta em baixo *speedup*.
- Elaborar um novo algoritmo paralelo. É a estratégia que possibilita maior desempenho, porém exige a reconstrução completa do algoritmo.
- Utilizar um outro algoritmo que resolve um problema similar. Essa técnica alia bom desempenho com menor esforço para o programador, se comparada à construção completa do algoritmo.

A computação paralela utiliza o princípio de dividir determinadas aplicações em partes menores, cada qual solucionando uma porção do problema. No entanto, com a aplicação dessa

técnica surgem problemas como a necessidade de ativação, comunicação e sincronização de processos, desnecessários na computação seqüencial.

Um determinado processo pode interagir com outro por meio de variáveis compartilhadas, em sistemas de memória compartilhada, ou através de troca de mensagens em sistemas de memória distribuída. Para se obter sincronização, é necessário manter a consistência dos dados, uma vez que são dados compartilhados e sofrem acesso simultâneo.

Para que ocorra controle na seqüência de execução paralela, o sincronismo é fundamental para a coerência dos resultados esperados na computação. Em um ambiente com memória compartilhada, geralmente, utilizam-se mecanismos tais como monitores e semáforos. Para obter sincronismo dos processos, em ambientes de memória distribuída, é comum a utilização de mecanismos de troca de mensagens, tais como RPC (*Remote Procedure Call*) (QUINN, 1994), RMI (*Remote Method Invocation*) (COULOURIS et al., 2001) e comunicação ponto-a-ponto (ALMASI; GOTTLIEB, 1994).

Outro aspecto importante para construção de programas paralelos está relacionado com o tipo de ferramenta utilizada no desenvolvimento dos códigos fontes. Isso pode ser realizado de três maneiras: utilizando linguagens de programação concorrentes, ambientes de paralelização automáticos ou ambientes de passagem de mensagens (ALMASI; GOTTLIEB, 1994).

As linguagens de programação permitem a criação de programas paralelos com a utilização de construções próprias das linguagens, necessárias para a implementação dos algoritmos.

Os ambientes de paralelização automática possibilitam a tradução de um código fonte seqüencial em um código paralelo. Para isso, busca-se absorver o paralelismo existente no código e gera-se um programa paralelo sem a necessidade de intervenção do programador.

Ambientes de passagem de mensagens são ambientes que permitem a criação, comunicação e sincronismo entre tarefas. Tratam-se de extensões para as linguagens seqüenciais, que são implementas por meio de bibliotecas de passagem de mensagens. A utilização desse mecanismo possibilita a computação paralela em sistemas de memória distribuída.

Outra possibilidade de implementação de aplicações paralelas é através do uso de objetos distribuídos. Essa abordagem alia os conceitos de orientação a objetos, encapsulamento, herança, reuso, etc, ao desenvolvimento de aplicações paralelas em ambientes de passagem de mensagens. Caracteriza-se pela utilização de um middleware [2] para gerência de objetos distribuídos (MOWBRAY; MALVEAU, 1997; CHIANG, 2001).

Exemplos de softwares que oferecem suporte ao desenvolvimento de aplicações paralelas são o PVM, o MPI e implementações CORBA, que são discutidos nas próximas subseções.

[2]Middleware é um software que faz a mediação entre um programa aplicativo e a rede de comunicação. Ele gerencia a interação entre as aplicações que executam sobre plataformas computacionais heterogêneas. O ORB (*Object Request Broker*), software que gerencia a comunicação entre objetos, é um exemplo de middleware

2.6.1 PVM (*Parallel Virtual Machine*)

O PVM é um conjunto integrado de bibliotecas e ferramentas de software, cuja finalidade é emular um sistema computacional concorrente heterogêneo, flexível e de propósito geral (GEIST et al., 1994). Como seu próprio nome sugere, permite que um conjunto de plataformas paralelas heterogêneas, conectadas em rede, seja utilizado como uma arquitetura paralela de memória distribuída, formando uma máquina paralela virtual.

O PVM fornece as funções que permitem ao usuário iniciar, comunicar e sincronizar as tarefas na máquina virtual. Uma tarefa do PVM é definida como uma unidade computacional análoga aos processos UNIX (freqüentemente é um processo UNIX). O modelo computacional do PVM é, portanto, baseado na notação de que uma aplicação consiste de várias tarefas, cada uma responsável por uma parte da carga de trabalho da aplicação.

No início de seu desenvolvimento, o PVM foi construído para ser utilizado em estações de trabalho UNIX. Atualmente, encontram-se implementações do PVM em uma variedade de plataformas, como por exemplo, computadores pessoais executando Linux e Windows ou máquinas maciçamente paralelas. Algumas características principais do PVM são (GEIST et al., 1994):

- Permite a configuração dinâmica da máquina virtual: existe uma coleção de máquinas que formam um *host pool*, máquinas com apenas um processador ou arquiteturas multiprocessadas (memória compartilhada ou distribuída) configuradas dinamicamente pelo usuário, que constituem uma plataforma de execução das aplicações;

- Transparência de acesso ao hardware: não existe a necessidade de distinção ou adaptação nos métodos de criação e comunicação entre as tarefas. As aplicações paralelas tanto podem ver o ambiente de execução como um conjunto de EPs virtual, tanto podem optar por explorar as capacidades de uma determinada máquina do *host pool*, atribuindo tarefas aos computadores mais apropriados;

- Computação baseada em processos: no PVM a unidade de paralelismo é a tarefa, que altera seu estado de execução entre computação e comunicação. É possível a execução de múltiplas tarefas em um mesmo EP;

- Modelo de passagem de mensagens: o conjunto de tarefas que são executadas cooperam e comunicam entre si por meio de troca de mensagens;

- Suporte à heterogeneidade: no sistema PVM existe suporte à heterogeneidade em termos de arquitetura, rede de comunicação e aplicações. Com base no modelo de passagem de mensagens, ele permite mensagens com mais de um tipo de dados transmitidos entre máquinas com diferentes representações de dados;

- Suporte a arquiteturas paralelas: o PVM usa os recursos do modelo de passagem de mensagens nativo em multiprocessadores para obter vantagens inerentes aos diferentes tipos de hardwares. Fabricantes, geralmente, fornecem suas próprias implementações otimizadas do PVM para seus sistemas, os quais podem se comunicar com versões de domínio público; diferentes versões do PVM podem interagir entre si.

O PVM é formado por duas partes principais. A primeira é constituída por uma coleção de processos *daemon* [3] (*pvmd*), que são executados em todos os EPs que fazem parte da máquina virtual.

O *pvmd*, que faz parte da máquina virtual de um usuário, não tem acesso a outro *pvmd*, pertencente à outra máquina virtual. É projetado para executar sobre um *login*, sem privilégios, o que reduz os riscos de uma máquina virtual interferir na execução de outras. Dessa maneira, múltiplos usuários podem configurar máquinas virtuais sobrepostas, e cada usuário pode executar várias aplicações PVM simultaneamente. O *pvmd* não faz processamento, mas atua exclusivamente como um roteador e controlador de mensagens, agindo como um ponto de contato entre cada computador da máquina paralela virtual (GEIST et al., 1994).

Em arquiteturas de máquinas multiprocessadas e memória distribuída, as primitivas de comunicação são mapeadas de forma direta nas chamadas nativas do sistema. Já em máquinas multiprocessadas com memória compartilhada, as primitivas de comunicação são implementadas utilizando buffers armazenados em memória.

2.6.2 MPI (*Message Passing Interface*)

O MPI é uma padronização, independente do sistema computacional paralelo, para ambientes de programação via troca de mensagens. O MPI surgiu da necessidade de se resolver alguns problemas relacionados às plataformas de portabilidade [4](P4, PARMACS, EXPRESS, PVM, LINDA, entre outros), por exemplo:

- o grande número de plataformas existentes ocasiona restrições em relação à real portabilidade de programas;
- o mau aproveitamento de características de algumas arquiteturas paralelas.

O processo de desenvolvimento do MPI iniciou em abril de 1992. Em novembro do mesmo ano, uma primeira versão foi apresentada (MPI 1). Em novembro de 1993 foi apresentada

[3] É um processo que executa em segundo plano e realiza uma operação específica em tempos pré-determinados, ou em resposta a certos eventos. O termo daemon é um termo UNIX, e alguns exemplos típicos são os processos que controlam a fila de impressão, correio eletrônico, etc.

[4] Ambientes para construção de aplicações paralelas que utilizam passagem de mensagens, ou seja, cada EP tem sua própria memória e comunica-se por meio de troca de mensagens.

a especificação do padrão MPI versão 1.0, sendo esta publicada em maio de 1994. Em junho de 1995, foi publicada a versão 1.1, apresentando correções de erros e melhor esclarecimento de determinadas propriedades do MPI. Em agosto de 1997 foi apresentada a versão 2.0, concluindo os trabalhos do Fórum MPI (SKJELLUM et al., 1994; MPI Forum, 2004b; MPI Forum, 2004a).

Considerando-se o limite de tempo imposto à formulação inicial (versão 1.0) do padrão, definiu-se um conjunto básico de rotinas relacionadas à comunicação ponto-a-ponto e coletiva, deixando para acrescentar novas rotinas e funcionalidades extras a partir da versão 2.0 (MPI Forum, 2004b; MPI Forum, 2004a).

O MPI 1.0 (SNIR et al., 1998) define um conjunto de 129 rotinas, que oferecem os seguintes serviços :

- Comunicação ponto-a-ponto: rotinas de comunicação que envolvem exatamente dois processos (sempre dentro de um grupo específico);
- Comunicação coletiva: rotinas de comunicação que envolvem dois ou mais processos;
- Suporte para grupos de processos: o MPI relaciona os processos em grupos, e estes processos são identificados pela sua classificação dentro desse grupo;
- Suporte para contextos de comunicação: contextos definem escopos que relacionam processos em grupos;
- Suporte para topologia de processos: o MPI fornece primitivas que permitem ao programador definir a estrutura topológica com a qual os processos de um determinado grupo se relacionam.

Um grande número de implementações MPI efetivam todas as rotinas definidas na versão 1.1 do padrão (MPICH e LAM, por exemplo), o que demonstra que essa versão conseguiu significativa aceitação na comunidade (SOUZA, 1997).

A versão 2.0, concluída em agosto de 1997, apresenta diversas correções e esclarecimentos à versão 1.1, além de definir conjuntos de novas rotinas. Entre as operações definidas a partir da versão 2.0, incluem-se (MPI Forum, 2004b; MPI Forum, 2004a):

- Gerenciamento de processos: rotinas de início, término e controle de processos. Pode-se fazer a comunicação entre processos iniciados em diferentes aplicações, através de rotinas de publicação de portas, ou seja, um processo define uma porta de comunicação global que qualquer outro processo MPI pode acessar;
- Entrada e saída paralelas: rotinas que permitem vários processos acessarem um arquivo concorrentemente;

- Rotinas coletivas estendidas: definição de rotinas coletivas que envolvem diferentes grupos de processos;

- Comunicação unilateral (*one-sided*): definem as rotinas de acesso remoto à memória (RMA - *Remote Memory Acess*), no qual o transmissor é responsável por controlar a transmissão e a recepção de uma mensagem. Esses tipos de rotinas são bastante utilizadas em computadores paralelos e têm como grande vantagem possibilitar rotinas de comunicação assíncronas.

Como o principal objetivo do MPI é garantir eficiência a qualquer sistema computacional paralelo, as funções básicas de sua biblioteca de troca de mensagens têm diversas variações entre as diferentes implementações existentes. Várias implementações do MPI, tanto comerciais como de domínio público, já estão disponíveis e várias outras devem ser lançadas. Empresas que já lançaram versões comerciais são: IBM, Meiko, Intel, Cray Research, Ncube, HP, Silicon Graphics, NEC e Telmat (LAM Team, 2004). O Ohio SuperComputer Center relaciona nove implementações de domínio público, no qual destacam-se os sistemas MPICH, CHIMP e LAM (LAM Team, 2004).

2.6.3 CORBA (*Common Object Request Broker*)

CORBA é um padrão para o desenvolvimento de aplicações distribuídas em diferentes domínios de atuação. É uma especificação definida pela OMG (*Object Management Group*), que permite a execução de aplicações distribuídas utilizando o paradigma de orientação a objetos. Possibilita o desenvolvimento de aplicações extensas e complexas, não prejudicando o suporte e o reuso de componentes. Além disso, conduzem o emprego da orientação a objetos em ambientes de aplicações distribuídas.

As aplicações definidas em CORBA utilizam a técnica de orientação a objetos e o modelo cliente/servidor fornecendo aos usuários uma visão uniforme e independente de localização. São formadas por um conjunto de componentes de software independentes, denominados objetos CORBA. Uma aplicação que requisita uma determinada operação é chamada de cliente; uma outra que executa as operações requisitadas por um cliente, denomina-se servidor. Dessa maneira, uma aplicação pode, em instantes distintos, comportar-se tanto como cliente quanto como servidor (MOWBRAY; MALVEAU, 1997).

A linguagem de especificação dos objetos clientes, os métodos e sua operações é denominada IDL (*Interface Definition Language*). O principal objetivo é fornecer uma interface de software utilizando uma notação própria, que pode ser mapeada para diferentes linguagens de programação. Aplicações clientes e servidores são independentes da linguagem de programação utilizada em sua implementação, bastando para isso, que a linguagem tenha um mapeamento para a IDL. Um objeto implementado na linguagem Java pode ser invocado por um cliente

implementado em C++, por exemplo (MOWBRAY, 1997).

A comunicação entre clientes e servidores na arquitetura CORBA é realizada através de um componente chamado ORB (*Object Request Broker*), que é o gerenciador de requisições. Ele permite a invocação direta entre os objetos, possibilita comunicação de forma transparente por meio do atendimento de requisições dos clientes e do repasse dessas requisições a objetos servidores (VINOSKI, 1997).

Quando a especificação CORBA foi definida, não se preocuparam com a possibilidade de utilizá-la para o desenvolvimento de aplicações paralelas. Entretanto, pode-se obter paralelismo com objetos implementados em várias *threads* [5].

Dessa maneira, nenhuma modificação na especificação do padrão CORBA é necessária. Além disso, permite-se a exploração de eventos concorrentes em vários processadores, que acessam a mesma memória principal compartilhada. Porém, essa forma de paralelismo é limitada ao hardware. Conforme aumenta-se a quantidade de objetos executados na mesma máquina, aumenta-se a disputa durante o acesso à memória principal, gerando problemas e atrasos inconvenientes.

Para fornecer paralelismo entre objetos localizados em máquinas diferentes, extensões da interface de definição foram propostas, bem como mecanismos assíncronos de invocação de objetos. Alguns exemplos são o COBRA (PRIOL; RENÉ, 1998) e PARDIS (KEAHEY; GANNON, 1997) que utilizam o modelo SPMD para o desenvolvimento de aplicações paralelas, e o PCB (XU; HE, 2000), que aborda tanto o modelo SPMD quanto o modelo MPMD.

Santos (2001) apresenta, por intermédio de alterações em uma implementação da especificação CORBA, uma extensão eficiente dos mecanismos de distribuição de tarefas. Também mostra sua integração com o software de escalonamento, sem alterações na interface de definição. Esse trabalho demonstra que o desempenho alcançado, empregando implementações eficientes desse modelo, é comparável ao desempenho obtido com ferramentas construídas especificamente para a computação paralela (SANTOS, 2001). Esse fato deve ser ressaltado, pois CORBA não foi criado para construção de aplicações paralelas.

2.7 Considerações finais

Este capítulo apresentou os principais conceitos de sistemas distribuídos. As características desses sistemas, aliadas à possibilidade de baixo custo de aquisição e manutenção, tornam viáveis sua utilização em computação paralela.

A computação paralela, antes constituída apenas por máquinas paralelas caracteriza-

[5] Parte de um programa que pode ser executada independentemente de outras partes. Sistemas Operacionais que suportam *multithreading* permitem que os programadores construam aplicações cujas *threads* possam ser executadas concorrentemente.

das pelo alto custo de aquisição e manutenção, baixa flexibilidade e dificuldade de programação, atualmente, pode se utilizar dos conceitos de sistemas distribuídos. Utilizando-se tais conceitos podem-se projetar máquinas paralelas virtuais onde os elementos de processamento encontram-se distribuídos pela rede de comunicação.

Máquinas paralelas virtuais apresentam algumas diferenças significativas em relação às máquinas paralelas reais. Máquinas paralelas são constituídas por vários elementos de processamento, geralmente homogêneos, interconectados por uma rede de comunicação dedicada e de alta velocidade. Por outro lado, máquinas paralelas virtuais apresentam elementos de processamento potencialmente heterogêneos, tanto em termos de configuração quanto de arquitetura. Além disso, a rede de comunicação é de menor velocidade e compartilhada por diferentes usuários e aplicações, que influenciam, substancialmente, no desempenho do sistema.

Essas diferenças criam novas necessidades de software tais como a adequação de técnicas de distribuição eficiente dos processos entre os elementos de processamento do sistema. Essas técnicas que buscam oferecer boas decisões de alocação de recursos e, conseqüentemente, maior desempenho.

Neste trabalho, uma técnica, que considera a rede de comunicação como um parâmetro para decisões de escalonamento, é analisada, bem como é demonstrada sua viabilidade e aplicabilidade sobre um sistema computacional distribuído executando aplicações paralelas com grande intensidade de operações em rede. Com o objetivo de avaliar o desempenho tanto do sistema computacional quanto da nova estratégia de escalonamento de processos, o próximo capítulo discute as principais técnicas para avaliação de desempenho.

CAPÍTULO

3

Avaliação de desempenho

3.1 Considerações iniciais

Este capítulo apresenta as principais técnicas de avaliação de desempenho existentes. Toda e qualquer atividade envolvida no processo computacional pode e deve ser avaliada, a fim de verificar suas expectativas de eficiência e confiabilidade. Neste trabalho, busca-se avaliar o desempenho sobre duas óticas: a primeira é a avaliação de desempenho das políticas de escalonamento com o objetivo de compará-las; a segunda, é a avaliação do sistema em termos de ocupação dos elementos de processamento e a carga imposta sobre a rede de comunicação.

O escalonamento de processos, como qualquer outra atividade computacional, pode ser avaliado usando técnicas específicas que forneçam melhores resultados. Técnicas de avaliação de desempenho podem ser aplicadas em diversas fases do desenvolvimento e no próprio uso de uma determinada política de escalonamento.

Para avaliar um sistema, utilizam-se técnicas que estimam seu comportamento em diferentes situações de carga. Essas técnicas de avaliação de desempenho fornecem resultados numéricos, que permitem comparar diferentes soluções para o mesmo problema. Tais resultados são denominados métricas de desempenho, que são responsáveis por formalizar e quantificar os objetivos da avaliação. A métrica de desempenho fornece um valor que permite afirmar se o desempenho é bom ou ruim, dependendo dos objetivos preestabelecidos.

As técnicas de avaliação de desempenho podem ser subdivididas em dois grupos, com diferentes aplicações e características, que são: técnicas de aferição e técnicas de modelagem (ORLANDI, 1995; FRANCÊS, 2001), as quais são apresentadas nas seções seguintes.

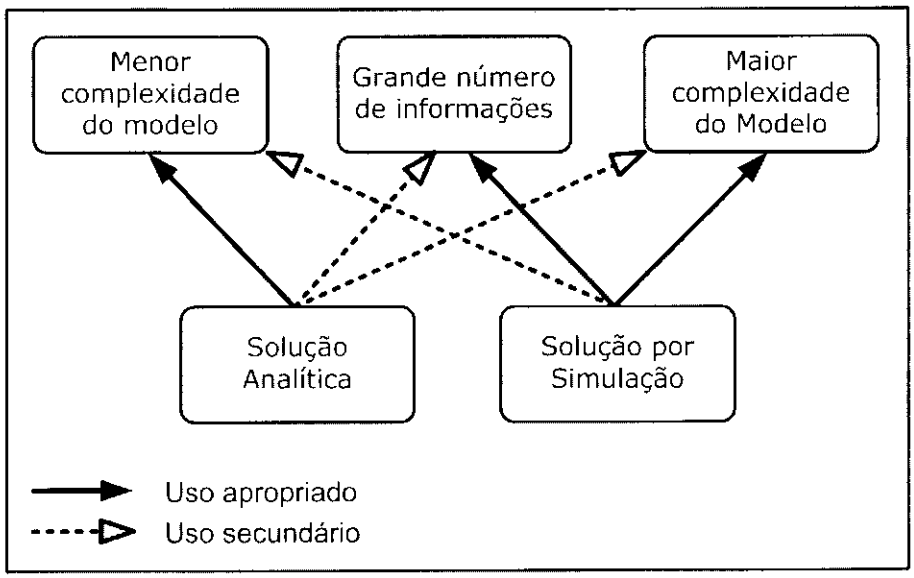

Figura 3.1: Soluções para o modelo de avaliação

3.2 Técnicas de modelagem

As técnicas de modelagem podem ser subdivididas em duas fases. Na primeira, é construído um modelo que representa o sistema avaliado. Esse modelo não precisa representar o sistema completo, mas sim as características relevantes para a avaliação. A modelagem também pode ser aplicada na avaliação de um sistema não existente, sendo possível experimentá-lo e aperfeiçoá-lo antes de sua implementação. Isso é interessante, pois evitam-se custos extras no desenvolvimento do sistema.

