UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Escalonamento de Processos: Uma Contribuição para a Convergência da Área

Paulo Sergio Lopes de Souza
Marcos José Santana
Regina Helena Carlucci Santana

Nº 46

NOTAS

São Carlos - SP

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2577

Escalonamento de Processos:
Uma Contribuição para a Convergência da Área

Paulo Sergio Lopes de Souza
Marcos José Santana
Regina Helena Carlucci Santana

Nº 46

NOTAS

Série Computação

São Carlos – SP
Mai./2000

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística

Escalonamento de Processos:
Uma Contribuição para a Convergência da Área

Paulo Sergio Lopes de Souza *
Departamento de Informática
Setor de Ciências Agrárias e de Tecnologia
Universidade Estadual de Ponta Grossa – UEPG
Caixa Postal 992/3
84010-220 – Ponta Grossa - PR

Marcos José Santana
Regina Helena Carlucci Santana
Departamento de Ciências de Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – Campus de São Carlos
Caixa Postal 668
13560-970 – São Carlos - SP

*Programa de Doutorado do Instituto de Física de São Carlos – Universidade de São Paulo

maio/2000

Os autores agradecem as agências CAPES, FAPESP e CNPq pelo auxílio ao Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC/USP.

Sumário

LISTA DE FIGURAS ..... **II**

LISTA DE TABELAS ..... **III**

RESUMO ..... **IV**

ABSTRACT ..... **V**

1. INTRODUÇÃO **1**

2. ESCALONAMENTO DE PROCESSOS **3**

2.1. Escalonamento Local 4
2.2. Escalonamento Global 5
2.3. Escalonamento em Plataformas Computacionais Distribuídas e Heterogêneas 7
2.4. Problemas com a Terminologia da Área de Escalonamento de Processos 8
2.4.1. Divergências na Terminologia *9*
2.4.2. Razões para as Divergências na Terminologia *18*
2.5. Taxonomias da Área de Escalonamento de Processos 22
2.5.1. Taxonomia de Casavant; Kuhl (1988) *23*
2.5.2. Taxonomia de Wang; Morris (1985) *30*
2.5.3. Taxonomia de Baumgartner; Wah (1991) *31*
2.5.4. Taxonomia de Lüling et al. (1993) *36*
2.5.5. Taxonomia de Xu; Lau (1997) *38*
2.5.6. Taxonomia usada por Quinn (1994) *42*
2.5.7. Taxonomia de Feitelson; Rudolph (1995) *44*
2.5.8. Taxonomia de Milojicic et al. (1993) *52*
2.5.9. Taxonomia usada por Tanenbaum (1995) *53*
2.6. Uma Convergência para as Taxonomias 54
2.6.1. Taxonomias baseadas nos processos x baseadas na topologia *56*
2.6.2. Taxonomias baseadas nos processos x baseadas nos processadores *58*
2.7. Componentes de um Algoritmo de Escalonamento 60
2.8. Índice de Carga 64
2.9. Desempenho Alcançado com o Algoritmo de Escalonamento 67
2.9.1. Objetivos e Métricas de Desempenho *67*
2.9.2. Fatores que Influenciam o Desempenho *69*
2.10. Escalonamento: a Teoria e a Prática 72
2.10.1. Fatores que Afastam a Teoria da Prática *72*
2.10.2. Fatores que Convergem a Teoria e a Prática *73*

3. CONCLUSÕES **75**

REFERÊNCIAS BIBLIOGRÁFICAS **77**

APÊNDICE 1 – RELAÇÃO DE NOVAS PROPOSTAS DE ALGORITMOS DE ESCALONAMENTO **91**

Lista de Figuras

Figura 1 - O escalonamento segundo Casavant; Kuhl (1988). 6
Figura 2 - O escalonamento segundo Baumgartner; Wah (1991). 6
Figura 3 – Taxonomia hierárquica de Casavant; Kuhl (1988). 24
Figura 4 – Um sistema distribuído local segundo Wang; Morris (1985). 31
Figura 5 – Relação hierárquica das políticas de escalonamento proposta por Baumgartner; Wah (1991). 33
Figura 6 – Taxonomia proposta por Lüling et al. (1993). 37
Figura 7 – Taxonomia proposta por Xu; Lau (1997). 40
Figura 8 – Exemplo de um modelo iterativo gradiente com dezesseis processadores [Lüling et al. (1993)]. 41
Figura 9 – Extensão da taxonomia de Casavant; Kuhl (1988) por Muniz (1994). 58

Lista de Tabelas

Tabela 1 – Revisão das taxonomias de políticas de escalonamento [Baumgartner; Wah (1991)]. 33

Tabela 2 – Taxonomia ESR proposta por Baumgartner; Wah (1991). 35

Tabela 3 – Relação descrevendo a taxonomia das políticas space sharing proposta por Feitelson;Rudolph (1995). 46

Tabela 4 – Classificação dos tipos de aplicações paralelas, baseada na especificação por recursos e também na habilidade de se adaptar a diferentes números de processadores [Feitelson; Rudolph (1996)]. 47

Tabela 5 – Taxonomia proposta por Milojicic et al. (1993) 52

Resumo

O escalonamento de processos é um dos principais itens que influenciam o desempenho de uma plataforma computacional distribuída (composta por vários elementos de processamento). Independentemente se essa plataforma é utilizada para computação paralela ou para sistemas distribuídos, há várias maneiras de se distribuir processos aos processadores, cada uma apresentando vantagens e desvantagens.

Para determinar corretamente qual política de escalonamento empregar, o usuário deve ser capaz de comparar diferentes propostas e escolher aquela que lhe é mais apropriada. No entanto, essa escolha não é trivial porque a literatura não apresenta uma terminologia consistente e também porque as taxonomias divergem na abordagem empregada para classificar as diferentes políticas de escalonamento.