A segunda fase, diz respeito à resolução do modelo criado na primeira. O modelo pode ser resolvido de duas maneiras: analiticamente ou por simulação. Técnicas analíticas fornecem a solução do modelo por meio de equações matemáticas, que através de probabilidades, demonstram o comportamento do sistema. Essa técnica, geralmente, é aplicada em sistemas de baixa complexidade.

Na simulação é desenvolvido um programa computacional que representa o modelo. Tal programa é submetido a experimentos, que corretamente analisados, resultam no comportamento do sistema (BANKS, 1998).

A decisão por uma das duas técnicas de solução do modelo, analítica ou simulação, depende do sistema em estudo, do objetivo da avaliação, da quantidade de parâmetros disponíveis, etc. A figura 3.1 (ORLANDI, 1995) apresenta uma proposta de quando utilizar cada uma das duas técnicas.

As principais ferramentas utilizadas para modelagem de sistemas e que visam representar seu comportamento são: Redes de Petri, Cadeias de Markov, *Statecharts* e Redes de Filas, .

As Redes de Petri permitem representar sistemas paralelos, concorrentes e assíncronos,

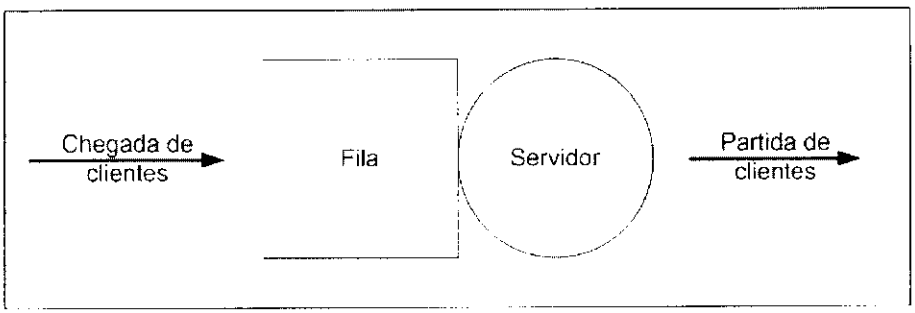

Figura 3.2: Modelo de redes de filas com um centro de serviço

utilizando conceitos matemáticos (MARSAN et al., 1998). Essa técnica se baseia em lugares, transições, arcos dirigidos e marcas. Os lugares correspondem a possíveis situações do sistema. As transições são os eventos que movem as marcas de um lugar para outro. Os arcos dirigidos representam as possíveis transições entre lugares. As marcas correspondem ao estado atual de um sistema (ZHOU; VENKATESH, 1999).

As Cadeias de Markov modelam sistemas de acordo com seus estados e suas probabilidades de transações. Uma cadeia de Markov é um processo estocástico que tem uma propriedade especial denominada *Memoryless*. Essa propriedade diz que uma transição para um estado depende somente do seu estado atual (KLEINROCK, 1976; FRANCÊS et al., 2000).

A técnica *Statecharts* é um extensão às máquinas de estado finito, que possibilita a representação de hierarquia, concorrência e comunicação entre os diversos estados de um determinado sistema (HAREL, 1987). Assim como Redes de Petri, baseiam-se na mudança de seus estados, diante da ocorrência de uma determinada ação.

Outra técnica amplamente utilizada e que este trabalho se baseia são as Redes de Fila, que são constituídas de entidades, denominadas clientes, servidores e filas. Os servidores são responsáveis por atender aos clientes de uma fila. As filas representam áreas de espera de clientes cujos pedidos excedem a capacidade do servidor. A chegada de clientes e o processo de atendimento pelos servidores seguem funções de distribuição de probabilidades, as quais definem o comportamento do sistema (SANTANA et al., 1994; LAVENBERG, 1983).

A figura 3.2 representa um centro de serviço formado por um servidor e uma fila. Os clientes chegam ao centro de serviço, esperam na fila se necessário, são atendidos pelo servidor e deixam o centro. Esse centro de serviço e os clientes constituem um modelo de redes de filas.

Esse modelo tem dois parâmetros. O primeiro especifica a intensidade da carga de trabalho, isto é, a taxa de chegada dos clientes (um cliente a cada 2 segundos ou 0.5 clientes/segundo). O segundo especifica a demanda de serviço, que é o tempo de serviço médio requerido por um cliente (1.25 segundos, por exemplo). Para determinar esses parâmetros, é possível avaliar tal modelo através da solução de equações simples, resultando em métricas de desempenho, tais como *utilização* (proporção do tempo de ocupação do servidor), *tempo residente* (tempo médio gasto por um centro de serviço atendendo um cliente), *tamanho da fila* (número médio de clientes

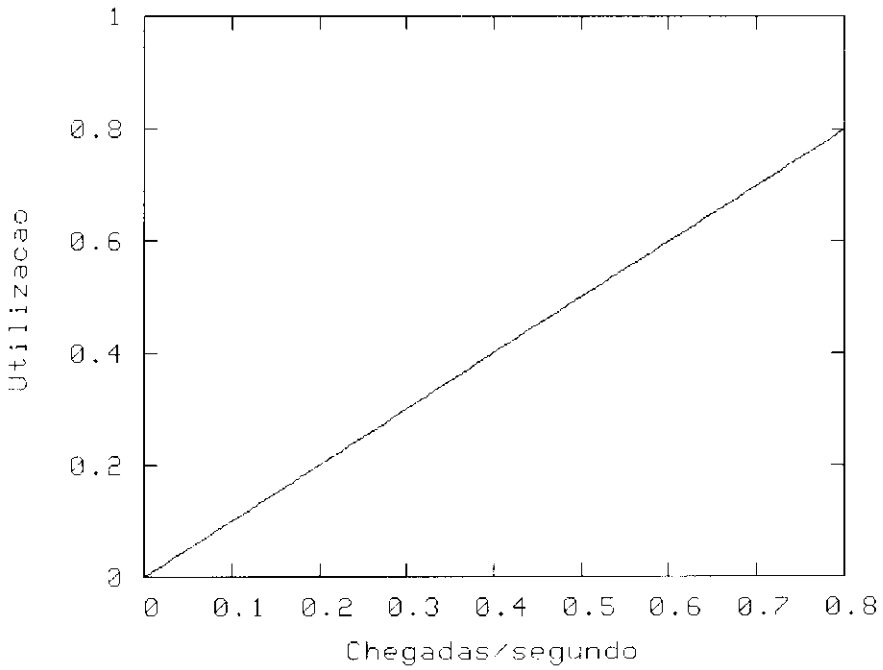

Figura 3.3: Utilização

no centro de serviço), *throughput* (taxa de clientes que passam pelo centro de serviço).

Para exemplificar, considerem-se os valores dessas métricas de desempenho:

- *utilização*: 0.625
- *tempo residente*: 3.33 segundos
- *tamanho da fila*: 1.67 clientes
- *throughput*: 0.5 clientes/segundo

As figuras 3.3, 3.4, 3.5 e 3.6 demonstram cada uma dessas métricas com uma intesidade de carga de trabalho variando de 0.0 a 0.8 chegadas/segundo. No ponto inferior do gráfico, não existe problema de saturação para taxa de chegada. No ponto alto, dado que o requisito de serviço médio é 1.25 segundos, a maior taxa possível no qual o centro de serviço pode tratar é um cliente a cada 1.25 segundos, ou 0.8 clientes/segundo; se a taxa de chegada for maior que isso, então o centro de serviço satura.

O que se observa nessas figuras é que a avaliação do modelo produz métricas de desempenho que são qualitativamente consistentes. Considerando a métrica *tempo residente*, quando a intensidade da carga de trabalho é baixa, espera-se que uma chegada de cliente, raramente, encontrará competição pelo centro de serviço. Dessa maneira, o cliente é atendido imediatamente e terá *tempo residente* aproximadamente igual ao seu tempo de serviço. Como a intensidade da carga de trabalho se eleva, aumenta-se o congestionamento e o tempo residente aumenta com ele. Inicialmente, esse aumento é gradual. Como a carga cresce, o tempo residente aumenta em

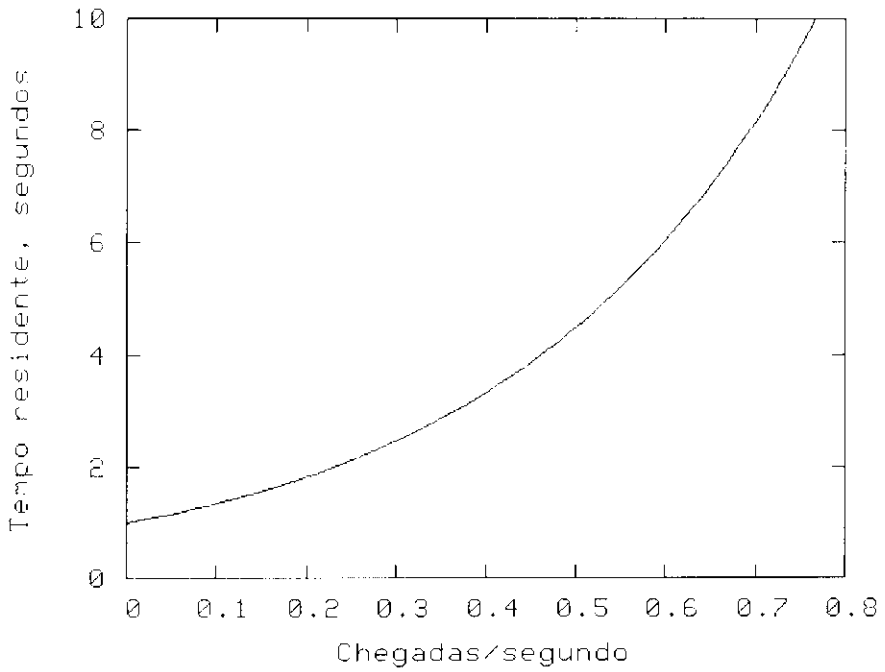

Figura 3.4: Tempo residente em fila e serviço

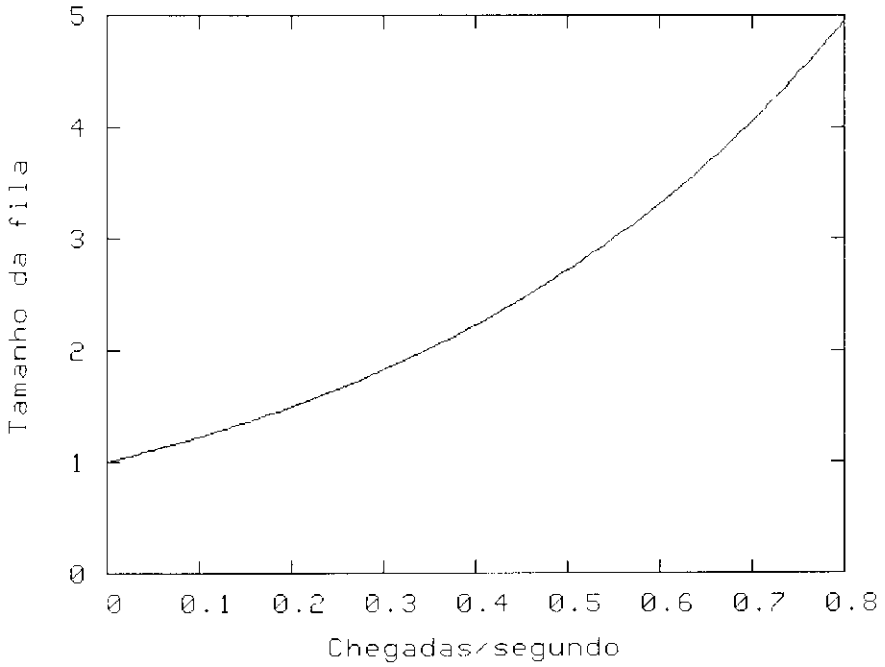

Figura 3.5: Tamanho da fila de espera

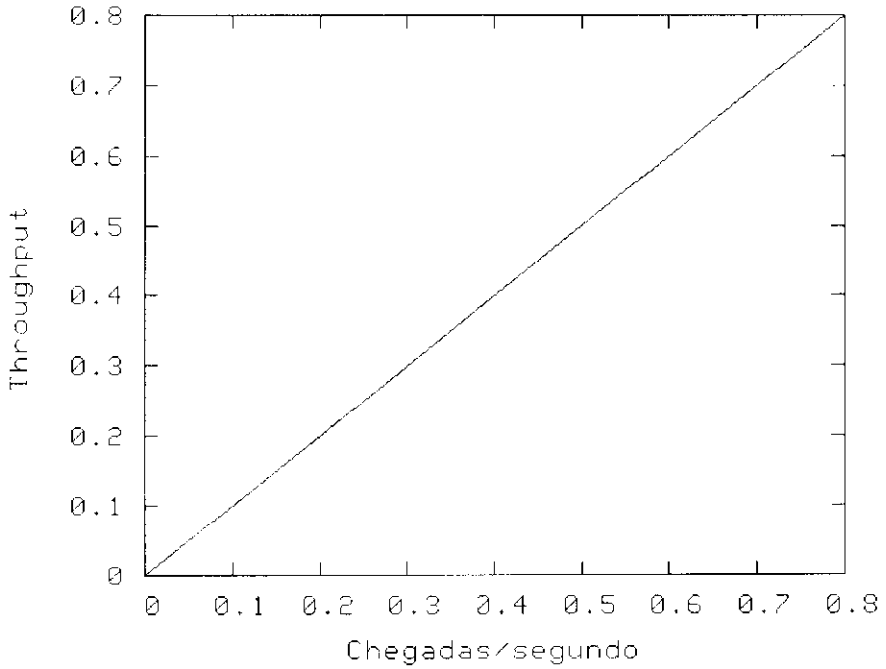

Figura 3.6: Throughput do centro de serviço

taxas cada vez mais velozes, até que o centro de serviço satura. Quaisquer aumentos na taxa de chegada resultam em crescimento descontrolado do tempo residente.

3.3 Técnicas de aferição

As técnicas de aferição podem ser subdivididas em monitoramento, *benchmark* e prototipagem. O monitoramento é a captura de dados do sistema durante sua execução. A ferramenta utilizada para monitoração é denominada monitor, e apresenta as seguintes funções: observar o desempenho do sistema, coletar estatísticas de desempenho, analisar os dados e mostrar resultados. Alguns monitores também identificam problemas e sugerem possíveis soluções (JAIN, 1991).

Benchmarks consistem na aplicação de uma carga de trabalho específica sobre o sistema. A maneira que esse sistema reagi a essa carga caracteriza seu desempenho. Essa técnica permite quantificar a capacidade do sistema em executar determinada operação. Um escalonamento de processos, nesse sentido, gerencia cargas de trabalho em busca do aumento de desempenho.

Para comparar e avaliar políticas de escalonamento, geralmente, criam-se cargas de trabalho sintéticas, aplicando-as sobre essas políticas. É possível comparar o comportamento de diferentes políticas e concluir a que oferece maior desempenho (FEITELSON; RUDOLPH, 1998).

Enquanto um *benchmark* padrão executa aplicações isoladas, um *benchmark* para políticas de escalonamento consiste de um modelo de chegada de processos ao sistema e quantificação da ocupação desses processos. Além da especificação da carga de trabalho, é preciso

definir métricas de desempenho que sejam significativas e que forneçam resultados precisos para comparação de diferentes políticas de escalonamento. Em Feitelson e Rudolph (1998), Chase et al. (1999) são apresentados resultados sobre a utilização de *benchmarks* na avaliação do escalonamento de processos.

Protótipos são implementações simplificadas de um determinado software computacional. O objetivo é implementar apenas as operações essenciais para avaliação, mantendo suas funcionalidades. Essa técnica também pode ser utilizada no desenvolvimento de políticas de escalonamento, mas o custo de implementação desses protótipos, torna-os pouco utilizados (FEITELSON; RUDOLPH, 1998).

3.4 Considerações finais

Este capítulo apresentou as principais técnicas de avaliação de desempenho. Algumas dessas técnicas podem ser utilizadas para avaliar o desempenho do escalonamento de processos. Durante o desenvolvimento de uma política de escalonamento podem ser empregadas técnicas de modelagem (e a solução do modelo por simulação ou técnicas analíticas), para prever o comportamento da política.

A modelagem pode ser usada no aperfeiçoamento e correção de erros de políticas já existentes. Nesse caso, é possível avaliar o comportamento da política de escalonamento nos quais diferentes fatores são modificados, tais como a carga de trabalho, o sistema onde a política é executada, etc.

Pode-se, ainda, desenvolver um protótipo que represente as características essenciais de uma política de escalonamento, obtendo-se dados preliminares de como será o comportamento da política implementada. *Benchmarks* podem ser utilizados para avaliar diferentes aspectos de uma política de escalonamento e, em especial, podem ser usados para comparar diferentes políticas.

Este trabalho utiliza duas das técnicas apresentadas, uma de aferição e outra de modelagem. A técnica de aferição utilizada é o *benchmark*, que é aplicado sobre o sistema, para avaliar capacidades de processamento dos computadores e rede. A técnica de modelagem empregada neste trabalho são as Redes de Fila e o modelo de avaliação proposto é resolvido por meio de simulação, que permite avaliar e obter refinamentos da política DPWP (ARAÚJO et al., 1999).

CAPÍTULO

4

Escalonamento de processos em sistemas distribuídos

4.1 Considerações iniciais

Uma tarefa importante em computação paralela distribuída é a atribuição dos processos que compõem as aplicações sobre os elementos de processamento (EPs). Essa distribuição pode ser realizada através da aplicação do usuário ou por um software de escalonamento. Neste último, pode-se implementá-la integrada à plataforma de portabilidade (PVM ou MPI) ou no sistema operacional. Isso influencia diretamente no desempenho do sistema, tanto em termos de utilização de recursos, como de satisfação dos usuários. Essa atividade é denominada escalonamento de processos (CASAVANT; KUHL, 1988).

O escalonamento em sistemas computacionais distribuídos tem sido vastamente estudado (CORBALAN et al., 2001; OH; HA, 1996; PARK; CHOE, 2002; RANAWEERA; AGRAWAL, 2000; J.Y.COLIN; P.CHRÉTIENNE, 1991; DARBHA; AGRAWAL, 1998; TOPCUOGLU et al., 1999; CHOE; PARK, 2002; RADULESCU et al., 1999). Vários trabalhos têm sido propostos, direcionados para diferentes sistemas computacionais e considerando diferentes objetivos. Esses objetivos estão relacionados a medidas de desempenho, tais como maximizar a utilização dos recursos computacionais e minimizar o tempo de resposta. Grande parte desses trabalhos refletem adaptações dos conceitos já empregados em sistemas uniprocessados. Embora a pesquisa na área de escalonamento esteja consolidada em arquiteturas paralelas (ESHAGHIAN; WU, 1997), na área de sistemas distribuídos a pesquisa encontra-se, ainda, em plena evolução.

Conceitos e técnicas comumente empregados em sistemas baseados em arquiteturas paralelas, muitas vezes possuem adequabilidade reduzida em sistemas computacionais distribuídos de propósito geral. Isso indica que, no contexto de sistemas distribuídos, vários conceitos e

técnicas precisam ser criados e revisados para garantir maior desempenho na implementação do escalonamento.

O escalonamento também se faz necessário em diversas áreas e níveis do processo computacional. Ele adquiriu importância muito grande com os avanços tecnológicos e a ampla utilização dos sistemas computacionais distribuídos. Tais sistemas apresentam alta capacidade de processamento, uma vez que podem constituir o sistema desde computadores pessoais até arquiteturas com inúmeros processadores com elevada potência computacional .

Decisões comuns são tomadas pelo software de escalonamento, tais como quais aplicações terão acesso aos recursos computacionais e qual a quantidade e a localização desses recursos destinados às aplicações paralelas. Essas decisões são afetadas por diferentes fatores, destacando-se a carga de trabalho do sistema, a heterogeneidade dos recursos e as diferentes necessidades dos usuários. Além disso, o software de escalonamento pode tratar de maneira diferente aplicações paralelas com diferentes comportamentos em relação à quantidade de processamento, comunicação e operações de entrada e saída (E/S) (SOUZA, 2000).

Em busca do embasamento necessário sobre o tema escalonamento de processos, este capítulo apresenta os principais conceitos sobre escalonamento de processos em sistemas computacionais distribuídos. O escalonamento visa a diminuição do tempo médio de resposta das aplicações através da melhor utilização dos recursos computacionais disponíveis. As decisões de escalonamento podem considerar diferentes parâmetros do sistema, tais como: a carga de processamento, a arquitetura dos EPs, o tipo de tecnologia da rede de comunicação e o tipo de sistema operacional.

A busca por decisões ótimas, tem direcionado diversas pesquisas na área, as quais utilizam outros parâmetros do sistema (RANAWEERA; AGRAWAL, 2000; CHOE; PARK, 2002; ARAÚJO et al., 1999; MACIIE et al., 2000; RADULESCU et al., 1999). Esse fato motiva o estudo e a avaliação da comunicação entre processos nas decisões de escalonamento, considerando a rede de comunicação como um parâmetro para as tomadas de decisões.

Este capítulo está organizado da seguinte maneira: na seção 4.2 são apresentados os conceitos sobre escalonamento de processos. Uma organização possível dos componentes de uma política de escalonamento é apresentada na seção 4.3. Na seção 4.4 são apresentados alguns exemplos de ambientes de escalonamento, que viabilizam o escalonamento em sistemas distribuídos, juntamente com suas características de implementação e seus principais objetivos e na seção 4.5 é apresentada uma taxonomia para área de escalonamento de processos em sistemas distribuídos.

4.2 Escalonamento de processos

Segundo a taxonomia de (CASAVANT; KUHL, 1988), o escalonamento de processos pode ser classificado de acordo com o número de EPs considerados no sistema. Global, se são considerados vários EPs e local, se o escalonamento é aplicado em um único EP.

O escalonamento local é realizado pelo sistema operacional e refere-se à atribuição de processos concorrentes, em apenas um processador. Sistemas operacionais multitarefa (Solaris, Linux, Windows NT) podem ser exemplos de sistemas que empregam esse tipo de escalonamento (SQUILLANTE et al., 2002).

O escalonamento local utiliza técnicas *time-sharing* para compartilhamento do tempo de uso do processador, no qual o tempo é dividido em pequenas fatias (*time-slice*), que são utilizadas pelo diversos processos submetidos à execução. O escalonamento local não é o enfoque deste trabalho e, portanto, informações mais detalhadas são encontradas em (TANENBAUM, 2001; ZHANG et al., 2000; BOVET et al., 2002).

No escalonamento global, fica caracterizada a presença de mais um EP, o que implica na distribuição dos processos entre esses EPs. O escalonamento pode ser definido como um recurso para o gerenciamento de recursos. Ele pode ser dividido em três componentes básicos (CASAVANT; KUHL, 1988; SOUZA, 2000; SOUZA, 2004):

- Consumidores (processos das aplicações dos usuários e do sistema);
- Recursos (processadores, memória, rede de comunicação, etc);
- Política de escalonamento.

Os consumidores utilizam os recursos e a política é responsável por atribuir os processos aos EPs de acordo com os objetivos e a disponibilidade dos recursos. A atividade de distribuição dos processos é intimamente relacionada a um objetivo, no qual todas as decisões são tomadas. Diminuir o tempo de execução e balancear a carga do sistema podem ser alguns exemplos de objetivos possíveis com o escalonamento.

Neste trabalho, os termos política e algoritmo de escalonamento serão tratados distintamente. A política ou estratégia determina quais, quando e como os mecanismos serão empregados para que o escalonamento seja realizado. Os mecanismos compreendem os comandos ou instruções que desempenham as atividades básicas do escalonamento, instruções que coletam informações sobre a carga do sistema ou instruções que trocam informações entre processadores (SOUZA, 2000).

Um algoritmo de escalonamento é um termo mais genérico, responsável por implementar políticas e mecanismos utilizados no escalonamento de processos. As políticas de escalonamento representam decisões como "permitir que aplicações com menor tempo de execução sejam

executadas prioritariamente"ou "permitir que aplicações interativas tenham maior prioridade na utilização de recursos". Dessa forma, as políticas de escalonamento definem como será o tratamento dado pelo escalonador aos diferentes fatores envolvidos na computação. Isso indica que a definição de políticas de escalonamento eficientes e suas implementações são pontos cruciais para que os objetivos sejam alcançados (SOUZA, 2000).

Um dos objetivos em sistemas computacionais distribuídos é o balanceamento de cargas entre os EPs. No processo de distribuição da carga de trabalho existem dois possíveis mecanismos: compartilhamento de carga (*load sharing*) e o balanceamento de carga (*load balancing*). O primeiro busca evitar estados nos quais EPs encontram-se sobrecarregados enquanto outros estão disponíveis, no segundo, além da carga compartilhada, esta deve se manter homogênea em todo o sistema (SHIVARATRI et al., 1992).

O balanceamento de carga é obtido por meio da distribuição e/ou migração de processos entre os EPs, visando diminuir o tempo de resposta do sistema. Dessa maneira, diminui-se o trabalho de EPs sobrecarregados e utilizam-se ciclos de processamento de EPs ociosos. Além do balanceamento de cargas, outros objetivos podem ser considerados, como por exemplo maximizar a utilização do sistema, melhorar o tempo de resposta experimentado pelos usuários e o tratamento de questões administrativas. O tratamento desses objetivos, torna a atividade de escalonamento ainda mais complexa (HARCHOL-BALTER; DOWNEY, 1996).

4.3 Organização de uma política de escalonamento

Em sistemas distribuídos, as diferenças de carga surgem devido à atribuição demasiada de tarefas para certos EPs e a sub utilização de outros, ou ainda pela influência causada por usuários que submetem arbitrariamente suas aplicações ao sistema. Isso gera a necessidade do balanceamento de carga, um dos objetivos do software de escalonamento para sistemas distribuídos. Esse objetivo pode ser alcançado por meio da utilização de migração de processos. Dessa maneira, Shivaratri et al. (1992) define como é o relacionamento entre os EPs durante a troca de informações sobre a carga do sistema. Shivaratri organiza os componentes de uma política de escalonamento dinâmica em quatro políticas:

- Política de transferência: determina a disponibilidade de um processador de participar de uma transferência de processos, seja como transmissor ou como receptor;
- Política de seleção: definido que um processador será um transmissor, a política de seleção é responsável por decidir qual processo será transferido;
- Política de localização: responsável por encontrar um parceiro para participar da transferência;

- Política de informação: a política de informação é responsável por definir em que momento informações sobre o estado do sistema são necessárias, de que EPs elas devem ser coletadas e que tipo de informação deve ser utilizada.

A carga de trabalho sobre um EP é uma informação valiosa e permite saber o quanto um determinado EP é capaz de receber e processar tarefas. Caso contrário, se o EP encontra-se sobrecarregado necessitando transferir tarefas para outros EPs. Dessa maneira, a utilização de índices de carga possibilita medir a carga de um EP. Esses índices são valores inteiros, não negativos e iniciados geralmente por zero (sistema ocioso) acrescendo positivamente seu valor com o aumento da carga computacional em um determinado EP (FERRARI; ZHOU, 1988; SHIRAZI et al., 1995; SOUZA, 2000).

Alguns exemplos de índices mais utilizados e estudados são: o tamanho da fila de processos, utilização da CPU, quantidade de memória disponível, tempo de resposta. Ainda é possível a combinação desses índices, para melhor representar o estado atual do sistema e obter assim comportamentos mais próximos da realidade (SOUZA, 2000; FERRARI; ZHOU, 1988).

Quanto mais complexo o mecanismo de coleta de um índice, maior a sobrecarga imposta ao sistema. Essa situação pode levar a resultados imprecisos para aplicações com requisitos rígidos em relação à demanda de recursos (em aplicações *CPU-Bound* o importante é saber quanto de CPU está sendo utilizado).

A utilização de índices de cargas absolutos em sistemas distribuídos podem não representar uma boa solução para a distribuição da carga. Isso porque tais sistemas apresentam grande heterogeneidade entre os EPs. Um EP de maior potência computacional pode ser uma opção viável, mesmo que em termos de valores absolutos do índice, ele possua maior carga de trabalho. Portanto, é necessária a normalização dos índices para considerar a potência computacional de todos os EPs envolvidos.

O índice de carga deve estar dirigido ao objetivo da política de escalonamento de processos. Ele é responsável por informar não só a carga atual do sistema, mas também ser capaz de predizer um comportamento futuro (SCHNOR et al., 1996).

4.4 Ambientes de suporte ao escalonamento de processos

Existem duas maneiras de implementação das políticas de escalonamento: uma interna à aplicação e outra externa. Na primeira existe total liberdade de como será realizada a distribuição das tarefas sobre os EPs, na segunda, tanto pode ser no próprio sistema operacional quanto através de um software. Esse software localiza-se entre o sistema operacional e a aplicação, conhecido como software de escalonamento, ou ambiente de escalonamento (KAPLAN; NELSON, 1994; BAKER; BUYYA, 1999).

Vários softwares de escalonamento estão disponíveis, estabelecendo quais aplicações terão acesso aos recursos e quais recursos serão disponibilizados para determinadas aplicações. Alguns deles são construídos para um determinado sistema computacional, como parte integrante do sistema, muito comum em sistemas distribuídos e sistemas computacionais paralelos (TANENBAUM; STEEN, 2002). A vantagem dessa abordagem é a implementação otimizada do escalonador para um determinado sistema computacional. Além disso, a possibilidade de se obter medidas de estado dos recursos devido ao acesso direto.

Quando o escalonador é implementado externamente à aplicação e ao sistema operacional, é possível que o administrador do sistema controle e planeje a utilização dos recursos de maneira estruturada. Há facilidade em se adaptar os sistemas computacionais heterogêneos e a possibilidade de oferecer uma interface independente do sistema operacional. Alguns exemplos de ambientes de escalonamento são apresentados a seguir.

- **LSF** (*Load Sharing Facility*): o LSF é um ambiente de escalonamento que segue o princípio de distribuição de carga entre os EPs. Permite a execução remota e o escalonamento de aplicações paralelas em redes de estações de trabalho heterogêneas. Possibilita a especificação de características da aplicação pelo usuário do sistema. Durante a distribuição de tarefas essas características são consideradas pelo software escalonador e alguns exemplos são: quantidade de memória principal, quantidade de espaço em disco, arquitetura dos EPs. O administrador do sistema pode inserir medidas de desempenho adicionais, a fim de tornar as decisões do escalonador mais flexíveis (Platform Computing Corporation, 2000).

- **Sun Grid Engine**: seu objetivo principal é a utilização ótima dos recursos computacionais em ambientes distribuídos. É implementado por meio de um conjunto de filas e de um escalonador mestre, que recebe as aplicações dos usuários e com a ajuda de processos *daemons*, que executam em cada EP, buscam os EPs apropriados para as aplicações. Implementa migração de processos e *checkpointing*, o que possibilita a recuperação do trabalho em caso de falhas nos EPs. Prioridades podem ser solicitadas e o administrador pode configurar restrições de acesso aos usuários do ambiente. Nesse ambiente, as políticas, assim como as filas e os períodos de verificação de carga dos EPs, podem ser configurados em tempo de execução, sem necessidade de interrupção na execução do ambiente (SUN, 2003).

- **Condor**: implementa o escalonamento de aplicações em sistemas fracamente acoplados e utiliza o conceito de que alguns EPs da rede desperdiçam tempo de processamento que poderia ser aproveitado por aplicações paralelas. Possui um gerente centralizado, que é responsável pelo escalonamento e atribuição de tarefas aos EPs. O usuário no Condor pode especificar a arquitetura do EP, a quantidade de memória, a quantidade de espaço em disco disponibilizado para troca (*swap*) e o sistema operacional. Implementa prioridades por meio do algoritmo *up-down*, que incrementa a prioridade de aplicações que esperam

tempo maior por recursos e reduz a prioridade de aplicações que utilizaram vários recursos em instantes de tempos passados (Condor Team, 2000).

- **DQS** (*Distributed Queueing System*): possui um conjunto de filas pelas quais os usuários submetem suas aplicações. Para realizar o escalonamento e a atribuição de tarefas ele se baseia na carga média observada do sistema; assim, os EPs que apresentam menor carga são os que recebem a aplicação. As políticas de escalonamento podem ser configuradas em tempo de execução através de uma interface gráfica ou por linha de comando. O usuário pode especificar quais são os requisitos de sua aplicação, que podem ser a quantidade de memória, arquitetura dos EPs, etc (BELCASTRO et al., 1988). O DQS também possui um escalonador auxiliar, que assume as atividades do escalonador mestre em caso de falhas.

- **Load Leveler**: visa melhorar o desempenho do sistema computacional por meio da utilização de EPs ociosos e reduzindo a carga em EPs sobrecarregados. É implementado em uma fila de prioridades, que recebe as aplicações submetidas ao sistema e à medida que os recursos tornam-se disponíveis as tarefas são atribuídas ao EPs. O usuário pode especificar os requisitos de sua aplicação, tais como a arquitetura, a quantidade de memória e a quantidade de espaço em disco para troca. (BAKER; BUYYA, 1999).

- **NQE** (*Network Queuing Environment*): é formado por um processo mestre, que é responsável pelo escalonamento das aplicações, e de processos *daemon*, que coletam informações sobre a carga nos EPs para serem submetidas ao processo mestre. O usuário fornece as características das aplicações, que são inseridas em filas específicas de acordo com essas características. Recursos importantes para o escalonador podem ser configurados pelo administrador, assim como as políticas de escalonamento e o número de filas do ambiente podem ser alterados em tempo de execução (Silicon Graphics, Inc.; Cray Research, Inc., 1998).

- **PBS** (*Portable Batch System*): o objetivo do PBS é fornecer um controle sobre a execução e o escalonamento de aplicações, tanto em uma rede de estações de trabalho, quanto em máquinas com processamento maciçamente paralelo. Os usuários fornecem informações sobre as aplicações e as políticas pelas quais é realizado o escalonamento (SYSTEMS, 2004).

- **AMIGO** (*dynAMical flexIble schedulinG envirOnmet*): esse software foi desenvolvido com o objetivo de fornecer uma maneira flexível e dinâmica de agrupar políticas de escalonamento e relacioná-las com as aplicações dos usuários em ambientes computacionais heterogêneos e distribuídos executando o sistema UNIX (SOUZA, 2000). Sua implementação baseia-se em um processo mestre e vários processos *daemons*, responsáveis por obter informações de carga dos EPs. Ele foi projetado com uma estrutura modular de duas camadas, facilitando sua organização e permitindo uma maneira mais fácil de adaptá-lo para diferentes sistemas computacionais.

Tabela 4.1: Relação entre os ambientes de escalonamento

	Engine	Condor	DQS	L.leveler	LSf	NQE	PBS	AMIGO
heterogeneidade	sim	sim	sim	sim	sim	sim	sim	sim
mensagens	sim	não	não	não	sim	s/n	não	sim
pol. dinâmicas	sim	não	sim	sim	sim	sim	sim	sim
s/ recomp.	s/n	não	s/n	s/n	sim	sim	sim	sim
checkpointing	sim	sim	sim	sim	sim	não	não	não
migração	sim	sim	não	sim	sim	não	não	não
balanceamento	sim	não	sim	sim	sim	sim	sim	sim
gratuito	sim	sim	sim	não	não	não	sim	sim

A camada superior do AMIGO é formada pela interface gráfica e arquivos de configuração, que oferecem o cadastro de classes de softwares, políticas, medidas de desempenho e *benchmarks*. Por meio dessa camada, o administrador do sistema pode relacionar as classes de softwares com as políticas mais adequadas, permitindo um escalonamento dirigido às classes de aplicações.

Na camada inferior, é definida toda a interação entre o AMIGO, as políticas de escalonamento cadastradas e as aplicações, bem como onde é realizado o escalonamento e a atribuição de tarefas baseada nas informações contidas nos arquivos de configuração. Essa interação é realizada por meio de um processo *daemon* (amigod), que deve executar em todos os EPs do sistema distribuído, e das bibliotecas de passagem de mensagens (PVM, MPI e TAO-CORBA).

As aplicações paralelas não precisam ser modificadas ou adaptadas e o escalonamento é realizado de forma transparente, ocultando detalhes de implementação e políticas empregadas. O AMIGO não possui mecanismos de *checkpointing*, que implementam tolerância à falhas, e migração de processos (SOUZA, 2000).

A tabela 4.1 relaciona as características e diferenças entre os softwares de escalonamento quanto: a heterogeneidade, mensagens (permite a utilização de ambientes de passagens de mensagens tais como PVM ou MPI), políticas dinâmicas (possibilita a reconfiguração do ambiente em tempo de execução), sem recompilação (faz o escalonamento sem a necessidade de recompilação do código fonte da aplicação), *checkpointing*, migração de processos, balanceamento de carga e domínio público.

4.5 Taxonomia para a área de escalonamento de processos

A taxonomia de Casavant e Kuhl (1988) propõe duas classificações: a hierárquica, na qual os termos se relacionam entre si; e a plana, na qual não existe nenhuma relação hierárquica entre os termos. Tais classificações são apresentadas nas seções seguintes.

4.5.1 Classificação hierárquica

A classificação hierárquica subdivide o escalonamento de processos em dois tipos: local (apenas um processador) e global (vários processadores). O escalonamento global é dividido em estático e dinâmico. No escalonamento estático, as alocações de recursos são definidas antes da execução das aplicações e explicitamente no código fonte ou durante a compilação do programa. O escalonamento dinâmico define a alocação de recursos durante a execução das aplicações e pode ser alterado a qualquer momento de sua execução.

As políticas de escalonamento estáticas são definidas na fase de desenvolvimento das aplicações. Elas requerem informações a respeito do comportamento dos processos, tais como tempo de execução, localização dos dados a serem manipulados, etc. Esse tipo de escalonamento é de implementação complexa e restrita. Caso a estimativa por recursos computacionais não corresponda à realidade, fatalmente, degradar-se-á o desempenho da aplicação.

O escalonamento dinâmico possibilita alocações de recursos durante a execução dos processos. É um tipo de escalonamento flexível, que utiliza informações sobre o estado do sistema (índices de carga) para auxiliar nas decisões de escalonamento. Para tomar essas decisões, tais políticas realizam coletas de informações sobre o comportamento da carga do sistema. Tais decisões podem priorizar a execução de um processo, a ocupação de determinado recurso ou outro aspecto.

É possível modificar a distribuição dos processos, por meio de migrações, permitindo melhor adequação caso o sistema tenha mudanças bruscas de carga. Uma desvantagem dessa abordagem é a sobrecarga imposta ao sistema pela própria política de escalonamento, durante a obtenção e análise dinâmica das informações. Tais informações, são utilizadas nas tomadas de decisões para distribuição dos processos e para possíveis migrações.

Políticas de escalonamento estáticas ainda podem ser subdivididas em duas outras. Entretanto, dado que esta dissertação não está relacionada a esse tipo de política, não serão abordadas suas subdivisões. As políticas de escalonamento dinâmicas são subdivididas em dois tipos: fisicamente distribuídas e não distribuídas. Essa subdivisão, diz respeito à responsabilidade pelo escalonamento, que pode ser centralizada em um único processador (fisicamente não distribuída), ou dividida entre vários processadores (fisicamente distribuída).

As políticas de escalonamento dinâmicas, fisicamente distribuídas, são subdivididas em cooperativas e não cooperativas. As cooperativas tomam decisões levando em consideração informações capturadas dos módulos em cada computador ou processador do ambiente. Tais módulos trabalham em conjunto para obter uma visão global do sistema. Nas políticas não cooperativas, as decisões são tomadas sem considerar informações externas. Nesse caso, cada processador decide baseado apenas em informações locais, no qual os módulos concorrem entre si para obter maior desempenho de suas próprias aplicações. A figura 4.1 apresenta a classificação hierárquica de Casavant e Kühl.

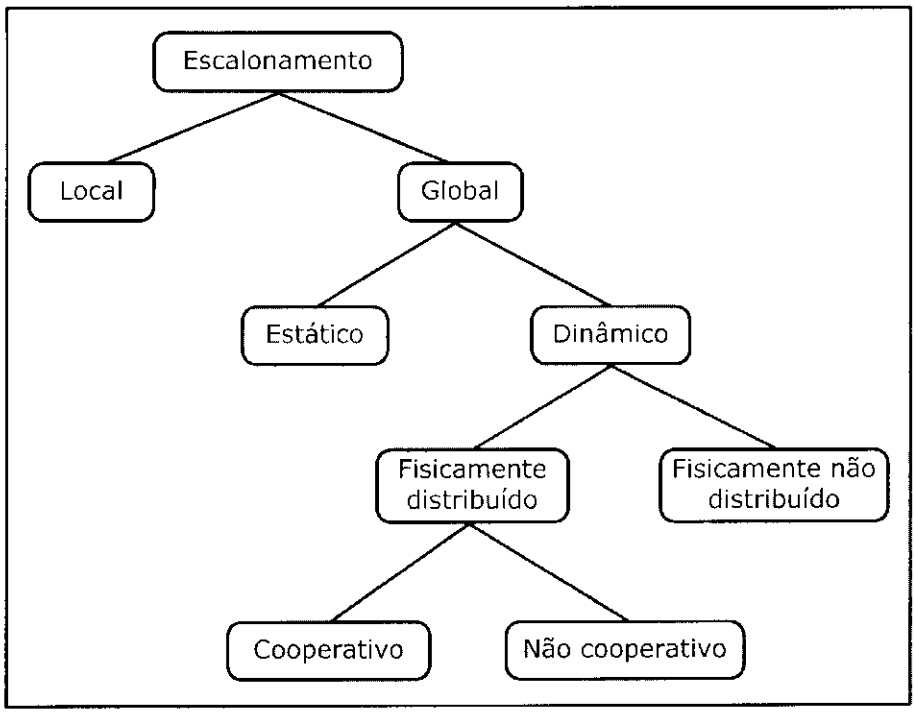

Figura 4.1: Classificação hierárquica de Casavant e Kuhl (1988)

4.5.2 Classificação plana

Alguns termos, que não têm relação hierárquica entre si, mas podem ser utilizados para classificação das políticas de escalonamento. Tais termos são definidos por Casavant e Kuhl (1988) na seguinte classificação, denominada plana:

- Política adaptativa - uma política de escalonamento é adaptativa quando ajusta seu próprio comportamento de acordo com a situação atual do sistema. Essas políticas podem definir, dinamicamente, regras e parâmetros em busca de maior desempenho.