Este trabalho tem por objetivo central apresentar a área de escalonamento de processos de uma maneira abrangente, realizando uma análise crítica da bibliografia disponível. As principais divergências encontradas na terminologia são apontadas, destacando-se os motivos destas. As taxonomias apresentadas são discutidas salientando-se suas vantagens e desvantagens, além de destacar alguns itens que limitam a abrangência e dificultam uma convergência dessas taxonomias.

Mais do que simplesmente apontar divergências, este trabalho propõe pontos de convergência. Esses pontos podem ser desde a simples questão da terminologia até questões sobre as taxonomias existentes.

Também são destacadas outras questões relativas à área, como por exemplo: componentes de um algoritmo de escalonamento, índices de carga e métricas para avaliação de desempenho do escalonamento.

Espera-se que pesquisadores iniciantes em escalonamento de processos encontrem neste texto, um material que lhes forneça um contexto geral sobre o assunto e, dessa forma, contribua para facilitar a compreensão de outros trabalhos.

Abstract

This work presents an overview of the process scheduling theory together with a critical analysis of the available literature. The main divergences found in the terminology are pointed, with emphasis on the reasons behind them. The taxonomies presented are discussed, pointing out their advantages and disadvantages. Some of the points that limit the taxonomies scope and difficult a convergence of them are also discussed.

More than simply pointing out divergences, this work proposes some convergence points for the taxonomies. This convergence ranges from the question of terminology to questions about taxonomies already proposed, which present significant differences.

Some other questions in this research area are also discussed, such as: components of the scheduling algorithm, the load index and metrics for the performance evaluation of the scheduling, etc.

This work makes possible for researchers with little experience on process scheduling to find a precious material with a general scope about the subject, making the understanding of other works easier.

1.Introdução

A literatura sobre escalonamento de processos é abundante. Facilmente são encontrados centenas de artigos contendo informações que contribuem para o aprimoramento da área. No entanto, ao estudar esses artigos percebe-se que eles seguem caminhos distintos, tornando difícil a convergência e o relacionamento do conhecimento adquirido.

Embora a qualidade e a importância dos trabalhos sejam indiscutíveis, a literatura não fornece uma visão abrangente da área, a ponto de permitir a comparação apropriada de certas propostas.

Esses problemas não acontecem por falta de taxonomias. A área de escalonamento possui várias delas, porém, as mesmas utilizam muitas vezes terminologias distintas e/ou conflitantes. Esse problema na terminologia não é exclusividade das propostas de novas taxonomias, ele acontece com freqüência em vários trabalhos, principalmente naqueles que propõem novos algoritmos de escalonamento.

Este trabalho tem por objetivo diminuir os problemas encontrados na literatura, fazendo um levantamento bibliográfico crítico sobre a área. A intenção não é apenas descrever o que já existe, mas também fornecer uma visão abrangente do assunto, destacando as divergências e como convergir para um entendimento apropriado. Com base na leitura deste trabalho, espera-se que pesquisadores iniciantes possam formar um contexto geral sobre o assunto, servindo de base para as suas futuras pesquisas na área.

O escalonamento é abordado como atividade principal e não apenas como parte integrante de um sistema operacional distribuído ou ferramenta de escalonamento exclusiva para a computação paralela. A meta é descrever o escalonamento de maneira independente da plataforma computacional distribuída (hardware e softwares básicos), dos seus objetivos e das suas implementações.

Para fornecer uma visão ampla e convergente sobre escalonamento de processos, são abordados alguns assuntos principais que compõem o conhecimento da área. Dentre esses assuntos, os problemas com a terminologia e as taxonomias existentes são discutidos com maior ênfase, por serem mais representativos aos objetivos propostos neste trabalho.

As divergências encontradas na terminologia da área são apresentadas, seguidas dos motivos que levam a essas divergências. Ao relacionar as divergências encontradas na terminologia, não houve a intenção de descrever taxonomias, e sim demonstrar através da literatura disponível, termos usados de maneira indiscriminada como sinônimos ou então atividades idênticas denominadas com termos distintos.

As principais taxonomias da área são apresentadas salientando-se suas vantagens e desvantagens. É descrito que muitas não apresentam pontos de convergência, dificultando a comparação adequada das mesmas. Apresentadas as taxonomias, é descrita uma proposta de convergência, não com o objetivo de criar uma nova taxonomia, e sim fornecer uma relação entre elas que possibilite comparar com eficiência dois algoritmos de escalonamento classificados sob duas taxonomias divergentes.

O texto apresentado neste trabalho é específico sobre escalonamento de processos. Assume-se que o leitor já possui conhecimentos ou no mínimo noções básicas sobre temas como sistemas distribuídos, computação paralela e redes de computadores.

2.Escalonamento de Processos

O escalonamento é estudado e praticado em áreas do conhecimento onde se deseja atribuir ou distribuir serviços para recursos, os quais irão realizar algum processamento sobre esses serviços. Bons exemplos dessas áreas são a engenharia de produção e a computação.

A definição da atividade de escalonar é dada no dicionário como sinônimo de dispor, pôr, colocar ou distribuir ordenadamente [Ferreira (1986)]. Essa definição aplica-se perfeitamente à computação, onde para executar um serviço (como por exemplo um processo) há a necessidade de colocá-lo para execução em um recurso de processamento (ou processador[1]) de maneira apropriada.

Para se iniciar uma discussão sobre escalonamento de processos é importante destacar a diferença existente entre o escalonamento feito em um único processador e o escalonamento feito em vários processadores. No primeiro caso, também chamado de escalonamento local [Casavant; Kuhl (1988)], a atribuição dos processos ao único processador disponível considera características como por exemplo: mais de um usuário utilizando o mesmo processador (multiusuário), processos concorrentes (multitarefa), tipos de aplicações e o algoritmo de escalonamento implementado pelo sistema operacional. No segundo caso, também denominado de escalonamento global [Casavant; Kuhl (1988)], devem ser considerados os vários processadores que compõem a plataforma[2] computacional distribuída. Além das características do escalonamento local, o escalonamento global deve preocupar-se em determinar qual(is) o(s) melhor(es) processador(es) para executar o(s) processo(s), considerando sempre os objetivos e as características da plataforma como um todo.

---

[1] Neste trabalho o termo "processador" é utilizado como sinônimo de elemento de processamento, recurso de processamento, nó, *host* e computador. O termo computador, quando qualificado como "computador paralelo" designa um equipamento com vários processadores, agrupados com o objetivo de executar aplicações paralelas. Um nó, segundo alguns autores, também pode conter vários processadores.

[2] Neste trabalho "plataforma" é usada como um termo genérico para se referir à união do hardware, software básico, topologia e demais características que compõe o ambiente computacional que se está utilizando.

2.1. Escalonamento Local

Tanenbaum (1992) descreve que o escalonamento local é realizado por um escalonador (*scheduler*), o qual é implementado por um algoritmo denominado algoritmo de escalonamento. Com as atuais plataformas monoprocessadoras, multiusuárias e multitarefas, esse algoritmo de escalonamento deve preocupar-se com vários fatores que não existiam nas antigas plataformas *batch* e monousuárias, onde o processador deveria simplesmente executar o próximo processo, formado através de cartões perfurados e posteriormente apresentar os resultados obtidos.

Além de permitir que os processos sejam executados no processador, um algoritmo de escalonamento local deve possuir alguns objetivos para poder ser considerado um bom algoritmo de escalonamento. Embora algumas vezes contraditórios, são exemplos de objetivos: (1) imparcialidade: ter certeza que todos os processos terão acesso adequado ao processador; (2) eficiência: manter o processador 100% do tempo ocupado; (3) tempo de resposta: diminuir o tempo de resposta para os usuários interativos; (4) *turnaround time*: diminuir o tempo que os usuários de aplicações *batch* devem esperar pela saída dos resultados, desde a submissão; e (5) *throughput* do sistema: aumentar o número de processos executados por unidade de tempo.

O algoritmo de escalonamento em plataformas multiusuárias/multitarefas não deve permitir que um processo monopolize o processador, executando até a sua finalização e, conseqüentemente, impedindo a imparcialidade. Para evitar que isso ocorra, existe a preempção, uma técnica amplamente utilizada nos algoritmos de escalonamento dos sistemas operacionais atuais. Esses algoritmos, também chamados de preemptivos, retiram um processo de execução toda vez que este, por exemplo, estoura o seu limite de tempo para execução.

A preempção permite que os processos concorrentes ao processador recebam uma fatia de tempo para serem executados (*time slice*) e, assim, compartilhem o processador (*time sharing*) proporcionalmente à demanda das aplicações. Com a técnica de *time sharing* as aplicações que necessitam mais processamento que entrada/saída tem mais acesso ao processador quando comparadas às aplicações que necessitam mais entrada/saída que processamento.

A preempção também é uma técnica empregada no escalonamento global, onde um processo pode ser suspenso em um processador e colocado em execução no mesmo ou em

outro processador. A atividade de mudar um processo em execução para outro processador é conhecida como migração de processos e é bem mais complexa que a preempção feita apenas localmente, por ter que tratar problemas como a localização de um processador destino, uma possível transferência do estado do processo para o processador destino, entre outros.

Existem vários algoritmos de escalonamento locais preemptivos e não-preemptivos. Exemplos de algoritmos não-preemptivos podem ser: o First-In-First-Out (FIFO) e o Shortest-Job-First (SJF). Exemplos de algoritmos preemptivos são: Round Robin, por Prioridades, por Múltiplas Filas e por Múltiplas Filas com Realimentação. Este trabalho não abordará os detalhes desses algoritmos, os quais encontram-se descritos em Machado; Maia (1992) e Tanenbaum (1992). O foco principal deste trabalho é o escalonamento global, apresentando detalhadamente os fatores que devem ser considerados para a escolha correta da política de escalonamento a ser empregada.

2.2. Escalonamento Global

Vários autores definem a atividade de escalonamento de processos para plataformas computacionais distribuídas.

Tanenbaum (1995) refere-se ao escalonamento global como "Alocação de Processadores", definindo essa atividade como sendo o estudo dos "algoritmos usados para determinar quais processos serão atribuídos para quais processadores".

Mullender (1993) define o escalonamento global como sendo um serviço de um sistema distribuído, o qual atribui processos para processadores. Esse serviço é denominado de serviço de gerenciamento de processos, enfatizando assim, os processos e não os processadores.

Saphir et al. (1995) definem escalonamento global como sendo "a atribuição de recursos específicos para uma aplicação paralela". Essa definição é voltada aos processadores e não aos processos, fato encontrado principalmente nos trabalhos para a computação paralela.

Shirazi; Hurson (1992) definem a área de escalonamento global como o estudo da distribuição dos processos entre os elementos de processamento, para atingir alguma meta de desempenho, tais como: maximizar o *throughput* ou encontrar o menor *turnaround time*. Essa definição, além de ser voltada à aplicação, agrega também outra importante característica dos algoritmos de escalonamento: os objetivos a que eles se propõem. Esses objetivos inseridos

no algoritmo de escalonamento, determinam a sua "personalidade" e restringem a sua área de atuação.