- Atribuição inicial - dada uma decisão de escalonamento, uma política que utiliza atribuição inicial, não modifica a distribuição até o final da execução dos processos. Esses processos não realizam migração (ou transferência).

- Reatribuição dinâmica - nesse tipo de política, os processos realizam preempção, para interromper a execução, e a migração, para redistribuir os processos entre processadores ou computadores do sistema.

- Compartilhamento de carga - compartilhar carga significa evitar estados nos quais processadores ou computadores do sistema estejam sobrecarregados, enquanto outros, ociosos.

- Balanceamento de carga - balancear carga significa manter as cargas distribuídas, eqüitativamente, sobre todos os recursos do sistema. Essa técnica utiliza a reatribuição dinâmica para manter a carga homogênea.

Apesar da classificação de Casavant e Kuhl (1988) ser bastante completa, ela não inclui alguns termos, comumente encontrados em trabalhos sobre escalonamento de processos. Alguns desses termos são: compartilhamento de espaço (*space sharing*), compartilhamento de tempo (*time sharing*) e escalonamento cooperativo (*gang scheduling*).

Políticas dinâmicas, que utilizam o compartilhamento de espaço, são simples de serem implementadas, porém, sua utilização isolada pode ocasionar em execuções com baixo desempenho. Esse fato é justificado pela falta de multiprogramação no instante em que as tarefas ficam bloqueadas realizando entrada e saída. Essa abordagem realiza a distribuição de tarefas por meio do mapeamento um para um, aplicável em sistemas homogêneos.

Em sistemas heterogêneos, esse mapeamento não se aplica, no qual a capacidade de processamento varia entre os diferentes elementos de processamento (EPs), tornando-se uma abordagem pouco utilizada. Portanto, isso indica que mais de uma tarefa pode ser atribuída por EP, reduzindo a gama de utilização das políticas que seguem estratégias *space sharing* em sistemas distribuídos (SENGER et al., 2002).

Estratégias de compartilhamento de tempo agregam grande número de trocas de contexto causando baixo desempenho devido à necessidade de sincronização, freqüentemente encontrada em aplicações paralelas. Essa estratégia é, geralmente, utilizada em arquiteturas com multiprocessadores simétricos (SMP) e não tem mostrado bons resultados quando o número de processos excede o número de processadores (CORBALAN et al., 2001). Entretanto, para cargas de trabalho de propósito geral (em sistemas distribuídos, por exemplo), essas estratégias são atrativas, pois fornecem baixos tempos médios de resposta para processos interativos e elevado *throughput* para processos *I/O-Bound* (CORBALAN et al., 2001).

Políticas de escalonamento, que seguem a estratégia de compartilhamento de tempo, são utilizadas em sistemas computacionais de alto desempenho e de propósito geral (ZHOU et al., 2000). Processos de uma mesma aplicação paralela necessitam de sincronização durante a sua execução. O escalonamento sincronizado de aplicações sobre os EPs, é um fator crítico para execuções eficientes em sistemas computacionais de tempo compartilhado.

Uma estratégia para sincronização de processos é a utilização de abordagens como *explicit coscheduling*, ou *gang scheduling* (escalonamento cooperativo). Nessa abordagem, o tempo é dividido em intervalos, denominados *scheduling slot* ou *time slot*. Todas as aplicações paralelas executam, simultaneamente, por um certo *time slot* e são controladas por um escalonador global.

Cada aplicação utiliza o EP no intervalo de tempo definido pelo escalonador, por exemplo, processos que interagem freqüentemente devem ser escalonados ao mesmo tempo. Quando um *time slot* termina, os processadores realizam troca de contexto ao mesmo tempo que ficam disponíveis para processos de outras aplicações. Caso haja espaço, aplicações podem ser alocadas no mesmo *time slot* e executadas, simultaneamente, em diferentes subconjuntos de EPs. Dessa maneira, o *gang scheduling* pode ser considerado como uma estratégia de escalonamento

que combina compartilhamento de tempo e compartilhamento de espaço (ZHOU; BRENT, 2001).

4.6 Considerações finais

Este capítulo apresentou o escalonamento de processos, no qual definem-se as regras de distribuição dos processos sobre os EPs. Tais regras auxiliam no aumento de desempenho e na utilização eficiente dos recursos do sistema.

O escalonamento de processos em sistemas computacionais distribuídos é realizado de acordo com uma política de escalonamento. A política tem como objetivo alcançar algumas metas, tais como o balanceamento de carga, a maximização da utilização de algum recurso (CPU, memória, disco), a minimização do tempo de resposta do sistema. A escolha de uma política de escalonamento é um fator de grande influência sobre o desempenho final do sistema. Dessa maneira, Feitelson e Rudolph (1998), consideram que os diferentes requisitos das aplicações podem ser tratados por diferentes mecanismos. Dessa maneira, aplicações *CPU-Bound* podem receber um tratamento diferenciado de aplicações *I/O-Bound*.

Os processos das aplicações podem ser caracterizados de acordo com o seu comportamento em relação à utilização de recursos em classes, tais como orientadas a CPU (*CPU-Bound*) ou orientadas às operações de entrada e saída (*I/O-Bound*). Isso depende da atividade que melhor representa seu padrão de comportamento durante a execução (SILVA; SCHERSON, 2000).

Os principais problemas decorrentes do escalonamento estão relacionados à capacidade do escalonador em lidar com diferentes fatores, tais como as características comportamentais das aplicações, a carga de trabalho e a heterogeneidade dos recursos computacionais. Esses fatores e a existência de diferentes objetivos tornam difícil o desenvolvimento de uma política de escalonamento geral que trate todo tipo de aplicação.

Alguns autores, Anastasiadis e Sevcik (1997), Corbalan et al. (2001), Souza (2000) demonstram através de experimentação prática e simulação, que políticas orientadas ao comportamento das aplicações têm maior desempenho em relação às políticas genéricas, que tomam suas decisões sem considerar a classe da aplicação. Tais resultados evidenciam que a política deve adaptar-se à classe da aplicação a ser escalonada, ou ainda, especializar-se para apenas uma determinada classe.

Tal evidência também é verificada em nível de sistema operacional. O sistema operacional Linux emprega três políticas de escalonamento diferentes, de acordo com a classe da aplicação a ser escalonada (uma política para processos interativos e duas políticas para processos da classe *real time*) (BOVET et al., 2002).

De modo semelhante, em um nível mais próximo da aplicação paralela, softwares específicos de escalonamento tais como o LSF (*Load Sharing Facility*) (seção 4.4) e o AMIGO (seção 4.4), permitem que várias políticas de escalonamento possam ser agregadas e empregadas, de

acordo com o tipo de aplicação a ser escalonada.

Grande parte dos processos de aplicações paralelas interagem freqüentemente entre si ampliando o uso da rede. Isso motiva a definição de políticas eficientes e direcionadas às aplicações *Network-Bound*, cumprindo um papel importante no desempenho de sistemas distribuídos. Nesse contexto, no capítulo 5 são apresentadas algumas políticas de escalonamento apontando seus principais pontos fortes e fracos, bem como alguns trabalhos que consideram atributos do subsistema de comunicação.

CAPÍTULO

5

Políticas de escalonamento de processos

5.1 Considerações iniciais

Este capítulo apresenta uma revisão de diversos trabalhos da área de escalonamento de processos. São apresentados trabalhos que definem políticas de escalonamento, técnicas para redução do custo de comunicação entre processos e trabalhos que visam caracterizar e quantificar o tráfego de comunicação causado pelas aplicações paralelas.

5.2 Políticas de escalonamento

O escalonamento é responsável por distribuir processos sobre elementos de processamento (EPs) visando o aumento no desempenho de aplicações. A definição de boas políticas de escalonamento melhoram, consideravelmente, o desempenho final em relação ao tempo médio de resposta das aplicações, como pode ser observado em (RANAWEERA; AGRAWAL, 2000; CHOE; PARK, 2002; RADULESCU et al., 1999; ARAÚJO et al., 1999; MACHE et al., 2000).

As próximas seções apresentam cada um desses trabalhos destacando suas principais características e apontando suas limitações.

5.2.1 Política de escalonamento para sistemas heterogêneos baseada na duplicação de processos

Em (RANAWEERA; AGRAWAL, 2000) é apresentada a política TDS (*Task Duplication Scheduling*), que busca diminuir o tamanho da fila do escalonador minimizando o tempo total de execução das aplicações. Esse trabalho utiliza as métricas *makespan* (tempo total de execução da

aplicação) e tempo de escalonamento (tempo gasto na geração de uma distribuição de processos) para avaliar o desempenho da política.

A entrada da política consiste de um grafo acíclico dirigido (GAD). Esse grafo é definido pela tupla (V, E, P, τ, c), onde: V é o conjunto de processos, E é o conjunto de arestas de comunicação, P é o conjunto de processadores, $\tau = \tau(j, p)$ onde $j \in V, p \in P$ e $\tau(j, p)$ indica o tempo de execução do $j^{\text{ésimo}}$ processo no $P^{\text{ésimo}}$ processador. $c = c(j, k)$ onde $j, k \in V$ e $c(j, k)$ indica o custo de comunicação entre j e k. Assume-se que o custo de comunicação é igual em todos os processadores.

A política busca a obtenção de poucas informações dos processos. Tais informações correspondem aos atributos:

- *est* - menor tempo de início
- *ect* - menor tempo de execução
- *last* - máximo tempo de início permitido
- *lact* - máximo tempo de execução permitido
- *fpred* - o predecessor de um processo
- *fproc* - o processador no qual um processo é executado em menor tempo

A política segue quatro passos. No primeiro, o GAD é percorrido de cima para baixo, calculando-se *est*, *ect*, *fpred*, *fproc* e *level* de cada processo. O *level* é definido como o maior valor da soma dos custos computacionais sobre os diferentes caminhos entre dois processadores no grafo. Os processos no vetor *queue* são classificados em ordem ascendente de *level*. No segundo passo calcula-se *last*, *lact* de cada processo percorrendo o grafo de baixo para cima. O terceiro passo gera um conjunto de clusters de processos, utilizando o menor número possível de processadores. No quarto passo são realizadas as duplicações de processos e o repasse de mensagens.

O desempenho da TDS é comparado ao de outra política de escalonamento chamada BIL (*Best Imaginary Level Scheduling*). Os resultados apresentados mostram que a TDS melhora o desempenho em 20% e, ainda, obtém-se um algoritmo de complexidade de tempo menor.

5.2.2 Política de escalonamento com condições ótimas baseada na duplicação de processos em sistemas heterogêneos

Choe e Park (2002) estendem a TDS identificando uma de suas limitações. Definem-se condições de otimização para determinar se o resultado do escalonamento é ótimo ou não,

segundo os objetivos pelos quais a política é projetada. A idéia principal é realizar duplicação de processos antes que esses sejam alocados nos elementos de processamento.

Dada uma aplicação, que é expressa como um GAD (grafo acíclico dirigido) onde, $G = (V, E, \tau, c)$, o problema consiste na atribuição de todos os processos sobre os elementos de processamento. Um dos objetivos é realizar uma atribuição na qual os processos são distribuídos minimizando-se o tempo de escalonamento e o *makespan*.

Esse trabalho busca a definição de uma política de escalonamento com condições ótimas; tais condições são definidas como: se um dado GAD satisfaz as *condições ótimas* de uma política de escalonamento, essa política distribui tal GAD com o menor tempo de escalonamento possível. Tal política deve garantir o menor tempo e não são necessários refinamentos adicionais, conforme aumenta-se o tamanho da entrada dos GADs.

A heterogeneidade do sistema é modelada da seguinte maneira: existem p processadores denominados $P_1, P_2, ...P_p$, e o tempo de execução necessário para processar uma única tarefa no processador P_i é s_i. As capacidades de processamento s_i são definidas como $s = s_1, s_2, ..., s_p$. Assume-se que $s_i \leq s_{i+1}$ para $1 \leq i \leq p$. Em geral, p, é menor que o número de processos submetidos à execução. O tempo de execução do processo n_i no processador P_j é $\tau_i \cdot s_j$. Dessa maneira, o escalonamento de processos em sistemas heterogêneos é tratado similarmente aos casos em sistemas homogêneos.

Melhora-se a TDS na medida em que as fases de duplicação de processos e de alocação são integradas numa única fase. Entretanto, a comparação de desempenho realizada nesse trabalho, entre TDS e TDS estendida, é apenas conceitual. Não são apresentados resultados práticos com ambas as políticas, mas apenas exemplos de execuções.

5.2.3 LLB: Uma política de escalonamento para sistemas com memória distribuída

Em (RADULESCU et al., 1999) é apresentada a LLB (*List-based Load Balacing*), uma política de escalonamento empregada em sistemas computacionais com memória distribuída. Essa política utiliza conceitos distintos entre mapeamento e escalonamento de processos. Um processo é mapeado, se seu EP destino está apto a recebê-lo. Um processo é escalonado se está mapeado e seus tempos de execução são conhecidos.

Um processo pronto caracteriza-se por não ter sido escalonado, mas todos os seus predecessores sim. O parâmetro *urgência* de um processo é definido como uma prioridade estática, armazenado em uma lista linear. A urgência de um processo é a soma dos tempos totais de computação e comunicação.

São utilizadas técnicas de mapeamento de grupos e ordenação de processos combinadas em uma única fase. Apenas processos prontos para execução são considerados na fase de

mapeamento. Quando um processo que pertence a um grupo é mapeado, todos os demais também são, para o mesmo elemento de processamento, reduzindo o custo de comunicação.

Em cada passo do algoritmo um processo é escalonado. No primeiro passo, o EP destino é selecionado como aquele que está há mais tempo ocioso. Cada EP tem uma lista de processos prontos já mapeados. Os processos prontos, mas ainda não mapeados, são armazenados em uma lista global. Inicialmente, as listas de processos prontos mapeados estão vazias e a lista global contém os processos de entrada. Todas as listas com processos prontos são classificadas em ordem decrescente do parâmetro urgência. Após selecionado o EP destino, o próximo passo é escolher qual processo será escalonado nele. Os candidatos são: o processo pronto mapeado mais urgente e o processo pronto não mapeado mais urgente. Se dois processos iniciam ao mesmo tempo, é selecionado o processo mapeado.

Finalmente, a lista de processos prontos é atualizada. O escalonamento dos processos pode fazer com que outros processos tornem-se prontos. Tais processos podem ser mapeados ou não. Os processos mapeados são adicionados na lista de processos prontos correspondente em seus respectivos EPs. Os processos não mapeados são adicionados na lista global de processos prontos.

O desempenho da política é comparado ao de três outras, LCA (*List Cluster Assignment*) (SARKAR, 1989), WCM (*Wrap Cluster Merging*) (YANG, 1993) e GLB (*Guided Load Balacing*) (RADULESCU; Van Gemund, 1998). Assim como na política TDS, o LLB utiliza uma função distribuição de probabilidades uniforme (função normal) para geração dos atrasos de comunicação.

As métricas para avaliação de desempenho das políticas são o *makespan* e o custo para se gerar um escalonamento. A política melhorou em 42% o tempo de resposta de uma aplicação em relação às estratégias de alocação de processos com baixa complexidade. Quando comparada às políticas de mais alta complexidade, como a política LCA, o desempenho da LLB é equivalente; quando são considerados processos com granulosidade grossa, o LLB é 16% melhor.

5.2.4 DPWP: uma política de escalonamento dinâmica para computação paralela virtual

Em (ARAÚJO et al., 1999) é apresentada a DPWP (*Dynamic Policy Without Preemption*), uma política com propósitos de balanceamento de carga para aplicações *CPU-Bound*. Exemplos de aplicações *CPU-Bound* são aquelas que realizam muitos cálculos matemáticos e acabam sobrecarregando a CPU de um determinado elemento de processamento.

As características principais da DPWP são: é uma política dinâmica, sem preempção e pode ser aplicada em ambientes heterogêneos. A escolha por uma política dinâmica é justificada pela sua capacidade de adequação às mudanças sofridas pelo sistema computacional, durante a execução das aplicações. Dessa maneira, observa-se a utilização de políticas dinâmicas em sis-

temas computacionais multiusuários, tais como sistemas computacionais distribuídos, uma vez que esse tipo de sistema caracteriza-se pela diversificação de aplicações e usuários concorrendo pelos recursos computacionais (CPU, rede de comunicação, discos, etc.).

Na DPWP, não é incluída nenhuma técnica para migração de processos (ou política com preempção). Assume-se que migrar um processo consiste em transferi-lo de um EP para outro e essa atividade pode acarretar diminuição no desempenho, pois, para que o processo possa ser restabelecido em outro EP é necessário que seu estado seja transferido.

A heterogeneidade é tratada pela DPWP por meio da normalização de todos os elementos de processamento do sistema em relação ao mais potente. Antes de iniciar a política, algumas configurações são necessárias, entre elas a execução de um *benchmark* sobre todos os elementos de processamento. Obtido os valores do *benchmark* em cada EP é realizada a normalização em ordem decrescente da potência computacional.

A DPWP pode ser dividida em duas fases: ordenação dos computadores segundo uma razão entre a capacidade de processamento e carga de processos e alocação dos processos respeitando a ordem estabelecida na primeira fase (detalhes da DPWP são apresentados no capítulo 6).

5.3 Subsistema de comunicação como parâmetro para decisões de escalonamento

Os estudos anteriormente apresentados, relacionados ao desenvolvimento de políticas de escalonamento, consideram uma série de parâmetros, contudo, desprezam as características de comunicação entre processos da mesma aplicação paralela. Essas características compreendem: a quantidade de mensagens trocadas entre processos; a distribuição dessas mensagens durante a execução das aplicações; e a queda de desempenho (*slowdown*) dos processos ao utilizarem o meio de comunicação.

Buscando resolver essas limitações de caracterização da comunicação, surgiram outros trabalhos tais como (MAO et al., 1999; NI et al., 1985; RYOU; WONG, 1989; KEREN; BARAK, 2003; CHODNEKAR et al., 1997), os quais apresentam análises do impacto da transmissão e recebimento de informações entre processos de aplicações paralelas. A partir dessas análises é possível definir estratégias de escalonamento que levem em consideração, tanto os requisitos das aplicações, quanto dos recursos computacionais envolvidos na comunicação entre processos. Esses trabalhos são apresentados nas próximas seções.

5.3.1 Política *On-line* para escalonamento de processos

Em (MAO et al., 1999) é apresentada uma política *on-line* que considera os custos de processamento e comunicação entre os processos. O escalonamento de processos é comparado ao problema do empacotamento bidimensional (AZAR; EPSTEIN, 1997): dada uma caixa com largura fixa e altura indeterminada, e também um conjunto de retângulos com largura e altura definidos, o objetivo é alocar os retângulos na caixa minimizando o tamanho do espaço ocupado.

Os retângulos são comparados com os processos a serem escalonados nos elementos de processamento de um sistema computacional paralelo. É proposto um modelo matemático que considera os custos de comunicação e processamento.

Para formalizar tal modelo, a equação (5.1) é definida. Se k_j processadores são escolhidos para executar uma aplicação paralela, c o custo de comunicação (ou custos de *overhead* tais como: acesso a memória compartilhada, sincronização entre os processos, etc) e $J_{1,2,\ldots,j}$ cada processo com tempo de execução p_j, então, o tempo total de execução, TJ, dessa aplicação é obtido aplicando-se a equação (5.1).

$$TJ = \sum_{j=1}^{n} \frac{p_j}{k_j} - ck_j \tag{5.1}$$

Os resultados desse trabalho são obtidos por meio de simulação. Avaliam-se esses resultados interpretando, basicamente, dois argumentos: p_j, que é o tempo gasto com processamento e c, o tempo gasto em comunicação. Os valores para p_j são obtidos através de uma função de distribuição de probabilidades uniforme com limites que variam de 20 a 200. Para c, assumem-se os valores 1, 5, 10 e 20 e a condição $c < p_j$.

Algumas limitações podem ser percebidas nesse modelo, tais como: a não utilização de uma função de probabilidade que represente de forma mais precisa a carga dos processos (intervalo $20 - 200$); um estudo mais elaborado para determinar valores para o custo de comunicação (explicação do porque c receber os valores 1, 5, 10 e 20) e uma comparação entre a estratégia proposta com outros algoritmos de escalonamento para o mesmo propósito.

5.3.2 Estratégias de alocação de recursos que consideram fatores de comunicação

Em (NI et al., 1985), é proposta uma técnica para escalonamento de processos em sistemas multiprocessados ponto-a-ponto, com baixo grau de crescimento em escala. Um elemento de processamento no estado ocioso, ou com pouca carga, envia requisições aos seus vizinhos para recebimento de processos. Esses vizinhos respondem à requisição informando seus estados por meio de dois tipos de mensagens: ocupado ou não ocupado. Dessa maneira, o elemento de processamento ocioso seleciona um vizinho e envia uma mensagem de entrega. O vizinho pode

responder de duas maneiras: enviando um novo processo ou uma mensagem de atraso, caso o novo processo tenha sido entregue a outro elemento de processamento. Com a utilização dessa técnica, busca-se diminuir o número de informações transmitidas pelo canal de comunicação.

Ryou e Wong (1989) apresentam um modelo de agrupamento de processos. À partir desses grupos, são definidas técnicas para tratamento de envio e recebimento de mensagens visando diminuição da sobrecarga de comunicação e tempo de resposta das aplicações. Uma estratégia denominada *set* procura reduzir o número de mensagens trocadas na rede de comunicação e o *overhead* da CPU em cada elemento de processamento.