Casavant; Kuhl (1988) definem o escalonamento global, como sendo um recurso para o gerenciamento de recursos. Esse gerenciamento é composto basicamente por uma política (ou mecanismo) usada para permitir o acesso e o uso de um recurso por seus vários consumidores, de modo eficiente (Figura 1). Os recursos significam, por exemplo: processadores, memórias, rede de comunicação, entre outros; e os consumidores são os usuários representados por suas aplicações seqüenciais distribuídas ou paralelas.

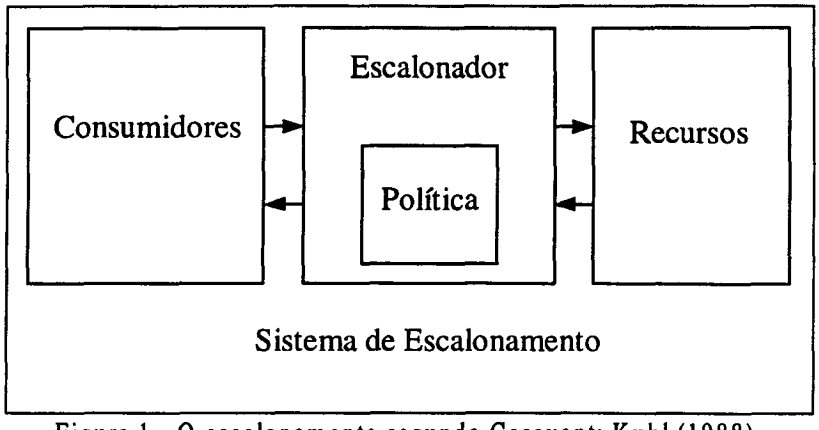

Figura 1 - O escalonamento segundo Casavant; Kuhl (1988).

Baumgartner; Wah (1991) têm uma definição peculiar, sobre o escalonamento em vários processadores. O escalonamento é definido como sendo o mapeamento de um conjunto de processos para um conjunto de processadores, porém, sob o ponto de vista ilustrado pela Figura 2.

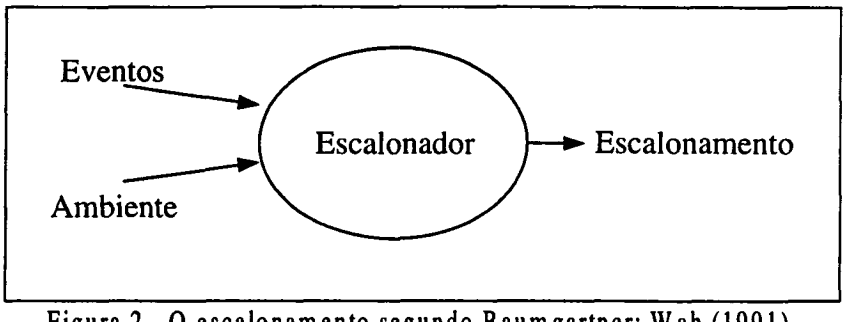

Figura 2 - O escalonamento segundo Baumgartner; Wah (1991).

Os eventos e o ambiente são as entradas para o escalonador e o escalonamento (ou mapeamento) é a saída do escalonador. Os eventos representam a menor entidade de escalonamento, como por exemplo processos. O ambiente engloba todas as demais características que podem afetar a escolha do escalonamento. Dentre essas características

estão: número de processadores, características físicas dos processadores, meio de comunicação, e os objetivos de desempenho. Para os autores os objetivos podem ser, por exemplo, o prazo máximo para a conclusão de uma aplicação de tempo-real ou então determinar se é possível obter alguma melhoria de desempenho com o escalonamento.

2.3. Escalonamento em Plataformas Computacionais Distribuídas e Heterogêneas

Uma plataforma computacional distribuída e heterogênea é uma coleção de elementos de processamento, fisicamente distribuídos, conectados por uma rede de comunicação e utilizada com objetivos específicos. Essa definição apesar de assemelhar-se à definição de redes de computadores é mais abrangente, porque os elementos de processamento podem pertencer a uma NOW (*Network of Workstations*) composta por computadores pessoais e/ou estações de trabalho [Anderson et al. (1995)], ou até mesmo a computadores paralelos, todos podendo estar interligados e formando uma única plataforma computacional.

Considerando as plataformas distribuídas, Tanenbaum (1996) cita que o termo "multiprocessadores" designa plataformas computacionais com mais de um processador, os quais compartilham o mesmo endereço de memória (possuem memória compartilhada ou centralizada). O termo "multicomputadores" designa plataformas onde cada processador possui a sua própria memória local, formando uma memória distribuída. Ainda segundo Tanenbaum (1996), as plataformas podem ser classificadas ou como fracamente acopladas (*loosely-coupled*) ou então como fortemente acopladas (*tightly-coupled*). Nas plataformas fortemente acopladas, o envio de uma mensagem entre dois processadores apresenta um atraso (*delay)* pequeno e uma taxa de transferência alta (número de *bits* enviados), quando comparadas às plataformas fracamente acopladas, onde o envio de mensagens possui um atraso maior e uma taxa de transferência mais baixa.

Em geral, os computadores paralelos são bons exemplos de plataformas fortemente acopladas e os sistemas distribuídos bons exemplos de plataformas fracamente acopladas (principalmente quando estes são constituídos por estações de trabalho conectadas por uma rede de comunicação padrão, como a *ethernet*).

Caso a plataforma distribuída e heterogênea possua na gerência das suas atividades um sistema operacional distribuído, tem-se um "sistema distribuído", caso contrário tem-se uma

plataforma computacional distribuída fisicamente e com possíveis diferentes arquiteturas e/ou capacidades de processamento. Essa plataforma computacional pode ter diferentes objetivos, os quais podem ser desde o simples compartilhamento de recursos (como unidades de disco e impressoras) ou então a execução de aplicações paralelas através da união lógica (ou virtual) dos processadores, por intermédio de ferramentas de softwares como o PVM e o MPI [Geist et al. (1994), Snir et al. (1996)].

As plataformas distribuídas e heterogêneas possuem características interessantes, as quais contribuem enfaticamente para a sua grande utilização e, algumas vezes, particularizam a atividade de escalonamento. A melhor relação custo x benefício é um dos fatores responsáveis pela grande utilização, principalmente quando a plataforma é composta por NOWs. Russ et al. (1997) citam que a principal razão para a grande utilização das NOWs é o seu baixo custo relativo e não o seu desempenho. No entanto, se forem consideradas as NOWs já instaladas, essa grande quantidade de recursos oferece um grande potencial para processamento. Anderson et al. (1995) demonstraram em seus estudos que mesmo em horários normais de trabalho, mais de 60% dos computadores estavam 100% disponíveis, tornando as NOWs fortes candidatas ao processamento paralelo. A utilização das NOWs para o processamento paralelo agregou novos fatores ao escalonamento de processos, os quais não apresentavam a mesma importância tanto na computação seqüencial feita em plataformas distribuídas, quanto na computação paralela feita em computadores "realmente" paralelos. As características multiusuária e heterogênea das NOWs são bons exemplos desses fatores que obrigaram as políticas de escalonamento adaptarem-se para uma melhor distribuição dos processos.

2.4. Problemas com a Terminologia da Área de Escalonamento de Processos

Para estabelecer uma taxonomia para a área de escalonamento de processos é necessário determinar por quais características os trabalhos desenvolvidos devem ser organizados. Para tanto, deve-se utilizar uma terminologia coesa que viabilize comparações coerentes e que realmente auxilie na escolha e utilização corretas do escalonamento de processos.

Entretanto, não há um conjunto de características que possam ser rotuladas como "padrão" para a comparação dos algoritmos de escalonamento propostos, nem tão pouco uma nomenclatura coesa que facilite essa comparação.

Devido à grande variedade de trabalhos na área, houve a proliferação de uma terminologia inconsistente e contraditória, com diferentes formulações e hipóteses para o problema do escalonamento [Casavant; Kuhl (1988)]. A conseqüência direta disso é a dificuldade para analisar os méritos de esquemas alternativos, principalmente quando é necessário compará-los com outros trabalhos já desenvolvidos.

Feitelson; Rudolph (1996) também apontam a dificuldade de relacionar os vários trabalhos já desenvolvidos e, indo além, citam que "em alguns casos, tais comparações estão equiparando maçãs com laranjas, como se elas fossem a mesma coisa". Com isso, alguns desses trabalhos são desconsiderados, por serem vistos erroneamente como irrelevantes e, em outros casos, é simplesmente difícil verificar se a comparação faz sentido ou não.

As datas das publicações citadas acima (1988 e 1996) demonstram que a falta de uma terminologia consistente persiste, mesmo se forem consideradas as propostas de novas taxonomias feitas dentro desse período de oito anos. Analisando-se as publicações posteriores a 1996, não foi encontrado nenhum trabalho que se propusesse a solucionar a questão.

2.4.1. Divergências na Terminologia

Há vários exemplos de divergências que ilustram os problemas na terminologia da área. Neste trabalho optou-se por escolher apenas quatro grupos de termos, os quais são abrangentes e significativos.

Os quatro grupos além de representarem termos básicos, contribuem para um maior entendimento das taxonomias que serão descritas posteriormente. Cada grupo possui termos afins usados indiscriminadamente como sinônimos ou não. Os grupos de termos escolhidos foram os seguintes: (1) aplicações, *jobs*, *threads*, tarefas, programas e processos; (2) políticas, estratégias, mecanismos e algoritmos de escalonamento; (3) escalonamento de processos, balanceamento de cargas, alocação de recursos e outros afins; (4) migração de processos, balanceamento de cargas dinâmico e escalonamento com preempção e outros afins.

Aplicações, *jobs*, *threads*, tarefas, programas e processos. Esses termos, apesar de algumas divergências, seguem uma padronização em função do seu uso na computação. O problema reside em distinguir certas definições como: um *job* corresponde a um processo ou a um conjunto de processos? Uma tarefa tem o mesmo significado de um processo ou tem o significado de um *thread*?

Devarakonda; Iyer (1989) e Tanenbaum (1995) usam o termo processo como sendo uma abstração de um programa em execução, composta por um código fonte, uma entrada, uma saída e um estado (dados do programa e pilha, contador de programa e todas as demais informações necessárias para executar o programa). Para Tanenbaum (1995) *threads* ou *lightweight processes* são diferentes linhas de execução (concorrentes) que um mesmo processo pode ter, compartilhando o mesmo espaço de endereçamento de memória. Cada *thread* executa seqüencialmente e de maneira análoga a um processo normal, no entanto compartilha memória com outros threads. Ainda para Tanenbaum (1995), um *job* é um programa ou um conjunto de programas.

Mullender (1993) segue a mesma terminologia de Tanenbaum (1995), porém destaca a "confusão" (termo usado pelo autor) que diferentes grupos de pesquisa fazem por usar nomes distintos e algumas vezes conflitantes para espaços de endereçamento e *threads*.

Feitelson; Rudolph (1995), Vesseur et al. (1995), Ferstl (1996), Islam et al. (1996), Russ et al. (1996) e Stellner; Trinitis (1997) definem *job* como sendo a maior "entidade" de execução (um programa ou aplicação) constituída de uma ou mais tarefas (processos ou *threads*).