Os trabalhos (NI et al., 1985; RYOU; WONG, 1989), anteriormente apresentados, não avaliam o estado de ocupação da rede de comunicação. O meio físico não é quantificado por quaisquer parâmetros e não é realizada nenhuma análise do impacto da comunicação sobre o escalonamento de processos.

5.3.3 Política de custo oportuno para redução do *overhead* de comunicação em clusters computacionais

Keren e Barak (2003) propõem uma política de custo oportuno, que considera aplicações que causam *overhead* de comunicação entre processos. Nesse trabalho é alterada uma função custo marginal (ASPNES et al., 1997) para permitir realocação dos processos entre os elementos de processamento. Utilizam-se conceitos de roteamento de circuitos virtuais para determinar o *overhead* da CPU em termos de comunicação. Cada elemento de processamento do sistema tem múltiplos recursos: CPU, memória, dispositivos de entrada/saída, etc. O objetivo é minimizar a sobrecarga de utilização desses recursos.

Considera-se um modelo computacional no qual cada processo é caracterizado por sua carga computacional, denominada *CPU weight*, e por um vetor que contém os *overheads* de comunicação das CPUs. Nesse vetor, cada elemento corresponde ao *overhead* de comunicação com outro processo. A carga total de um EP é definida como a soma dos *CPU weights* e dos *overheads* de comunicação de todos os processos.

O problema da atribuição *on-line* de processos que realizam operações na rede de comunicação aos EPs, é definido como: dado um cluster M com n EPs e uma seqüência independente de processos que chegam ao sistema em intervalos que seguem uma função de distribuição de probabilidades uniforme, o EP i é definido em termos de sua velocidade de CPU $v(i)$. Um processo j apresenta peso de CPU $w(j)$ e um vetor de *overheads* de comunicação $\vec{o}(j) = (o_1(j), o_2(j), ..., o_n(j))$, onde $o_i(j)$ é o *overhead* de comunicação da CPU entre o processo j e o EP i. Quando um processo chega, ele é atribuído, *on-line*, para o mesmo EP i. Isso aumenta a carga total de CPU do EP k, $k \in M$, por $p_k^i(j)$, definido como:

$$p_k^i(j) = \begin{cases} w(j)/v(i) + \sum_{i' \neq i} o_{i'}(j)/v(i) & \text{se } i = k, \\ o_k(j)/v(k) & \text{caso contrário.} \end{cases} \quad (5.2)$$

Da mesma maneira, quando um processo termina, a carga de todos os EPs diminui. O objetivo da política é minimizar o carga de CPU para todos os EPs. Quando um processo é atribuído ao EP, aumenta-se sua carga de CPU em um conjunto de outros EPs. O método utilizado pela política é escalonar um processo que chegou em outro EP minimizando a soma dos custos marginais de todos os EP participantes do escalonamento.

Formalmente, assumem-se que os pesos (*CPU weights*) são coletados, nos quais a carga máxima de CPU de um algoritmo ótimo *off-line*, *OPT*, é 1. Define-se $a = 1 + \gamma/2$ para a mesma constante $\gamma in]0, 1[$. Então, o custo marginal de atribuição do processo j no EP i é:

$$H_i(j) = \sum_{k \in M} a^{l_k(j)+p_k^i(j)} - a^{l_k(j)} \quad (5.3)$$

O modelo proposto para avaliação do *overhead* em (KEREN; BARAK, 2003) não considera o impacto que cada processo causa em operações na rede de comunicação. Além disso, o modelo não avalia o peso desse *overhead* numa medida que quantifique o atraso de CPU, tal como a quantidade de MIPS (milhões de instruções por segundo) consumidos pela CPU para realizar operações de envio e recebimento de mensagens através da rede.

5.3.4 Uma metodologia para caracterização de tráfego gerado por aplicações paralelas

Chodnekar et al. (1997) definem uma metodologia para caracterização da comunicação de aplicações paralelas. São considerados três atributos para captura do tráfego: freqüência ou taxa de geração de mensagens, distribuição espacial das mensagens e tamanho das mensagens.

O atributo taxa de geração de mensagens representa um componente temporal do subsistema de comunicação. A distribuição espacial representa o padrão de tráfego, ou seja, com qual processo, outro relacionado, deseja se comunicar.

Essa metodologia é utilizada para caracterização do comportamento das aplicações sobre um sistema com uma topologia de rede 2-D chaveada e roteamento determinístico. Esse comportamento é obtido avaliando o desempenho das aplicações simulando tal topologia, coletando (por meio de logs) os eventos de rede na simulação e analisando tais eventos por meio de modelos de regressão e técnicas estatísticas, para quantificar os atributos de tráfego.

Os resultados obtidos das análises dos logs das aplicações mostram que o tempo de chegada, na maioria das aplicações, segue a variação de uma função de distribuição exponencial para todos os EP do sistema. Assim, utilizando-se tais informações de distribuição, a taxa de geração de mensagens de uma aplicação pode ser modelada.

Um outro resultado desse trabalho mostra que a distribuição espacial das mensagens de uma aplicação paralela pode ser representada matematicamente. Por meio de análises do comportamento e captura de padrões de acesso das aplicações, tal atributo pode ser representado como uma função distribuição de probabilidades polinomial.

Esse trabalho limita-se à avaliação dos requisitos de comunicação das aplicações paralelas, desconsiderando seus impactos sobre o escalonamento de processos.

5.4 Considerações finais

Políticas de escalonamento de processos definem as regras de distribuição dos processos sobre os EPs. O resultado de um escalonamento eficiente, ou seja, que alcança os objetivos pelos quais se propõe auxilia na utilização eficiente do sistema. Entretanto, esse escalonamento é considerado um problema NP completo e várias políticas de escalonamento para resolver esse problema foram propostas (OH; HA, 1996; PARK; CHOE, 2002; RANAWEERA; AGRAWAL, 2000; J.Y.COLIN; P.CHRÉTIENNE, 1991; DARBHA; AGRAWAL, 1998; TOPCUOGLU et al., 1999; CHOE; PARK, 2002; RADULESCU et al., 1999).

Embora a literatura tenha demonstrado a necessidade de especialização das políticas de acordo com a classe da aplicação, pouco esforço tem sido notado para a definição e implementação de políticas destinadas às aplicações não orientadas à CPU (*CPU-Bound*). Esse fato indica a necessidade da definição e implementação de políticas que tratem aplicações com diferentes padrões de utilização de recursos, e não apenas aplicações *CPU-Bound*. Um exemplo são as aplicações *I/O-Bound*, que se caracterizam principalmente pelas operações de entrada e saída através do acesso aos dispositivos locais de armazenamento e/ou por meio da utilização da rede de comunicação.

As limitações dos trabalhos apresentados neste capítulo, motivam o estudo das características de ocupação da rede de comunicação, bem como o impacto dessa ocupação no escalonamento de processos. Essas limitações estão relacionadas às características de comunicação (volume, tráfego, *overhead*) das aplicações e do impacto dessa atividade na troca de informações sobre a CPU. Dessa maneira, o próximo capítulo apresenta uma avaliação do comportamento do subsistema de comunicação e qual sua influência no escalonamento de processos.

CAPÍTULO

6

Escalonamento baseado na avaliação do impacto da comunicação entre processos

6.1 Considerações iniciais

Conforme apresentado no capítulo 5, vários trabalhos sobre escalonamento de processos foram propostos. Dentre esses trabalhos, existem aqueles que definem estratégias para distribuição de processos sobre os EPs, tais como: Ranaweera e Agrawal (2000), Choe e Park (2002), Radulescu et al. (1999), Araújo et al. (1999), Mache et al. (2000). Outros trabalhos consideram características de comunicação entre processos: Mao et al. (1999), Ni et al. (1985), Ryou e Wong (1989), Keren e Barak (2003), Chodnekar et al. (1997). Esses trabalhos apresentam análises do impacto da transmissão e recebimento de informações entre os processos das aplicações. A partir dessas análises é possível definir estratégias de escalonamento que levem em consideração, tanto os requisitos das aplicações, quanto os recursos computacionais envolvidos na comunicação entre processos.

As análises dos trabalhos anteriormente apresentados conduziram o estudo sobre o impacto do processamento sobre a rede de comunicação e, conseqüentemente, a adoção da rede como um parâmetro nas decisões de escalonamento. Durante o desenvolvimento desses estudos foi definido um novo índice de carga baseado na ocupação de CPU e da rede de comunicação. Esse índice de carga foi aplicado à política de escalonamento DPWP, gerando uma extensão dessa política, denominada NBSP. Em seguida, essa extensão foi comparada à DPWP original. A NBSP, utilizando o índice de carga proposto, obteve maior desempenho, demonstrando a importância em se considerar a rede de comunicação como um parâmetro nas decisões de escalonamento.

6.2 Avaliação do impacto no processamento causado pela comunicação

As limitações relacionadas à quantificação da carga de comunicação sobre a rede e a inexistência de uma análise de influência da comunicação sobre o escalonamento, encontradas nos trabalhos (MAO et al., 1999; NI et al., 1985; RYOU; WONG, 1989; KEREN; BARAK, 2003; CHODNEKAR et al., 1997), motivam o estudo do impacto no processamento causado pela comunicação.

Esse impacto advém da sobrecarga (*overhead*), latência e volume (tamanho e freqüência de troca de mensagens) transmitido de mensagens. Tais parâmetros são utilizados neste trabalho para modelar o comportamento de comunicação dos processos. A sobrecarga representa o tempo consumido no empacotamento e desempacotamento de mensagens. A latência é o tempo consumido na transmissão de uma mensagem sobre a camada física de rede. O volume é a quantidade de bytes transferidos na rede em uma determinada janela de tempo (por exemplo, em bytes/segundo).

Dessa maneira, é definido um modelo para quantificação da carga de comunicação dos processos sobre a rede de comunicação do sistema e, a partir desse modelo, é proposta uma extensão da política DPWP. Essa abordagem visa a utilização da rede de comunicação como um parâmetro nas decisões de escalonamento. As próximas subseções descrevem a política de escalonamento DPWP, o modelo de comunicação empregado e a extensão da política DPWP, respectivamente.

6.2.1 DPWP - *Dynamic Policy Without Preemption*

A DPWP é definida pelos quatro componentes principais propostos por Shivaratri et al. (1992), que são: política de transferência, política de seleção, política de localização e política de informação.

A política de seleção busca selecionar os processos que podem participar da distribuição de carga. A DPWP considera que os processos novos que chegam no sistema são candidatos a participar na distribuição de carga. Entretanto, esses processos devem ser considerados antes de iniciar a execução, devido o fato da DPWP não implementar migração de processos.

A política de transferência verifica qual elemento de processamento deve ser considerado como receptor ou transmissor de carga. Na DPWP, o elemento de processamento com maior carga é considerado um transmissor e, por conseguinte, pode realizar a transferência para outro EP menos carregado.

A política de localização na DPWP é responsável por identificar qual EP será o receptor quando ocorrer uma transferência de dados. Isso foi implementado de maneira descentralizada permitindo assim que um EP transmissor procure por um receptor que tenha o menor índice

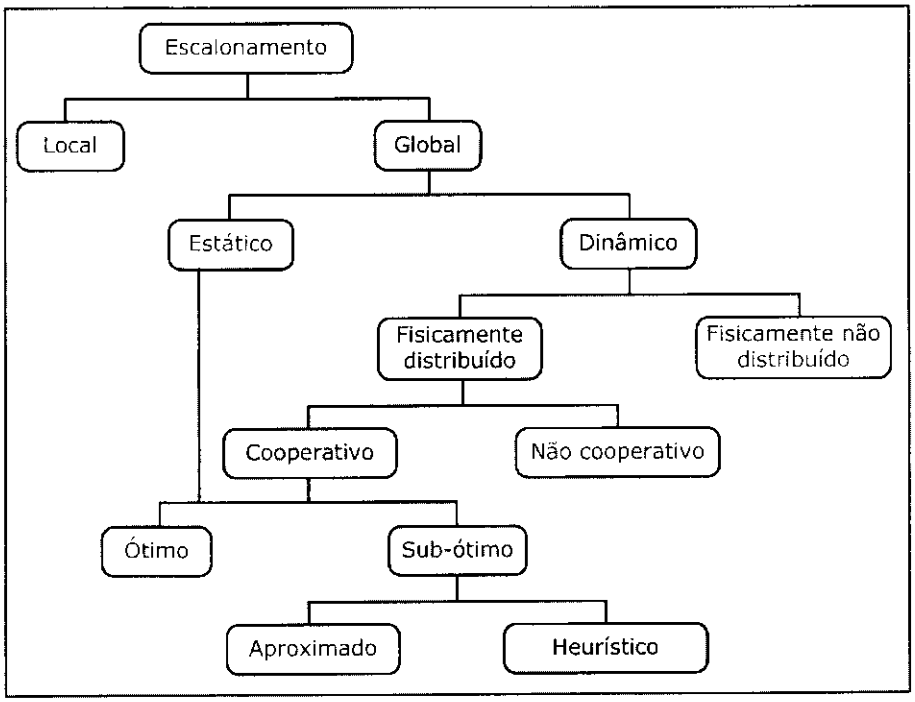

Figura 6.1: Classificação de Casavant para a DPWP

de carga. A decisão de qual é o EP com menor índice ocorre por meio de uma tabela com informações de carga.

Na DPWP, a política de informação decide quando as informações sobre outros EPs devem ser coletadas, quais informações devem ser coletadas e onde coletar. A DPWP implementa um modelo de política de informação dirigida à mudança de estado. Um EP envia suas informações apenas quando seu estado de carga é alterado acima de um valor predefinido. Essas informações são enviadas para todos os EPs que participam ou podem vir a participar do escalonamento. O objetivo disso é permitir que todos eles tenham a informação atualizada sobre a situação do sistema.

O índice de carga utilizado pela DPWP é a média de processos na fila da CPU nos últimos 20 segundos. O índice é atualizado nesse intervalo de tempo e uma janela é mantida com os valores das coletas passadas para o cálculo da média.

Segundo a classificação hierárquica apresentada por Casavant e Kuhl (1988), a política DPWP é uma política de escalonamento global, dinâmica, fisicamente distribuída e não cooperativa. A figura 6.1 mostra a classificação de Casavant para a política DPWP.

A DPWP é classificada como global, pois é considerado um sistema com vários processadores. Isso corresponde à tomada de decisão de qual EP executará um determinado processo. Dinâmica, pois as tomadas de decisão são realizadas em tempo de execução, fisicamente distribuída, pois todos os EPs têm o mesmo poder de decisão. A DPWP é considerada cooperativa, pois a tomada de decisão em um determinado EP leva em consideração os resultados que essa decisão possa causar sobre o sistema.

A política DPWP é classificada como sub-ótima, pois a geração de escalonamentos ótimos é um problema do tipo NP-completo (RANAWEERA; AGRAWAL, 2000). Nesse tipo de problema, só é possível gerar soluções ótimas em casos específicos, por exemplo: em uma situação em que o tempo de execução de todas as tarefas é igual e somente dois processadores são utilizados. Em casos gerais, isso é muito difícil, visto que um problema NP-completo requer um período de tempo polinomial no tamanho de sua entrada e exponencial para ser solucionado.

Como alternativa, o escalonamento sub-ótimo busca encontrar uma solução aceitável para o problema. Essa solução pode seguir duas abordagens: aproximada e heurística. A solução aproximada pode adotar algum método de busca conhecido, enquanto a heurística utiliza regras especiais, como inteligência artificial por exemplo, para obter a solução. A DPWP adota a abordagem heurística.

6.2.2 Modelo de comunicação

O modelo de comunicação define o comportamento do volume de tráfego presente na rede de comunicação. Esse volume é parametrizado por meio da latência e da sobrecarga. Esse modelo busca quantificar a carga imposta sobre a rede de comunicação e o atraso sobre os EPs devido às operações na rede. Esse atraso é definido como o *slowdown* causado pelo empacotamento e desempacotamento das mensagens nos elementos de processamento.

O modelo também define equações que quantificam os custos de processamento e de comunicação dos elementos de processamento em execução de aplicações paralelas. Também são definidas equações para determinar a largura máxima de banda (*bandwidth*), utilizada para as tomadas de decisões de escalonamento.

As equações que caracterizam o impacto da comunicação são baseadas no modelo LogP (CULLER et al., 1993) e são formalmente descritas pela equação (6.1), onde: $A_{m,i}$ representa o atraso, em segundos, que uma mensagem m causa aos processos localizados no computador i, definido como emissor, e no computador j, definido como receptor da mensagem; O_m é a sobrecarga para empacotar a mensagem m no computador i e desempacotá-la no j, fato que justifica sua multiplicação pela constante 2; L_m é a latência, em segundos, que representa o tempo de tráfego da mensagem pelo meio físico.

$$A_{m,i,j} = 2 * O_m + L_m \tag{6.1}$$

Outra característica importante do modelo é a definição de uma constante k, que determina a máxima utilização da rede de comunicação. A distribuição dos processos é realizada de acordo com essa constante. Com base nessa constante e na ocupação relativa que cada computador do sistema tem sobre a rede, são adotadas duas técnicas para atribuição dos processos aos EPs: a primeira aloca todos os processos de uma mesma aplicação no mesmo elemento de

processamento, denominada escalonamento em grupo e a segunda distribui, eqüitativamente, os processos entre os elementos de processamento disponíveis por intermédio da rede de comunicação.

A adoção de uma das técnicas depende da constante k, que delimita a quantidade de largura de banda utilizada por computador na rede. Essa largura de banda utilizada é determinada pela equação (6.2), onde: α representa cada um dos processos alocados no computador i; $nprocs$ é a quantidade de processos no computador i; N_α é a quantidade de bytes por segundo, atualmente em transferência.

$$NC_i = \sum_{\alpha=1}^{nprocs} N_\alpha \tag{6.2}$$

Caso a carga de comunicação imposta pelos processos, NC_i do EP i, seja superior ao valor definido para k ($k < NC_i$), é realizado um escalonamento em grupo. Caso contrário, enquanto a carga de comunicação é inferior a k ($k > NC_i$), os processos são alocados nos EPs disponíveis por intermédio da rede de comunicação.

Distribuir os processos segundo um escalonamento em grupo, aparentemente, pode resultar numa solução pouco eficaz, uma vez que, EPs podem permanecer ociosos. Entretanto, estudos comprovam que no decorrer do tempo, conforme novos processos são submetidos e começam a ocupar os EPs ociosos, melhora-se a utilização do sistema (FEITELSON, 2001).

6.2.3 Política NBSP (*Network-Bound Scheduling Policy*)

Para avaliar o comportamento dos parâmetros definidos pelo modelo anteriormente apresentado, sobre um escalonador de processos, propôs-se uma extensão da política de escalonamento DPWP (*Dinamic Policy Without Preemption*) (ARAÚJO et al., 1999). A política DPWP foi desenvolvida no grupo de pesquisas no qual este projeto está inserido. A extensão da política DPWP criada é denominada NBSP.

A política DPWP, originalmente cria um vetor que descreve a capacidade livre C_i (equação 6.3) de cada processador (EP), onde $i \in [1, n]$ representa cada elemento de processamento, $CPU_{length,i}$ o tamanho da fila de processos do EP i em determinado instante e $Bench_i$ a capacidade total do EP i, medida por um *benchmark* adotado.

$$C_i = \frac{CPU_{length,i}}{Bench_i} \tag{6.3}$$

Esse vetor é mantido em ordem ascendente em cada um dos processadores do sistema. Quando um processador recebe uma requisição para executar um grupo de processos, a DPWP os distribui seqüencialmente sobre os processadores desse vetor. Caso a quantidade de processos seja maior que o número de processadores, a DPWP reinicia a distribuição dos processos a partir

do processador localizado na primeira posição do vetor. Dessa forma, privilegia-se a alocação de processos, primeiramente, sobre os processadores mais ociosos.

A NBSP incorpora duas modificações propostas no modelo de comunicação: adoção de equações que caracterizam o impacto de comunicação e distribuição de processos de acordo com a constante k, que representa a quantidade de utilização da rede de comunicação.

A NBSP é descrita a seguir. Essa descrição procura formalizar a atuação da política na atividade de escalonamento de processos.

Algoritmo 6.1 NBSP - *Network-Bound Scheduling Policy*

```
1: coleta_índice_cpu();
2: ordena_cpu();
3: enquanto existe processos na fila do escalonador faça
4:    com ← coleta_carga_rede();
5:    enquanto com ≤ k faça
6:       escalone_rede();
7:    fim enquanto
8:    escalone_local();
9: fim enquanto
```

As linhas 1 e 4 do algoritmo 6.1 representam a coleta de informações dos índices de carga relacionados à capacidade de processamento e à taxa de transmissão, respectivamente. A linha 2 executa a rotina que ordena os elementos de processamento segundo sua capacidade computacional. Isso é realizado por meio do *benchmark* TSCP, o qual é descrito no capítulo 7. As linhas 6 e 8 dizem respeito ao escalonamento propriamente dito, que fica condicionado aos índices de carga de CPU e de rede, demonstrando-se a importância do uso de índices de carga na atividade de escalonar processos em um sistema computacional distribuído.

A rotina ordena_cpu é responsável por classificar os elementos de processamento. Essa classificação considera as capacidades em MIPS e a carga dos processos presentes no elemento de processamento. O algoritmo 6.2 descreve a rotina ordena_cpu. Essa rotina aplica o método da bolha para ordenação dos EPs do sistema. Tal método é utilizado por ter baixa complexidade em relação ao tamanho do problema. Esse tamanho corresponde ao número de elementos de processamento disponíveis no sistema nEP, que em sistemas distribuídos, geralmente, não são valores altos. Um sistema distribuído com 600 EPs já é difícil de se encontrar, no entanto para o algoritmo 6.2 esse número corresponde a entradas pequenas e, conseqüentemente, tempos de execução baixos do algoritmo de ordenação.

Nos experimentos, os valores para a constante k sofrem variações com objetivo de definir suas melhores configurações. Podem-se observar melhorias nas decisões de escalonamento, conseguindo, conseqüentemente, aumento no desempenho das aplicações paralelas.

Algoritmo 6.2 Ordena CPU

```
1: EP ← vetor com elementos de processamento;
2: nEP ← número de elementos de processamento;
3: i ← 0;
4: j ← 0;
5: enquanto i ≤ nEP faça
6:    j ← i+1;
7:    enquanto j ≤ nEP faça
8:       se EP[i].nprocs / EP[i].mipsCapacity ≥ EP[j].nprocs / EP[j].mipsCapacity então
9:          troca (EP[i], EP[j]);
10:      fim se
11:      j ← j+1;
12:   fim enquanto
13:   i ← i+1;
14: fim enquanto
```

6.3 Considerações finais

Este capítulo apresentou a análise do impacto no processamento causado pela comunicação entre processos. Tal análise resultou na definição de um novo índice de carga que considera a ocupação da rede de comunicação e, conseqüentemente, auxilia nas tomadas de decisões de escalonamento. Esse índice foi aplicado à política de escalonamento DPWP gerando uma extensão dessa política, denominada NBSP.

A NBSP é indicada para o escalonamento de aplicações *Network-Bound* em ambientes homogêneos e heterogêneos. Resultados por simulação permitiram comprovar as contribuições dessa política estendida. Essas contribuições são referentes ao aumento de desempenho das aplicações que realizam comunicação intensa e melhor utilização da rede de comunicação no instante da distribuição dos processos que compõem as aplicações paralelas. Os resultados obtidos, através de simulação, e sua análise são descritos no próximo capítulo.

CAPÍTULO

7

Avaliação de desempenho e resultados

7.1 Considerações iniciais

Este capítulo apresenta os resultados obtidos por meio de experimentos práticos e simulações, que visam avaliar o comportamento da política de escalonamento NBSP em relação à DPWP. Os experimentos práticos servem para definir valores dos índices de carga considerados (carga de CPU e rede de comunicação) e, à partir desses índices, a simulação é parametrizada. Utilizando parâmetros tais como sobrecarga, latência e volume transmitido de mensagens, é possível deixar a simulação com características mais próximas de uma execução real. Tanto as ferramentas desenvolvidas para realização dos experimentos práticos, quanto o simulador adotado, são discutidos nas seções seguintes.

7.2 Experimentos para aquisição dos parâmetros de simulação

Para avaliar o impacto de processamento causado pela comunicação, o simulador desenvolvido por Mello e Senger (2004) [1] foi estendido. Essa extensão agrega parâmetros específicos para quantificação da carga sobre o meio físico de transmissão de mensagens: sobrecarga de comunicação, latência, volume e freqüência de transmissão.

A sobrecarga é representada por meio da relação MIPS consumidos por bytes transferidos. A latência, pela taxa de transferência dos pacotes através da rede e o volume, por funções de distribuição de probabilidade.

[1] disponível em: www.icmc.usp.br/ mello/outr.html

Tabela 7.1: Configuração dos EPs utilizados nos experimentos

EP	CPU	Memória	Placa rede
1	P4, 2.4 Ghz	512 MB	10/100
2	Athlon, 1.67 Ghz	256 MB	10/100
3	P3, 600 Ghz	256 MB	10/100
4	P2, 350 Mhz	128 MB	10/100

Com o objetivo de capturar a relação MIPS consumidos por bytes transferidos (equação 7.3) e latência (equação 7.2), foi desenvolvido um programa baseado no modelo cliente/servidor. No código do cliente, foram implementadas rotinas para envio de pacotes com tamanho fixo (64 Kbytes, tamanho máximo do pacote ethernet; esse valor adotado permite medir tempos de resposta em condições de ocupação máxima do canal).

No servidor, foram implementadas rotinas para recebimento dos pacotes. Na medição da taxa de transmissão através da interface de rede, os programas cliente e servidor foram executados em dois computadores distintos.

Para realizar os experimentos sobre um ambiente heterogêneo, foram analisados quatro computadores com capacidades distintas de processamento (tabela 7.1). Em cada computador é executado o *benchmark* TSCP (KERRIGAN, 2004).

O TSCP mede o desempenho, no pior caso, de operações que manipulam números inteiros. Esse *benchmark* é muito similar ao SPEC CINT2000 (HENNING, 2000). Ele utiliza como base para comparação uma estação de trabalho SGI *Silicon Graphics*®[2], a qual consome 11,233 milhões de MIPS para executar a função *bench()*. Utiliza também o comando *perfex* para coletar as informações sobre o TSCP. Para efeito de comparação, esse comando *perfex* é executado em conjunto com os programas *gcc* e *gzip* (dois programas incorporados ao SPEC CINT2000).

A figura 7.1 apresenta um gráfico comparativo entre esses *benchmarks*, onde C/A é o desempenho em operações de carga e armazenamento (*load/store*), desvios (*branches*), operações lógicas/aritméticas (ULA), comunicação entre processos (IPC - *Interprocess Communication*), percentual de erro nas predições de desvio (P. desvio) e a taxa de erro em cache L1.

Em cada computador do ambiente heterogêneo é executado o *benchmark* TSCP sobre duas condições de carga: computador ocioso e executando operações de transferência de pacotes através da interface de rede.

O TSCP retorna os valores da capacidade de processamento em MIPS. Esse *benchmark* executa até que os valores convirjam e atinjam um intervalo de confiança de 95%. A carga de comunicação é estabelecida pelo programa cliente/servidor, que fica enviando (cliente) e recebendo (servidor) pacotes através da interface de rede, enquanto o intervalo de tempo Δt

[2]Estações de trabalho projetadas para atenderem os requisitos de alto desempenho de aplicações científicas, de engenharia e para editoração gráfica.

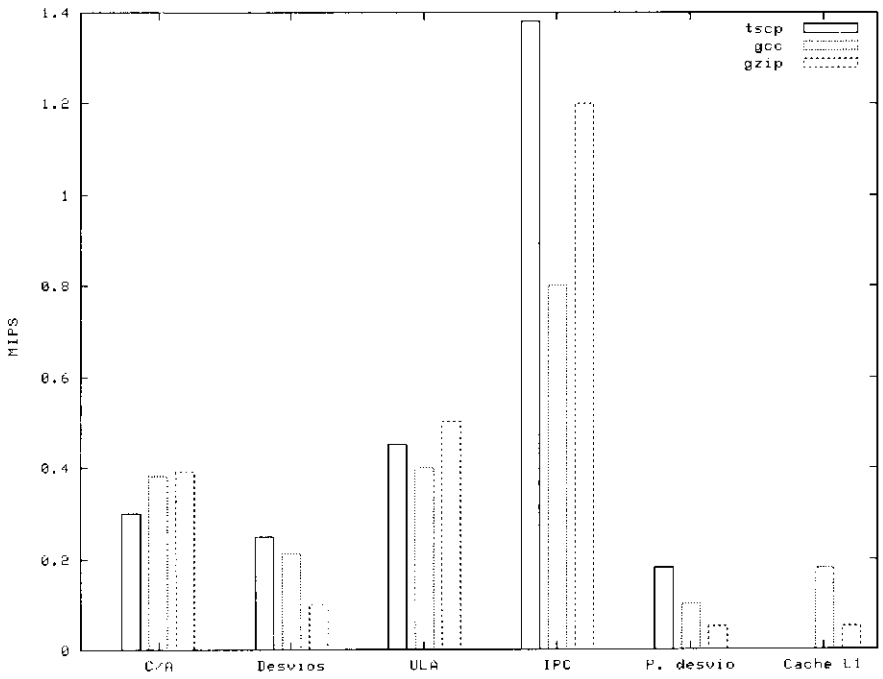

Figura 7.1: Comparação entre os *benchmarks* TSCP, gcc e gzip

(equação 7.2) não é alcançado e o intervalo de confiança chegue a 95%.

Uma versão mais recente do simulador implementa um conjunto de *benchmarks*. Esses *benchmarks* têm o objetivo de capturar a capacidade de CPU (mips), atraso devido a troca de contexto entre os processos (memo), taxa de leitura e escrita em disco (discio) e tempo gasto no envio e recebimento de mensagens (net). O *benchmark* net segue a mesma idéia do programa cliente/servidor e pode ser utilizado para captura do atraso da comunicação entre processos.

Utilizando essa técnica, é possível identificar a queda no desempenho do elemento de processamento durante transferências de pacotes pela rede. Para auxiliar no cálculo dessa queda de desempenho é definida a equação (7.1), onde: Mc_i é a quantidade de MIPS consumidos pelo EP i transferindo pacotes através da rede de comunicação, Mo_i é o valor de execução do *benchmark* TSCP no EP ocioso; e Mt_i, o valor de execução do TSCP no EP em transferência de pacotes pela rede.

$$Mc_i = Mo_i - Mt_i \tag{7.1}$$

A equação (7.2) quantifica a largura de banda de rede ocupada por intermédio do programa cliente/servidor, onde: Tx é a taxa de transferência da rede, $pcts$ é a quantidade total de pacotes transmitidos no intervalo de tempo Δt e len é o tamanho do pacote, em bytes.

$$Tx = \frac{pcts \times \frac{len}{1024}}{\Delta t} \tag{7.2}$$

A equação (7.3) relaciona as equações (7.1 e 7.2) definindo a capacidade de comunicação

Tabela 7.2: Exemplo de uso da equação (7.3)

Parâmetro	Valor
Mo_x	1.541,55 MIPS
Mt_x	1.203.61 MIPS
Tx	11.000 Kbytes/s

de um determinado EP em Kbytes/segundo/MIPS, onde: Cc_i é a capacidade de comunicação do EP i, Tx é a taxa de transmissão (equação 7.2) e Mc_i é a quantidade de MIPS consumidos pelo EP em transferência de pacotes pela rede (equação 7.1).

$$Cc_i = \frac{Tx}{Mc_i} \tag{7.3}$$

Um exemplo de uso da equação (7.3) pode ser obtido considerando um EP_x com as características apresentadas na tabela 7.2.

Os valores para Mo_x e Mt_x são obtidos por meio da execução do *benchmark* TSCP. Tx é obtida por meio do programa cliente/servidor. Aplicando-se a equação (7.3) obtêm-se as equações (7.4) e (7.5).

$$\begin{aligned} Mc_x &= 1.541,55 - 1.203,61 \\ &= 337,94 \end{aligned} \tag{7.4}$$

$$\begin{aligned} Cc_x &= \frac{11.000}{337,94} \\ &= 32,55 KBytes/s/MIPS \end{aligned} \tag{7.5}$$

As equações (7.4) e (7.5) são unificadas para encontrar uma razão da quantidade de MIPS consumidos na transmissão de um conjunto de dados. No exemplo anteriormente apresentado, 1 MIPS é consumido pelo EP_x na transferência de 32,55 Kbytes por segundo (equação 7.5).

O consumo de MIPS pelo EP em transferência de dados na rede é considerado como um *slowdown* causado pelo empacotamento e desempacotamento das mensagens. Tal *slowdown* segue as definições de sobrecarga consideradas no modelo LogP (CULLER et al., 1993) e anteriormente definidas no capítulo 6 (equação 6.1).

Essa observação da queda de desempenho, *slowdown* e sobrecarga, motiva a realização de experimentos, que permitem parametrizar o simulador. Além disso, esses experimentos permitem a aquisição de resultados mais semelhantes a execuções no ambiente real. Foram realizados experimentos sobre os computadores com características descritas na tabela 7.1.

O primeiro gráfico (figura 7.2), apresenta os valores da capacidade de processamento para cada EP. A primeira coluna mostra a capacidade dos EPs no estado ocioso, ou seja, não

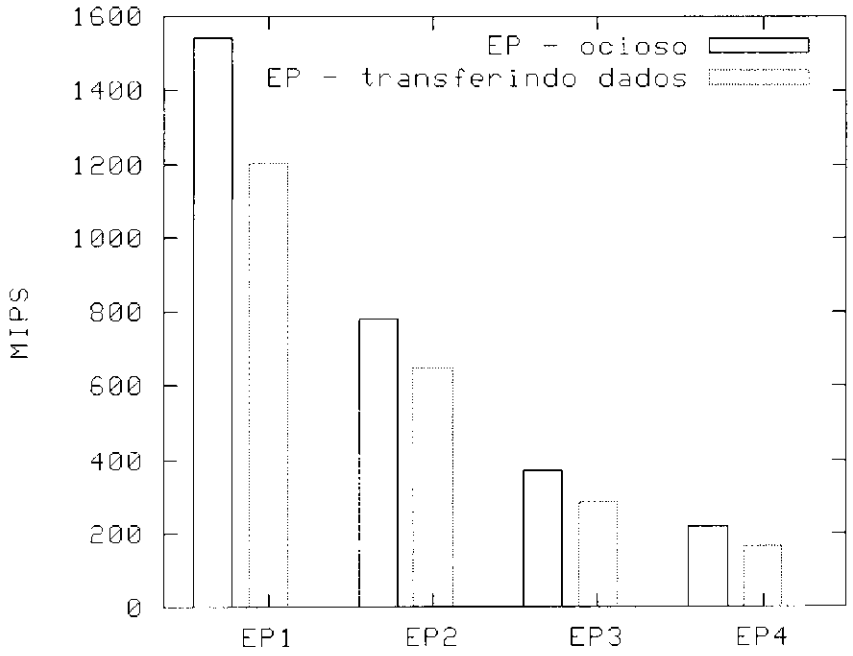

Figura 7.2: Capacidade de processamento dos EPs

executam operações sobre a rede de comunicação. Na segunda coluna, os valores em MIPS dos EPs em transferências pela rede. Esses EPs executam o programa cliente/servidor, que realiza operações de envio/recebimento através da rede de comunicação e calcula a taxa de transferência (equação 7.2).

A partir desses resultados é possível definir a relação Kbytes/segundo/MIPS, equação (7.3), que auxilia na parametrização do simulador deixando-o com características mais próximas do real.

Além desses experimentos, são realizadas avaliações, visando a obtenção da taxa de transmissão dos EPs. Tais avaliações consistem em executar o programa cliente/servidor em cada EP sobre duas situações: EPs ociosos e executando o *benchmark* TSCP. Os valores da taxa de transmissão resultantes estão compreendidos entre 11.500 e 12.000 Kbytes/s. Observa-se que a taxa de transmissão permanece sem grandes variações em todos os EPs, mesmo executando, simultaneamente, operações sobre a CPU (TSCP) e sobre a rede de comunicação (programa cliente/servidor). O intervalo de confiança considerado é de 95% nas execuções do programa cliente/servidor e do TSCP.

7.3 Simulações

As análises anteriormente citadas, em relação aos MIPS consumidos para transmitir uma determinada taxa, em Kbytes/segundo, permitem parametrizar o simulador. Esses parâmetros correspondem à quantidade de MIPS que um determinado processo consome de CPU, devido à

sobrecarga causada pelo empacotamento e desempacotamento de mensagens. Agrega-se a essa quantidade, o total de MIPS gastos por esse processo em transmissão pela rede, equação (7.3).

Utilizar um modelo de simulação é atrativo, pois isso segue uma tendência encontrada nos trabalhos da área de escalonamento de processos, nos quais adotam a simulação como ferramenta de apoio na avaliação do escalonamento. Tal tendência pode ser observada nos trabalhos de Mello e Senger (2004), Keren e Barak (2003), Mao et al. (1999), Feitelson (2001), Mu'alem e Feitelson (2001).

A simulação permite total controle dos parâmetros de entrada e fornece resultados que se aproximam de uma execução real, pois tais parâmetros são obtidos através de resultados coletados de experimentos reais. Nesses experimentos, cada execução é realizada até que se atinja um intervalo de confiança de 95%. O simulador é configurado por meio desses experimentos que visam representar o comportamento dos processos das aplicações e de cada elemento de processamento do ambiente computacional distribuído.

O modelo de carga de Feitelson é utilizado para representar o comportamento dos processos das aplicações paralelas. Representa cargas de trabalho reais e possibilita que simulações sejam realizadas a partir dele. Esse modelo é baseado na análise de seis *logs* de execução dos seguintes ambientes:

- 128-node iPSC/860 at NASA Ames;
- 128-node IBM SP1 at Argonne;
- 400-node Paragon at SDSC;
- 126-node Butterfly at LLNL;
- 512-node IBM SP2 at CTC;
- 96-node Paragon at ETH, Zurich.

Esse modelo define o tempo de execução dos processos como uma função de distribuição de probabilidades de cauda pesada. Isso porque, eventualmente, a carga de trabalho sobre o sistema torna-se excessivamente alta, o suficiente para contrabalançar as situações de baixa carga de ocupação. O modelo de Feitelson sintetiza as cargas impostas pelas aplicações, gerando um único arquivo de *log*; cada linha desse *log* representa uma aplicação submetida ao sistema.

Mesmo sendo utilizado como carga de trabalho em diversos trabalhos (MELLO; SENGER, 2004; MU'ALEM; FEITELSON, 2001; TALBY et al., 1999; FEITELSON, 2001; FEITELSON, 1996; FEITELSON; RUDOLPH, 1998), é interessante aplicar outras cargas de trabalho como entrada para o simulador, demonstrando sua flexibilidade e atestando a política de escalonamento *Network-Bound* apresentada neste trabalho.

A ocupação dos processos no sistema segue o modelo de Feitelson (MU'ALEM; FEITELSON, 2001; TALBY et al., 1999; FEITELSON, 2001; AIDA, 2000). Esse modelo fornece duas informações ao simulador: carga e número de processos. A carga representa a quantidade de MIPS que cada processo consome de CPU. O número de processos, a quantidade de processos de uma determinada aplicação paralela. O simulador gera os resultados em segundos, na média do tempo de resposta dos processos e adota um intervalo de confiança de 95%, ou seja, esses resultados são gerados até que convirjam e atinjam 95%.

Esse intervalo de confiança segue as definições de (SCHEFLER, 1988), que permitem o cálculo da variância em tempo de execução da simulação. Dessa maneira, não é necessário o término da execução das n (adota-se $n \geq 30$)[3] simulações para o cálculo da variância.

No simulador, o intervalo de chegadas de processos ao sistema é definido por uma função de distribuição de probabilidades Poisson com média de 1500 segundos. Essa distribuição foi adotada a partir dos experimentos de Feitelson (2001), Feitelson (1996), demonstrando que a chegada de processos em ambientes paralelos determinados pelos *logs* de execução seguem essa mesma distribuição, o qual foi adotado como carga de trabalho nas simulações.

Além da sobrecarga e da latência, definidos anteriormente, o modelo proposto neste trabalho requer outros parâmetros: o tamanho das mensagens, ou seja, a quantidade de bytes transferidos através da rede; e o tempo gasto para transmissão dessas mensagens. Esses parâmetros são utilizados, segundo a equação (7.6), para mensurar o volume médio de mensagens transmitidas pelo canal de comunicação.

Para compreender o uso desses parâmetros no simulador, considere-se a situação na qual a quantidade de bytes que um processo transmite seja representada por uma Poisson com média 1.000 Kbytes. Da mesma maneira, representa-se o tempo de transferência dessa quantidade por meio de uma Poisson com média 10 segundos.