Chen; King (1996) e Harchol-Balter et al. (1997) usam o termo tarefa sem definir explicitamente o que ele representa, onde as descrições feitas induzem o leitor a acreditar que uma tarefa seja sinônimo de um processo. Em outro trabalho de Harchol-Balter [Harchol-Balter; Downey (1997)], também sobre escalonamento, a terminologia usada já é outra. O termo tarefa não aparece mais, sendo substituído pelo termo processo.

O trabalho de Saphir et al. (1995), desenvolvido pela NASA, cita que o termo aplicação paralela corresponde a "um programa MIMD – *Multiple-Instructions-Multiple-Data*, o qual é projetado para execução simultânea em vários processadores com memória distribuída". Para Saphir et al. (1995) "um *job* paralelo é uma seqüência de operações requisitadas por um usuário, onde uma aplicação paralela é a parte principal". Apesar dessa distinção entre *job* e aplicação, os autores citam posteriormente que usam esses termos como equivalentes. A definição de uma aplicação paralela como "um programa MIMD" não é adequada. A sigla MIMD foi estabelecida por Flynn (1972) para classificar arquiteturas paralelas e não programas concorrentes. Há aqui, portanto, uma grande confusão com a terminologia, não só em função do erro com o uso da sigla MIMD, mas também devido ao fato de se definir aplicação e *job* como coisas distintas e depois usá-las como sinônimos.

Este trabalho segue, na medida do possível, a mesma terminologia adotada pela maioria dos trabalhos pesquisados. Assim, os termos processo e *thread* tem o mesmo significado apresentado por Tanenbaum (1995).

O termo tarefa é empregado como sinônimo de um processo. Os termos aplicação e *job* são reservados para designar a maior "entidade" de execução. Dessa forma, uma aplicação pode conter vários processos que cooperam entre si para resolver um determinado problema (no caso uma aplicação concorrente) ou apenas um processo (no caso uma aplicação seqüencial).

Políticas, estratégias, mecanismos e algoritmos de escalonamento. Embora alguns autores como Casavant; Kuhl (1988) considerem erroneamente os termos política e mecanismo como sinônimos, na realidade não são. A política ou estratégia determina como os mecanismos serão empregados para que o escalonamento seja realizado, considerando os objetivos propostos para o algoritmo de escalonamento. Os mecanismos compreendem os comandos e/ou as instruções utilizados para, por exemplo, coletar informações sobre a carga do sistema e trocar informações entre os processadores [Song et al. (1997)]. Algoritmo de escalonamento será utilizado neste trabalho como um termo mais genérico o qual implementa as políticas e os mecanismos utilizados pela atividade de escalonar.

Os mesmos mecanismos podem ser utilizados por políticas de escalonamento diferentes e assim, produzirem resultados completamente diferentes. Um exemplo de mecanismo é a preempção citada anteriormente, onde os mecanismos são os responsáveis por aplicar a atividade de suspensão do processo em execução, salvando todo o estado do processo. Questões como: quando a preempção será feita, qual processo será transferido e para qual processador o processo será enviado, dependem da política de escalonamento empregada.

Alguns trabalhos como o *SPRITE* e o *V Distributed System* não separam a política dos mecanismos. Esses algoritmos, implementados no *kernel* do sistema operacional, contêm ambos: política e mecanismos necessários ao escalonamento. Essa abordagem impõe problemas com a portabilidade do sistema de escalonamento, embora, potencialmente, atinja um melhor desempenho em função de estar implementada em um *kernel* específico à plataforma utilizada [Eskicioglu (1989)].

Tanenbaum (1992) ressalta a importância de se separar a política dos mecanismos, a fim de permitir que o usuário intervenha através de parâmetros (nenhuma ou pouca transparência), nas decisões do algoritmo de escalonamento. Essa preocupação deve-se à diversidade de

aplicações e também à dificuldade para determinar antecipadamente as características e necessidades dos processos escalonados.

Escalonamento de processos, balanceamento de cargas e alocação de recursos. Feitelson; Rudolph (1995) citam que o maior problema de se pesquisar sobre escalonamento de processos em computadores paralelos é que essa expressão possui significados diferentes para diferentes pessoas. Embora essa afirmação limite o problema apenas aos computadores paralelos (deveria abranger as plataformas distribuídas como um todo), ela é verdadeira.

Alguns dos termos usados como sinônimos para a atividade de escalonamento são: balanceamento de carga (*load balancing*), balanceamento de carga dinâmico/estático/adaptável (*dynamic/static/adaptive load balancing*), distribuição de carga (*load distribution*), compartilhamento de carga (*load sharing*), compartilhamento de recurso (*resource sharing*), sistema de gerenciamento de recurso/*job* (*resource/job management system*), alocação de recurso (*resource allocation*), escalonamento de recurso (*resource scheduling*), escalonamento de processador (*processor scheduling*), escalonamento dinâmico/estático (*dynamical/static scheduling*), escalonamento paralelo (*parallel scheduling*), atribuição de tarefa (*task assignment*), política de atribuição (*assignment policy*), escalonamento *on-line/off-line* (*on-line/off-line scheduling*).

Muitos desses termos são válidos porque são usados para distinguir as políticas propostas, através de alguma característica do escalonamento em si. O problema reside em não se explicar o significado dos termos empregados, utilizando-os diretamente como sinônimo da atividade de escalonamento. Outro grande problema é a falta de um contexto geral, situando as novas propostas (e seus termos) em um contexto conhecido por todos, para que se possa entender o que foi desenvolvido e os seus objetivos. Em alguns dos textos publicados, se (e somente se) o leitor possuir um conhecimento maior da área, é possível deduzir as semelhanças e diferenças entre os termos empregados. Outras vezes os autores utilizam, sem necessidade, termos que contribuem somente para confundir ainda mais a área.

Um dos problemas mais comuns é o uso do termo balanceamento de cargas como sinônimo de escalonamento realizado em tempo de execução da aplicação. A associação direta desses termos, mesmo que implícita, não é correta, porque o balanceamento das cargas de uma plataforma é um dos possíveis objetivos que um algoritmo de escalonamento pode ter, e não o único. Mullender (1993) exemplifica esse problema fazendo a seguinte afirmação: "é

possível, através de uma alocação prudente, otimizar o desempenho do sistema como um todo. Esse processo é conhecido como balanceamento de carga...".

O trabalho feito por Bestavros (1997) é um exemplo de que o balanceamento de cargas nem sempre é o objetivo do escalonamento. Bestavros (1997) propõe uma nova política de escalonamento para um sistema distribuído de tempo real; utiliza corretamente o termo "escalonamento" e explica a terminologia utilizada. A política proposta faz o escalonamento de tal forma que as necessidades de tempo-real dos processos sejam atendidas dentro dos limites especificados. O balanceamento da carga na plataforma computacional é irrelevante nesse caso.

O artigo de Lüling et al. (1993), embora proponha uma nova taxonomia, serve como exemplo para o uso conflitante de vários termos da área. Os termos balanceamento de carga, compartilhamento de recursos, escalonamento de recursos, escalonamento de *jobs*, migração de tarefas e balanceamento de cargas dinâmico são usados como sinônimo. Alguns desses termos, como migração de tarefa e escalonamento de *job* não representam a mesma coisa para vários outros autores.

Franklin; Govindan (1996), Graham; Barker (1994) e Tripathi; Karnik (1995) utilizam os termos balanceamento de carga estático para designar a atribuição inicial de processos em tempo de execução e balanceamento de carga dinâmico para designar a transferência de processos de um processador para outro (migração de processos). A utilização do termo estático aqui está divergente com o mesmo termo usado por vários outros autores para designar o escalonamento feito em tempo de compilação. Outra característica do trabalho de Franklin; Govindan (1996) é a utilização do termo adaptável como sinônimo de dinâmico, fato encontrado somente nesse artigo, dentre os pesquisados neste trabalho.

Alanyali; Hajek (1995) utilizam o termo balanceamento de carga dinâmico, apenas como sinônimo para escalonamento, sem fazer menção à migração. Porém, os autores definem corretamente balanceamento de carga como sendo um dos possíveis objetivos para a alocação de recursos. O trabalho de Alanyali; Hajek (1995) demonstra que o problema principal não está em rotular a atividade de escalonamento com o seu objetivo, mas sim em não explicar essas diferenças. Outra questão nesse trabalho é a preferência dos autores com o termo "alocação de recursos" em detrimento ao termo "escalonamento de processos", o que não está errado. Casavant; Kuhl (1988) têm uma explicação sobre esses dois termos: "Escalonamento e alocação nada mais são do que dois termos que descrevem a mesma atividade, com diferentes

pontos de vista". Enquanto o "escalonamento de processos" aborda a questão sob o ponto de vista dos processos, o termo "alocação de recursos" é direcionado aos processadores.

Shivaratri et al. (1992) distinguem "compartilhamento de carga" de "balanceamento de carga". O compartilhamento de carga denomina os algoritmos que têm como meta evitar estados de não-compartilhamento. Tais estados são aqueles em que um processador permanece ocioso enquanto ainda há tarefas para serem executadas em outros processadores. Por outro lado, o balanceamento de carga, embora também tente evitar os estados de não-compartilhamento, vai além de simplesmente compartilhar a carga, tentando equilibrar a carga dos processadores. Xu; Lau (1997) apresentam a mesma definição para os termos compartilhamento e balanceamento de carga. O trabalho de Wang; Morris (1985), adota outra postura e afirma que os termos balanceamento e compartilhamento de carga como sinônimos.

No trabalho de Ferstl (1996) é utilizado o termo "sistema de gerenciamento de *job* e de recurso". Nesse trabalho há uma separação explícita do termo "escalonador", este como um item do sistema de gerenciamento de *job* e recurso, o qual é responsável por atribuir a carga de trabalho para os processadores e influenciar os *jobs* que já estão executando nesses processadores. O trabalho de Ferstl (1996) não é o único a destacar o "escalonamento" (ou escalonador), separando-o como se fosse um item independente e/ou diferente das demais atividades feitas nas ferramentas de software propostas. Além do trabalho de Ferstl (1996), Saphir et al. (1995) e Krueger et al. (1994) também são exemplos de trabalhos que apresentam essa separação, cada um a seu modo.

O trabalho de Saphir et al. (1995) também utiliza o termo "sistema de gerenciamento de *job*", ressaltando as necessidades e deficiências da área de escalonamento. Nesse trabalho o "sistema de gerenciamento de *job*" é dividido em três partes principais: (1) um servidor de usuário para submeter/remover *jobs* e pesquisar sobre o *status* desses *jobs*; (2) um escalonador de *jobs* para escaloná-los de acordo com uma política de escalonamento; e (3) um gerenciador de recursos para alocar, monitorar e garantir a alocação e as políticas de escalonamento. Embora haja uma pequena explanação sobre esses termos, esta é insuficiente para determinar como essa divisão pode ser corretamente relacionada aos outros trabalhos, que usam uma terminologia diferente. Isso deve-se, principalmente pelo fato dos autores utilizarem no restante do artigo apenas o termo genérico "gerenciador de recursos", não fazendo mais distinção entre as três partes apresentadas. São usados ainda outros termos espalhados pelo artigo, tais como: escalonamento de "recurso" com o mesmo sentido de

escalonamento de "processos" e alocação de recurso preemptiva no sentido de migração de processos.