Nessa situação, cada processo apresenta ocupação de rede da ordem de 100 Kbytes/s. Esse valor é obtido por meio da equação (7.6), onde, $Peso_{com}$ expressa quanto um determinado processo consome de largura de banda, *carga* é o valor em Kbytes transmitidos e *tempo* o intervalo de tempo gasto na transferência de *carga* através da rede de comunicação.

$$Peso_{com} = \frac{carga}{tempo} \tag{7.6}$$

O simulador adota quaisquer funções de distribuição de probabilidades definidas na biblioteca PSOL[4] e podem ser escolhidas pelo usuário. Nos experimentos conduzidos, a taxa de transmissão das mensagens (equação 7.6), é representada por uma função de distribuição de

[3]Definido pelo Teorema Central do Limite, onde: qualquer que seja a distribuição da variável de interesse para grandes amostras, a distribuição das médias amostrais serão aproximadamente normalmente distribuídas e tenderão a uma distribuição Normal à medida que o tamanho da amostra crescer. Uma amostra grande é definida como $n \geq 30$

[4]Uma biblioteca estatística licenciada sob GNU/GPL e desenvolvida por University of Alabama

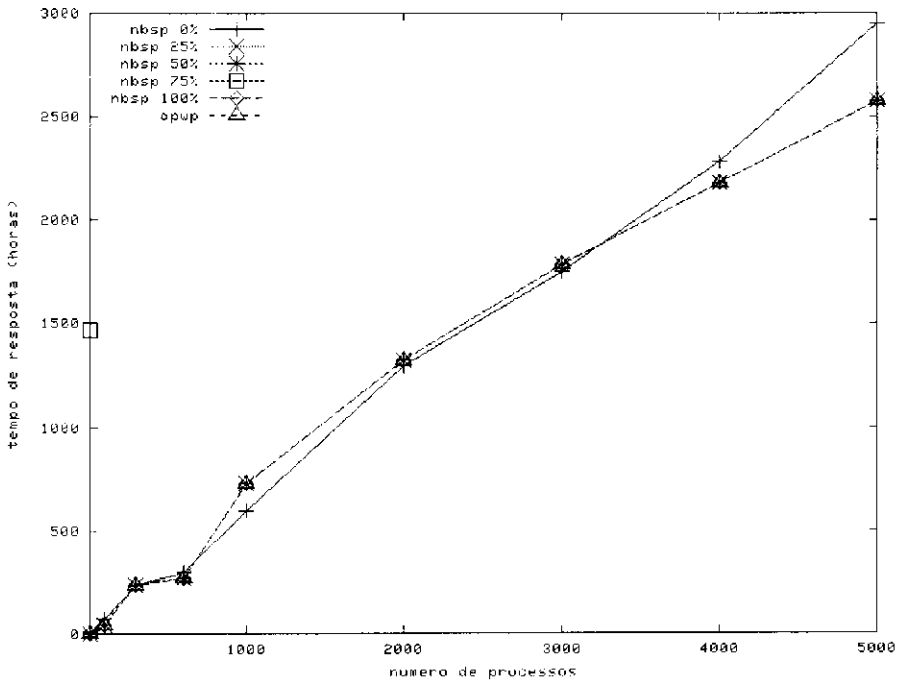

Figura 7.3: Poisson carga = 500 Kbytes; tempo = 60 segundos

probabilidades Poisson. Tal função foi adotata segundo experimentos comparativos, nos quais utilizou-se a função hiperexponencial, definida no trabalho de Chodnekar et al. (1997) para caracterização dessa mesma taxa e verfificou-se que a Poisson segue esse mesmo comportamento.

Para investigar a influência da comunicação sobre o escalonamento de processos, foram realizados experimentos comparativos entre as políticas DPWP e NBSP. Ambas foram implementadas no simulador desenvolvido em Mello e Senger (2004), permitindo comparar o tempo de resposta médio dos processos. Houve variações dos valores para a constante k, observando-se o comportamento da política estendida em diferentes situações da carga de ocupação da rede.

Os resultados desses experimentos são apresentados nos gráficos das figuras 7.3, 7.4 e 7.5. O eixo y representa o tempo de resposta médio dos processos (em horas) e o eixo x o número de processos submetidos ao sistema.

A figura 7.3 apresenta os resultados obtidos no experimento que utiliza a função de distribuição de probabilidades Poisson. Essa função representa a carga de ocupação dos processos sobre a rede, com média de 500 Kbytes e tempo de transferência de 60 segundos. Ainda, considera-se na NBSP, variações dos valores para a constante k, que representa o limite máximo de utilização da rede, em: 0%, 25%, 50%, 75% e 100%.

Essa constante k influencia as decisões de escalonamento, pois tal política, por meio da equação (6.2), verifica o total de largura de banda utilizada pelos processos. Conforme o valor de retorno dessa equação (6.2) (maior ou menor que a constante k), decide-se distribuir os processos entre os EPs disponíveis; ou em grupo, ou no mesmo EP.

Observando a figura 7.3, o desempenho entre DPWP e NBSP são equivalentes, ressal-

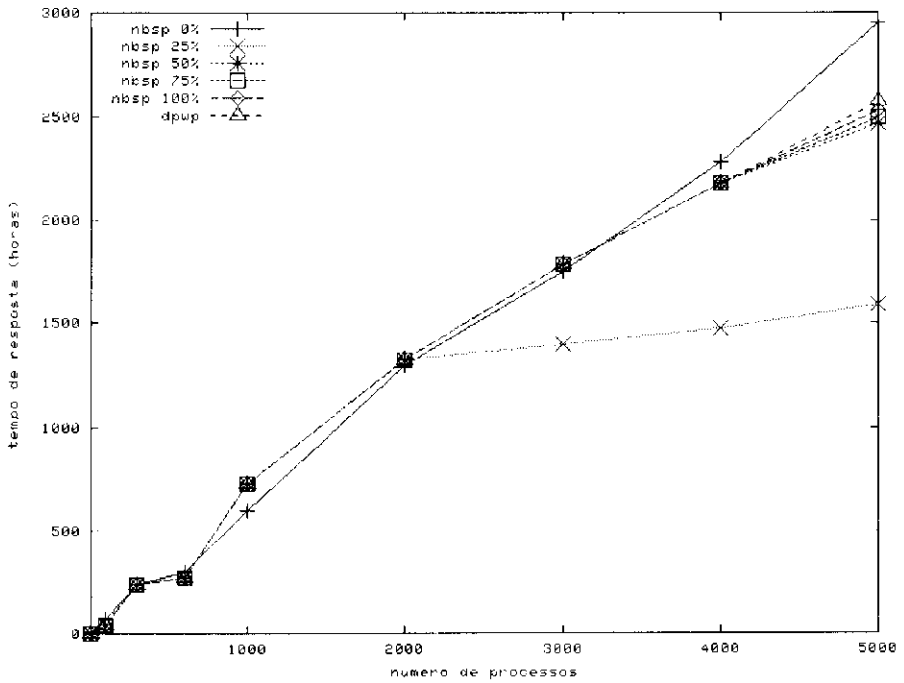

Figura 7.4: Poisson carga = 10.000 Kbytes; tempo = 60 segundos

tando o fato que, na configuração 0%, o desempenho da NBSP é menor. Isso se deve à alocação de todos os processos da mesma aplicação em um único EP, diferente de um escalonamento regido pela DPWP, que distribui os processos entre os EPs melhorando a ocupação e, conseqüentemente, balanceando a carga do sistema.

A NBSP obtém melhores resultados conforme a carga de comunicação do sistema aumenta. São apresentados os resultados do experimento parametrizado segundo uma função distribuição de probabilidades Poisson, com média 10.000 Kbytes para carga de ocupação e 60 segundos de tempo de transferência (figura 7.4).

Nessas condições de carga, à partir do instante no qual os processos não são distribuídos pelos computadores de forma a ocupar o subsistema de comunicação, diminui-se a espera por sincronização entre os processos nos EPs e, conseqüentemente, melhora-se o tempo de resposta das aplicações, que realizam muitas operações sobre a rede de comunicação.

Pode-se, ainda, concluir, por meio da figura 7.4, que a NBSP sobre variação de 25% de uso da rede, melhora o desempenho em relação a DPWP. À medida que a carga de ocupação dos processos na rede de comunicação aumenta, a NBSP causa menores atrasos aos EPs e diminui o tempo médio de resposta das aplicações.

Na figura 7.5, são apresentados os resultados do experimento parametrizado segundo uma função distribuição de probabilidades Poisson. Para representar a carga de ocupação, utiliza-se tal função com média 50.000 Kbytes e 60 segundos de tempo de transferência. A NBSP, em todos os percentuais de utilização da rede, exceto 0%, aumenta, consideravelmente, o desempenho das aplicações em relação a DPWP. Isso pode ser observado mesmo quando o sistema apresenta-se

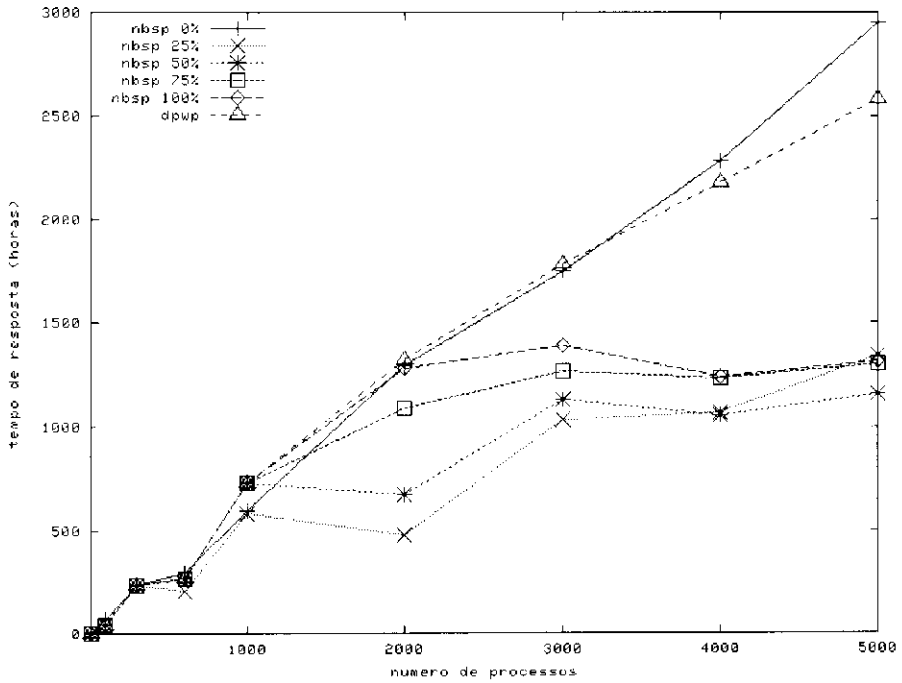

Figura 7.5: Poisson carga = 50.000 Kbytes; tempo = 60 segundos

com baixa quantidade de processos (600 processos).

Os resultados obtidos atestam as contribuições da política NBSP. Essas contribuições envolvem o aumento de desempenho de aplicações paralelas *Network-bound* e a utilização da rede de comunicação como um parâmetro nas decisões de escalonamento. Diferente das políticas apresentadas nos trabalhos prévios correlatos Mao et al. (1999), Keren e Barak (2003), que avaliam os custos de comunicação, porém não consideram o efeito da carga de ocupação da rede sobre as decisões de escalonamento.

Decisões de escalonamento em situações de carga na qual o volume de comunicação entre os processos é baixa, recomenda-se a adoção da DPWP, como pôde ser comprovado nos experimentos.

A NBSP pode ser implementada em um ambiente de escalonamento real seguindo as definições do projeto da DPWP (ARAÚJO et al., 1999), que está atualmente, incorporada no AMIGO (SOUZA, 2000). Para a implementação da NBSP no AMIGO é necessária a definição da interface entre o AMIGOD e a política NBSP. Isso é obtido implementando o código da política e incorporando-a nas estrutura do AMIGO (arquivo de configuração *policy.cfg*). Também é necessária a criação da tabela de cargas que armazena informações sobre os EPs. Recuperação do arquivo de configuração *benchmark.cfg* juntamente com os *benchmarks* adotados.

A NBSP apresenta deficiências relacionadas ao seu comportamento não preemptivo. Em situações de mudanças bruscas da carga de trabalho, ou até mesmo escalonamentos realizados de forma ineficiente não podem ser alterados até o término de execução da aplicação. Políticas de escalonamento que implementam técnicas de migração de processos não têm esse problema e

são exemplos de abordagens alternativas.

7.4 Considerações finais

Foram apresentados neste capítulo os mecanismos utilizados para execução dos experimentos e simulações. Tais experimentos possibilitam a parametrização do simulador adotado. Foram discutidos os parâmetros da simulação, bem como a carga de trabalho adotada, que segue o modelo de Feitelson (2001).

Os resultados obtidos através das simulações comprovam as contribuições da política NBSP. Essas contribuições envolvem o aumento de desempenho de aplicações *Network-bound* e a utilização da rede de comunicação como um parâmetro nas decisões de escalonamento.

O escalonamento direcionado às classes de aplicações constitui uma boa alternativa para definição de políticas de escalonamento específicas. Tais políticas avaliam os dispositivos principais do sistema distribuído mais exigidos pelas aplicações, tais como aplicações *Network-Bound*, que têm como gargalo de desempenho a rede. Nesse caso, a política NBSP é indicada para o escalonamento dessas aplicações de comunicação intensa, em situações de alta carga de ocupação dos canais de comunicação.

CAPÍTULO

8

Conclusões

8.1 Considerações finais

O escalonamento visa distribuir de maneira eficiente os processos de uma aplicação. Dentre os objetivos almejados pelo escalonamento destaca-se a minimização do tempo médio de resposta das aplicações que executam sobre o sistema distribuído. Com esse objetivo, esta dissertação investigou a influência da comunicação sobre o desempenho de aplicações paralelas *Network-Bound*.

É interessante analisar essa influência considerando o subsistema de comunicação como um parâmetro nas decisões de escalonamento das aplicações. Para avaliar esse parâmetro (consumo da banda de rede por processo) foi proposta uma extensão da política DPWP, constituindo a NBSP, que considera ainda a latência, a sobrecarga de comunicação e o volume de transmissão de mensagens.

Essa extensão utiliza a equação (6.2) para quantificar a ocupação da rede. Foi definida uma constante k, que delimita a utilização máxima de banda. Conforme o valor de retorno da equação (6.2) e da constante k, duas técnicas de escalonamento podem ser adotadas: escalonamento em grupo, ou por intermédio da rede de comunicação.

Observando a atuação dessa extensão e da constante k, buscou-se variá-la e analisá-la com o objetivo de demonstrar seu comportamento. Os resultados obtidos mostram que há benefícios na adoção da rede de comunicação como um parâmetro em decisões de escalonamento de processos.

Foram realizados experimentos comparativos entre a DPWP e a NBSP por meio de simulação. Tais experimentos confirmam que o uso da política NBSP diminui significativamente o tempo médio de resposta dos processos em situações de alta ocupação da rede de comunicação

(figuras 7.4 e 7.5). Da mesma maneira, em situações de carga do sistema onde a ocupação da rede é baixa (figura 7.3), recomenda-se um escalonamento supervisionado pela DPWP, que realiza balanceamento de carga e, conseqüentemente, aumenta o desempenho do sistema em relação ao tempo médio de resposta das aplicações.

Os resultados obtidos atestam na prática o tipo de escalonamento proposto neste trabalho, que define duas regras de decisões: escalonamento dos processos em grupo e por intermédio da rede. Tais regras são aplicadas à NBSP e nota-se que conforme a carga de comunicação imposta sobre o sistema ultrapassa uma certa taxa de utilização da largura de banda disponível ($\approx$ 166 Kbytes/s equação 7.6, figura 7.4), a NBSP começa a apresentar ganhos de desempenho em relação a DPWP original. Quanto maior a carga de comunicação mais perceptível é a diferença dos tempos médios de resposta (figura 7.5).

8.2 Relacionamento com os trabalhos desenvolvidos no grupo de pesquisa

O grupo de pesquisa Sistemas Distribuídos e Programação Concorrente do Instituto de Ciências Matemáticas e Computação da Universidade de São Paulo (ICMC-USP) tem direcionado esforços na linha de pesquisa relacionada ao escalonamento eficiente de aplicações paralelas sobre sistemas distribuídos. Desse esforço, foi definido e implementado um ambiente de escalonamento flexível e dinâmico, o AMIGO (SOUZA, 2000). Tal ambiente permite o escalonamento de aplicações paralelas em sistemas computacionais distribuídos, agregando políticas de escalonamento e mecanismos que podem ser configurados, dinamicamente, por meio de uma interface gráfica sem a necessidade de alteração no código da aplicação do usuário. Seguindo esse contexto, os trabalhos relacionados são:

- Especificação e implementação do AMIGO (SOUZA, 2000)
- Implementação da interface gráfica do AMIGO (Campos Jr., 2001)
- Integração do AMIGO com o PVM e implementação da política DPWP (ARAÚJO et al., 1999)
- Integração do AMIGO com uma implementação da especificação MPI e implementação da política DPWP (FIGUEIREDO, 2000)
- Integração do AMIGO com uma implementação da especificação CORBA (SANTOS, 2001)
- Monitoração do escalonamento em sistemas distribuídos (SOUZA, 2004)
- Índices de carga em sistemas distribuídos (doutorado em andamento)

- Aquisição do conhecimento de aplicações paralelas aplicado ao escalonamento de processos (doutorado em andamento)

Nesse sentido, a NBSP pode ser implementada em conjunto com o AMIGO (arquivo *police.cfg*, inserindo o código fonte da NBSP na estrutura de diretórios: AMIGO_PATH/polices /dpwp-e). Essa implementação deve seguir as especificações definidas no trabalho de Araújo et al. (1999). Outra recomendação, é agregar ao AMIGO o índice de carga que representa os parâmetros *overhead*, latência e volume transmitido de mensagens pela rede. Da mesma maneira, é necessário incorporar os *benchmarks* que avaliam a carga de ocupação da CPU e da rede de comunicação adequadamente na estrutura de configuração do AMIGO (arquivo *benchmarks.cfg*)

É possível utilizar o software de monitoração proposto no trabalho Souza (2004) para avaliar, em tempo de execução, o desempenho do escalonamento regido pela NBSP e, se for o caso, sugerir outro tipo de escalonamento.

O trabalho de Senger et al. (2004) determina os padrões de comunicação das aplicações. São definidos estados (operações sobre a cpu, dispositivos de entrada/saída e comunicação) pelos quais as aplicações passam no decorrer de sua execução. Por intermédio desse trabalho, e também através do AMIGO, que realiza um escalonamento por classe de aplicação, situações nas quais aplicações paralelas *Network-Bound* executam sobre um sistema com alta utilização de banda da rede, é altamente recomendado o uso da NBSP.

De modo geral, o projeto que tinha como objetivo atender a uma classe de aplicações paralelas particular (*Network-Bound*), obteve bons resultados, ressaltando a importância em se considerar diferentes classes de aplicações para construção e definição de políticas de escalonamento específicas.

8.3 Contribuições

As contribuições principais deste trabalho são: a definição de um modelo de comunicação, que caracteriza o volume de tráfego gerado sobre a rede de comunicação; a proposta de um escalonamento que considera a rede de comunicação como um parâmetro nas decisões de distribuição de processos e a extensão da política de escalonamento DPWP, que agrega parâmetro específico de comunicação sobre o ambiente distribuído.

O modelo define equações matemáticas que auxiliam na representação da carga de trabalho imposta sobre a CPU e à rede de comunicação. A carga sobre a CPU é decorrente de operações de empacotamento e desempacotamento das mensagens, já a carga sobre a rede, do volume transmitido de dados. Esse modelo também define uma constante k, que determina a máxima utilização da rede de comunicação. A partir dessa constante duas técnicas para distribuição dos processos podem ser adotadas: escalonamento em grupo ou por intermédio da rede de comunicação.

A partir dessas avaliações, que envolveram a definição do modelo, equações matemáticas e experimentos em um ambiente real, foi proposta uma nova abordagem para o escalonamento de processos de aplicações da classe *Network-Bound*. Essa proposta considera a rede de comunicação como um parâmetro nas decisões de escalonamento. Para comprovar o aumento de desempenho no uso dessa abordagem, a política DPWP foi estendida e implementada em um simulador adotado. Tal simulador também foi estendido, agregando as características de comunicação tanto dos processos, quanto dos elementos de processamento do sistema distribuído.

8.4 Trabalhos futuros

As contribuições deste trabalho motivam como trabalhos futuros a implementação do modelo em um ambiente distribuído real com objetivos de atestar os resultados de simulação. Para tal implementação é necessário investigar, localizar e verificar aplicações paralelas reais. Em alguns trabalhos tais como Bailey e Frederickson (1991), Cremonesi e Gennaro (2002), Rosti et al. (2002), Vetter e Mueller (2003), utilizam-se aplicações fornecidas pelos pacotes NAS e ParkBench. Além disso, é necessário escolher em qual software de escalonamento será implementada a política NBSP, dentre os softwares de escalonamento discutidos na seção 4.4, recomenda-se a implementação no AMIGO, por sua flexibilidade, dinamismo e por já incorporar a política DPWP.

O projeto e implementação de outras políticas de escalonamento que considerem outras classes de aplicações, tais como *Disk-Bound*, *Memory-Bound*, *CPU-Bound*. Este trabalho atestou a abordagem que considera o escalonamento orientado à classe de aplicações. A classe específica *Network-Bound* foi tratada nesta dissertação obtendo bons resultados e espera-se que o mesmo ocorra para outras classes de aplicações.

Avaliar as decisões de escalonamento realizadas pela NBSP aplicando técnicas de monitoração, por exemplo utilizando o software PSMS (SOUZA, 2004). Essa proposta se baseia em objetivos previamente definidos. Assumindo que a política NBSP visa o escalonamento de aplicações de comunicação intensiva, os objetivos esperados podem ser: reduzir a carga de tráfego gerado sobre a rede, diminuir a espera por sincronização entre os processos e diminuir o tempo de resposta da aplicação. É possível incorporar no PSMS tais objetivos e avaliar o comportamento de aplicações *Network-Bound* durante a execução. Essa avaliação pode ajudar na adaptação do algoritmo, na identificação e correção de deficiências e até mesmo na proposta de um novo algoritmo.

Utilizar técnicas de predição do comportamento dos processos para a tomada de decisões de escalonamento mais específicas. Essas técnicas são baseadas em conceitos de inteligência artificial que ajudam a predizer um comportamento futuro. De posse dessas informações é possível desenvolver políticas de escalonamento ainda mais refinadas e, provavelmente, que

Tabela 8.1: Matriz de comunicação

Processos	P1	P2	P3	P4
P1	0	1	0	0
P2	0	0	0	0
P3	0	1	0	0
P4	0	0	0	0

forneçam maior desempenho às aplicações. Para aplicações da classe *Network-Bound* uma informação valiosa seria uma matriz de comunicação contendo quais as relações de interação entre os processos. Uma aplicação que tem quatro processos e a relação de interação dada pela matriz de comunicação apresentada na tabela 8.1, pode ter uma política que privilegie o escalonamento em grupo dos processos que comunicam entre si e distribua através da rede os processos que não comunicam entre si. Dessa maneira, os processos da aplicação cuja interação entre os processos é dada pela tabela 8.1, podem ser distribuídos da seguinte maneira: **P1**, **P2** e **P3** no mesmo elemento de processamento e **P4** em outro elemento de processamento.

Referências

AIDA, K. Effect of job size characteristics on job scheduling performance. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer Verlag, 2000. (Lecture Notes in Computer Science, v. 1911). p. 1–17.

ALMASI, G. S.; GOTTLIEB, A. *Highly Parallel Computing*. 2. ed. Redwood City: The Benjamin/Cummings Publishing Company, Inc, 1994. 689 p.

ANASTASIADIS, S. V.; SEVCIK, K. C. Parallel application scheduling on networks of workstations. *J. Parallel Distrib. Comput.*, Academic Press, Inc., v. 43, n. 2, p. 109–124, 1997.

ANDERSON, T. E. et al. A case for now (networks of workstations). *IEEE Micro*, IEEE Computer Society Press, v. 15, n. 1, p. 54–64, 1995.

ANGLANO, C. A comparative evaluation of implicit coscheduling strategies for networks of workstations. In: *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*. [S.l.]: IEEE Computer Society, 2000. p. 221–228.

ARAÚJO, A. P. F. et al. DPWP: A new load balancing algorithm. In: *5th International Conference on Information Systems Analysis and Synthesis - ISAS'99*. Orlando, U.S.A.: [s.n.], 1999.

ARPACI, R. H. et al. The interaction of parallel and sequential workloads on a network of workstations. In: *Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. Ottawa, Ontario, Canada: ACM Press, 1995. p. 267–278.

ASPNES, J. et al. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, ACM Press, v. 44, n. 3, p. 486–504, 1997.

AZAR, Y.; EPSTEIN, L. On two dimensional packing. *J. Algorithms*, Academic Press, Inc., v. 25, n. 2, p. 290–310, 1997.

BAILEY, D. H.; FREDERICKSON, P. O. Performance results for two of the nas parallel benchmarks. In: *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*. Albuquerque, New Mexico, United States: ACM Press, 1991. p. 166–173.

BAKER, M.; BUYYA, R. Cluster computing: the commodity supercomputer. *Software Practice and Experience*, v. 29, n. 6, p. 551–576, 1999.

BANKS, J. (Ed.). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. New York: John Wiley and Sons, 1998. 864 p.

BELCASTRO, V. et al. An overview of the distributed query system dqs. In: *Proceedings of the International Conference on Extending Database Technology*. [S.l.]: Springer-Verlag, 1988. p. 170–189.

BERMAN, F.; WOLSKI, R. Scheduling from the perspective of the application. In: *Proceedings of the High Performance Distributed Computing (HPDC '96)*. [S.l.]: IEEE Computer Society, 1996. p. 100.

BOVET, D.; CESATI, M.; ORAM, A. *Understanding the Linux Kernel*. 2. ed. [S.l.]: O'Reilly & Associates, Inc., 2002. 816 p.

Campos Jr., A. de. *Desenvolvimento de uma interface gráfica para um ambiente de escalonamento*. Dissertação (Mestrado) – Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2001.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, v. 14, n. 2, p. 141–154, Fevereiro. 1988.

CHASE, J. S. et al. Network I/O with trapeze. In: *Proceedings of 1999 Hot Interconnects Symposium*. [S.l.: s.n.], 1999.

CHIANG, C.-C. A distributed object computing architecture for leveraging software reengineering systems. In: *Proceedings of the 2001 ACM symposium on Applied computing*. Las Vegas, Nevada, United States: ACM Press, 2001. p. 653–657.

CHIOLA, G.; CIACCIO, G. Efficient parallel processing on low-cost clusters with gamma active ports. *Parallel Comput.*, Elsevier Science Publishers B. V., v. 26, n. 2-3, p. 333–354, 2000.

CHODNEKAR, S. et al. Towards a communication characterization methodology for parallel applications. In: *Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture (HPCA '97)*. [S.l.]: IEEE Computer Society, 1997. p. 310–321.

CHOE, T.; PARK, C. A task duplication based scheduling algorithm with optimality condition in heterogeneous systems. In: *International Conference on Parallel Processing Workshops (ICPPW'02)*. [S.l.]: IEEE Computer Society, 2002. p. 531–536.

Condor Team. *Condor Version 6.1.17 Manual*. University of Wisconsin-Madison: [s.n.], 2000. Disponível em: <http://www.cs.wisc.edu/condor/manual/v6.1/ref.html>. Acesso em: 19 nov. 2004.

CORBALAN, J.; MARTORELL, X.; LABARTA, J. Improving gang scheduling through job performance analysis and malleability. In: *Proceedings of the 15th international conference on Supercomputing*. [S.l.]: ACM Press, 2001. p. 303–311.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems*: Concepts and design. 3. ed. Harlow: Addison Wesley, 2001. 771 p.

CREMONESI, P.; GENNARO, C. Integrated performance models for spmd applications and mimd architectures. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, v. 13, n. 7, p. 745–757, 2002.

CULLER, D. et al. Logp: towards a realistic model of parallel computation. In: *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*. San Diego, California, United States: ACM Press, 1993. p. 1–12.

DARBHA, S.; AGRAWAL, D. P. Optimal scheduling algorithm for distributed-memory machines. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, v. 9, n. 1, p. 87–95, 1998.

ESHAGHIAN, M. M.; WU, Y.-C. Resource estimation for heterogeneous computing. *Future Gener. Comput. Syst.*, Elsevier Science Publishers B. V., v. 12, n. 6, p. 505–520, 1997.

FEITELSON, D. G. Packing schemes for gang scheduling. In: *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer-Verlag, 1996. p. 89–110.

FEITELSON, D. G. Metrics for parallel job scheduling and their convergence. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer, 2001. p. 188–205. Lect. Notes Comput. Sci. vol. 2221.

FEITELSON, D. G.; RUDOLPH, L. Metrics and benchmarking for parallel job scheduling. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer, 1998. v. 1459, p. 1–24.

FERRARI, D.; ZHOU, S. An empirical investigation of load indices for load balancing applications. In: *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation*. [S.l.]: North-Holland, 1988. p. 515–528.

FIGUEIREDO, A. T. *Interface AMIGO-MPI: Uma abordagem flexível e dinâmica para o escalonamento de processos*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2000.

FLYNN, M. J.; RUDD, K. W. Parallel architectures. *ACM Comput. Surv.*, ACM Press, v. 28, n. 1, p. 67–70, 1996.

FRANCÊS, C. R. L. *Statecharts Estocásticos e Queuing Statecharts: Novas Abordagens para Avaliação de Desempenho Baseadas em Especificação Statecharts*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, Agosto 2001.

FRANCÊS, C. R. L. et al. *Modelagem e Especificação Utilizando Redes de Petri e Statecharts*. São Carlos, 2000.

GEIST, A. et al. *PVM (Parallel virtual machine): a users' guide and tutorial for networked parallel computing*. Cambrige, Massachusetts: MIT Press, 1994. 279 p.

GRAMA, A. et al. *Introduction to Parallel Computing*. 2. ed. [S.l.]: Addison Weslesy, 2003. 656 p.

HARCHOL-BALTER, M.; DOWNEY, A. B. Exploiting process lifetime distributions for dynamic load balancing. In: *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. Philadelphia, Pennsylvania, United States: ACM Press, 1996. p. 13–24.

HAREL, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., v. 8, n. 3, p. 231–274, 1987.

HENNING, J. L. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, IEEE Computer Society Press, v. 33, n. 7, p. 28–35, 2000.

HWANG, K.; XU, Z. *Scalable Parallel Computing*: Technology, architecture, programming. 1. ed. New York: McGraw-Hill, 1998. 802 p.

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. New York: John Wiley and Sons, 1991. 685 p.

J.Y.COLIN; P.CHRÉTIENNE. C.P.M. Scheduling with small communication delays and task duplication. *Operations Research*, v. 39, p. 680–684, 1991.

KAPLAN, J. A.; NELSON, M. L. *A Comparison of Queueing, Cluster and Distributed Computing Systems*. [S.l.], 1994.

KEAHEY, K.; GANNON, D. Pardis: A parallel approach to corba. In: *Proceedings of the 6th International Symposium on High Performance Distributed Computing (HPDC '97)*. [S.l.]: IEEE Computer Society, 1997. p. 31–39.

KEREN, A.; BARAK, A. Opportunity cost algorithms for reduction of i/o and interprocess communication overhead in a computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, v. 14, n. 1, p. 39–50, Janeiro 2003.

KERRIGAN, T. C. *TSCP Benchmark Scores*. 2004. Disponível em: <http://home.comcast.net-/~tckerrigan/bench.html>. Acesso em: 17 jul. 2004.

KLEINROCK, L. (Ed.). *Queueing Systems: Computer Applications*. New York: John Wiley and Sons, 1976.

LAM Team. *LAM/MPI Parallel Computing*. 2004. Disponível em: <http://www.lam-mpi.org/>. Acesso em: 09 nov. 2004.

LAVENBERG, S. S. *Computer Performance Modeling Handbook*. Orlando, FL, USA: Academic Press, Inc., 1983. 399 p.

LIU, C. et al. Design and evaluation of a resource selection framework for grid applications. In: *Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02)*. [S.l.]: IEEE Computer Society, 2002. p. 63–72.

MACHE, J.; LO, V.; GARG, S. Job scheduling that minimizes network contention due to both communication and I/O. In: *Proceedings of the 14th International Symposium on Parallel and Distributed Processing*. Cancun, Mexico: IEEE Computer Society, 2000. p. 457–464.

MACHE, J. et al. The impact of spatial layout of jobs on parallel i/o performance. In: *Proceedings of the sixth workshop on I/O in parallel and distributed systems*. Atlanta, Georgia, United States: ACM Press, 1999. p. 45–56.

MAO, W.; CHEN, J.; III, W. W. On-line algorithms for a parallel job scheduling problem. In: *in Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*. [S.l.: s.n.], 1999. p. 753–757.

MARSAN, M. A.; BOBBIO, A.; DONATELLI, S. Petri nets in performance analysis: An introduction. In: *Lectures Notes in Computer Science*. [S.l.]: Springer-Verlag, 1998. v. 1491, p. 211–256.

MELLO, R. F.; SENGER, L. J. A new migration model based on the evaluation of processes load and lifetime on heterogeneous computing environments. In: *16th Symposium on Computer Architecture and High Performance Computing*. Foz do Iguaçu, PR - Brasil: IEEE Computer Society, 2004. p. 222–227.

MOWBRAY, T. J.; MALVEAU, R. C. *CORBA Design Patterns*. [S.l.]: John Wiley & Sons, Inc., 1997. 333 p.

MOWBRAY, W. A. R. T. J. *Inside CORBA : distributed object standards and applications*. Reading, Mass.: Addison-Wesley, 1997. 376 p.

MPI Forum. *MPI-2: Extensions to the Message Passing Interface*. 2004. Disponível em: <http://www.mpi-forum.org/>. Acesso em: 09 nov. 2004.

MPI Forum. *MPI-2 Journal of Development*. 2004. Disponível em: <http://www.mpi-forum-.org/>. Acesso em: 09 nov. 2004.

MU'ALEM, A. W.; FEITELSON, D. G. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel & Distributed Syst.*, v. 12, n. 6, p. 529–543, jun. 2001.

MULLENDER, S. (Ed.). *Distributed Systems*. 2. ed. New York: Addison-Wesley Publishing Company, 1993. 601 p. (ACM Press Frontier Series).

NAVAUX, P. O. A. Introdução ao processamento paralelo. *RBC – Revista Brasileira de Computação*, SBC Sociedade Brasileira de Computação, v. 5, n. 2, p. 31–43, 1989.

NEARY, M. O.; CAPPELLO, P. Internet-based tsp computation with javelin++. In: *Proceedings of the 2000 International Workshop on Parallel Processing*. [S.l.]: IEEE Computer Society, 2000. p. 137–144.

NI, L. M.; XU, C.; GENDREAU, T. B. A distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engineering*, IEEE Press, v. 11, n. 10, p. 1153–1161, 1985.

OH, H.; HA, S. A static scheduling heuristic for heterogeneous processors. In: *Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II*. [S.l.]: Springer-Verlag, 1996. p. 573–577.

ORLANDI, R. C. G. S. *Ferramentas para Análise de Desempenho de Sistemas Computacionais Distribuídos*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, Março 1995.

PARK, C. I.; CHOE, T. Y. An optimal scheduling algorithm based on task duplication. *IEEE Trans. Comput.*, IEEE Computer Society, v. 51, n. 4, p. 444–448, 2002.

Platform Computing Corporation. *LSF Administrator's Guide*. junho 2000. Disponível em: <http://help.aei-potsdam.mpg.de/sc/lsf/admin_4.0.1/>. Acesso em: 19 nov. 2004.

PRIOL, T.; RENÉ, C. Cobra: A corba-compliant programming environment for high-performance computing. In: *Proceedings of the 4th International Euro-Par Conference on Parallel Processing*. [S.l.]: Springer-Verlag, 1998. p. 1114–1122.

QUINN, M. J. *Parallel Computing: Theory and Practice*. 2. ed. New York: McGraw Hill, 1994. 446 p.

RADULESCU, A.; GEMUND, A. van; LIN, H. LLB: A fast and effective scheduling algorithm for distributed-memory systems. In: *13th International and 10th Symposium on Parallel and Distributed Processing - IPPS/SPDP*. [S.l.: s.n.], 1999. p. 525–530.

RADULESCU, A.; Van Gemund, A. J. C. GLB: A low-cost scheduling algorithm for distributed-memory architectures. In: *Proceedings of the Fifth International Conference on High Performance Computing*. [S.l.]: IEEE Computer Society, 1998. p. 294–301.

RANAWEERA, S.; AGRAWAL, D. P. A task duplication based scheduling algorithm for heterogeneous systems. In: *Parallel and Distributed Processing Symposium - IPDPS 2000*. [S.l.: s.n.], 2000. p. 445–450.

ROSTI, E. et al. Models of parallel applications with large computation and i/o requirements. *IEEE Trans. Softw. Eng.*, IEEE Press, v. 28, n. 3, p. 286–307, 2002.

RYOU, J.; WONG, J. S. K. A task migration algorithm for load balancing in a distributed system. In: *XXII Annual Hawaii International Conference on System Sciences*. [S.l.: s.n.], 1989. p. 1041–1048.

SANTANA, R. H. C. et al. *Técnicas para Avaliação de Desempenho de Sistemas Computacionais*. São Carlos, SP – Brasil, 1994.

SANTOS, R. R. *Escalonamento de aplicações paralelas: Interface AMIGO–CORBA*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2001.

SARKAR, V. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*. Tese (Doutorado) — MIT, 1989.

SCHEFLER, W. C. *Statistics: concepts and applications*. [S.l.]: Benjamin-Cummings Publishing Co., Inc., 1988. 590 p.

SCHNOR, B. et al. Scheduling of parallel applications on heterogeneous workstation clusters. In: YETONGNON, I. K.; (ED.), S. H. (Ed.). *Proceedings of PDCS'96 – ISCA 9th International Conference on Parallel and Distributed Computing Systems*. Dijon, France: [s.n.], 1996. p. 330–337.

SENGER, L. J.; SANTANA, M. J.; SANTANA, R. H. C. Uma nova abordagem para a aquisição de conhecimento sobre o comportamento de aplicações paralelas na utilização de recursos. In: SAKUDE, M. T. S.; CESAR, C. A. C. (Ed.). *Workshop de Computação – WorkComp 2002*. São José dos Campos: ITA/CTA, 2002. p. 79–85.

SENGER, L. J.; SANTANA, M. J.; SANTANA, R. H. C. A new approach fo acquiring knowledge of resource usage in parallel applications. In: *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2003)*. [S.l.: s.n.], 2003. p. 607–614.

SENGER, L. J.; SANTANA, M. J.; SANTANA, R. H. C. Using runtime measurements and historical traces for acquiring knowledge in parallel applications. In: BUBAK, M. et al. (Ed.). *International Conference on Computational Science (ICCS 2004)*. [S.l.]: Springer, 2004, (Lecture Notes in Computer Science, v. 3036).

SHIRAZI, B. A.; HURSON, A. R.; KAVI, K. M. (Ed.). *Scheduling and Load Balancing in Parallel and Distributed Systems*. [S.l.]: Wiley-IEEE Computer Society Press, 1995. 520 p.

SHIVARATRI, N. G.; KRUEGER, P.; SINGHAL, M. Load distributing for locally distributed systems. *Computer*, IEEE Computer Society Press, v. 25, n. 12, p. 33–44, 1992.

Silicon Graphics, Inc.; Cray Research, Inc. *Introducing NQE*. abr. 1998. Número da publicação: 2153 3.3. Disponível em: <http://www.cesup.ufrgs.br/craydoc/manuals/2153_3.3/2153_3-.3-manual.pdf>. Acesso em: 19 nov. 2004.

SILVA, F. A. B. D.; SCHERSON, I. D. Improving Parallel Job Scheduling Using Runtime Measurements. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer, 2000. p. 18–38. Lect. Notes Comput. Sci. vol. 1911.

SKJELLUM, A. et al. *Extending the message passing interface (MPI)*. Mississippi State University, 1994.

SNIR, M. et al. *MPI: The Complete Reference*. Cambridge, Massachusetts: MIT Press, 1998.

SOUZA, M. A. de. *Avaliação das Rotinas de Comunicação Ponto-a-Ponto do MPI*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, Junho 1997.

SOUZA, M. A. de. *Uma abordagem para a avaliação do escalonamento de processos em sistemas distribuídos baseada em monitoração*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, Junho 2004.

SOUZA, P. S. L. *AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos*. Tese (Doutorado) — IFSC-USP, Junho 2000.

SQUILLANTE, M. S. et al. Modeling and analysis of dynamic coscheduling in parallel and distributed environments. In: *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. Marina Del Rey, California: ACM Press, 2002. p. 43–54.

SUN. *Sun Grid Engine: White paper*. 2003. Disponível em: <http://wwws.sun.com/software/gridware/sgeee53/wp-sgeee/index.html>. Acesso em: 19 nov. 2004.

SYSTEMS, V. *PBS: Portable Batch Systems*. 2004. Disponível em: <http://pbs.mrj.com>. Acesso em: 19 nov. 2004.

TALBY, D.; FEITELSON, D. G.; RAVEH, A. Comparing logs and models of parallel workloads using the co-plot method. In: FEITELSON, D. G.; RUDOLPH, L. (Ed.). *Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer, 1999. p. 43–66. Lect. Notes Comput. Sci. vol. 1659.

TANENBAUM, A. S. *Modern Operating Systems*. 2. ed. Upper Saddle River, N.J.: Prentice-Hall, Inc., 2001. 951 p.

TANENBAUM, A. S.; STEEN, M. van. *Distributed Systems*: Principles and paradigms. Upper Saddle River, NJ: Prentice Hall, 2002. 803 p.

TINETTI, F. G.; BARBIERI, A. An efficient implementation for broadcasting data in parallel applications over ethernet clusters. In: *17 th International Conference on Advanced Information Networking and Applications (AINA'03)*. [S.l.]: IEEE Computer Society, 2003. p. 593–597.

TOPCUOGLU, H.; HARIRI, S.; WU, M. Task scheduling algorithms for heterogeneous processors. In: *Proceedings of the Eighth Heterogeneous Computing Workshop*. [S.l.]: IEEE Computer Society, 1999. p. 3–14.

VETTER, J. S.; MUELLER, F. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. *J. Parallel Distrib. Comput.*, Academic Press, Inc., v. 63, n. 9, p. 853–865, 2003.

VINOSKI, S. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, IEEE Press, v. 35, n. 2, p. 46–55, February 1997.

XU, C. wei; HE, B. PCB: a distributed computing system in corba. *J. Parallel Distrib. Comput.*, Academic Press, Inc., v. 60, n. 10, p. 1293–1310, 2000.

YANG, T. *Scheduling and Code Generation for Parallel Architectures*. Tese (Doutorado) — Dep. of CS, Rutgers Univ., May 1993.

ZHANG, Y. et al. A simulation-based study of scheduling mechanisms for a dynamic cluster environment. In: *Proceedings of the 14th international conference on Supercomputing*. Santa Fe, New Mexico, United States: ACM Press, 2000. p. 100–109.

ZHOU, B. B.; BRENT, R. P. On the development of an efficient coscheduling system. In: *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer-Verlag, 2001. p. 103–115.

ZHOU, B. B.; WALSH, D.; BRENT, R. P. Resource allocation schemes for gang scheduling. In: *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. [S.l.]: Springer-Verlag, 2000. p. 74–86.

ZHOU, M.; VENKATESH, K. *Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach*. [S.l.]: World Scientific Pub Co., 1999.