De um modo geral, "sistema de gerenciamento de *jobs*" e "sistema de gerenciamento de recursos" são termos empregados para ferramentas de software que, além de empregarem algum de tipo de escalonamento, também acrescentam outras características como uma interface com o usuário, monitoramento do sistema, tolerância a falhas, entre outras. Porém, o escalonamento continua sendo a principal finalidade desses sistemas de gerenciamento e essas outras características, apesar de importantes para, por exemplo, facilitar o uso da ferramenta e também aumentar a satisfação de seus usuários, são atividades secundárias.

A separação do termo "escalonamento" é empregada em geral nos trabalhos que envolvem de alguma maneira, a computação paralela. Nessas separações o termo "escalonamento" é empregado no sentido de "política de escalonamento". As outras partes, tais como: "gerenciador de recursos", "alocador de processadores", "gerenciador de tarefas" e "servidor de usuários" são, na realidade, ou (1) mecanismos para coleta de informações sobre a plataforma computacional e "reserva" de processadores para um grupo de processos (*space sharing*) ou (2) uma interface para o usuário ter acesso ao escalonamento para, por exemplo, submeter os seus processos.

Feitelson et al. (1997) e Shmoys et al. (1995) utilizam o termo escalonamento *on-line* como sinônimo do escalonamento feito em tempo de execução [Chen; King (1996) e Casavant; Kuhl (1988)]. Entretanto, a definição apresentada pelos autores não relaciona explicitamente esses termos, abordando o escalonamento *on-line* como aquele que não conhece antecipadamente nenhuma informação sobre o processo que será executado. Com isso não é possível determinar a melhor maneira de se escalonar o processo durante a fase de compilação do mesmo. Como não há uma relação explícita entre os dois termos (escalonamentos *on-line* e dinâmico), essa associação só é possível para leitores com um maior grau de conhecimento ou que no mínimo saibam a definição de um escalonamento dinâmico. Outra característica desses trabalhos é o enfoque voltado inteiramente para aplicações paralelas e não para aplicações seqüenciais executadas em um sistema distribuído.

Schwiegelshohn (1996) utiliza o termo escalonamento *off-line* de maneira análoga à utilização do termo escalonamento *on-line*, indicando o escalonamento feito com informações prévias sobre o comportamento e as necessidades do processo. Outros autores como Casavant; Kuhl (1988), denominam esse escalonamento *off-line* como escalonamento estático.

Shu; Wu (1996) utilizam o termo "escalonamento paralelo" em sua proposta. O termo "paralelo" refere-se a um escalonamento realmente feito em paralelo e de uma maneira síncrona por todos os processadores envolvidos na execução da aplicação paralela. O escalonamento é dividido em duas fases, sendo que em uma (a "fase de sistema") todos os processadores sincronamente coletam as informações sobre a carga do sistema e efetivamente escalonam as suas tarefas; e na outra fase (a "fase de usuário") todos os processadores executam os processos escalonados.

Feitelson et al. (1997) e Gibbons (1997) também utilizam o termo "escalonamento paralelo". O termo é usado no sentido de "escalonamento de *jobs* paralelos" e em nada assemelha-se ao trabalho de Shu; Wu (1996) citado anteriormente.

Neste trabalho será utilizado o termo escalonamento de processos, no lugar dessa grande quantidade de termos inseridos na literatura. A opção por esse termo deve-se à simplicidade e objetividade da terminologia. As diferentes políticas e objetivos que o escalonamento de processos pode vir a ter, serão definidas e denominadas ainda neste trabalho, quando forem apresentadas as taxonomias encontradas na literatura.

Migração de processos, balanceamento de carga dinâmico e escalonamento com preempção. A migração de processos (ou escalonamento global com preempção) é uma técnica utilizada por alguns algoritmos de escalonamento que permite a suspensão de um processo em um processador e a sua conclusão em outro processador. A migração de processos altera um escalonamento já realizado, devido a, por exemplo, variações ocorridas na plataforma computacional.

Um algoritmo de escalonamento com migração de processos é apenas uma das possíveis variações que um algoritmo de escalonamento pode possuir. No entanto, essa característica está sendo destacada das demais em função da sua complexidade e importância. Alguns dos termos encontrados na literatura para designar a migração de processos são: escalonamento com preempção (*scheduling with preemption*), balanceamento de carga dinâmico (*dynamical load balancing*), balanceamento de carga automático (*automatic load balancing*), migração preemptiva (preemptive migration), migração dinâmica de tarefa (*dynamical migration task*), alocação de tarefa automatizada (automated task allocation) e "reatribuição dinâmica" (*dynamic reassignment*).

O termo balanceamento de carga dinâmico é utilizado como um sinônimo para migração de processos em vários artigos, tais como nos trabalhos publicados por Ferstl

(1996), Vesseur et al. (1995) e Harchol-Balter; Downey (1997). Nos dois primeiros trabalhos essa associação é clara e bem explicada, porém, no último deles essa associação não é feita de uma maneira simples. Harchol-Balter; Downey (1997) utilizam, além de "balanceamento de carga dinâmico", termos como: migração preemptiva e não-preemptiva (esta última também chamada no artigo de "execução remota"). A migração preemptiva representa o escalonamento de processos ativos (com suspensão e transferência de processos durante a execução). A migração não-preemptiva envolve apenas processos novos em um processador remoto. Essa terminologia não é comum e não foi encontrada em nenhum outro artigo pesquisado para este trabalho.

O termo "balanceamento de carga dinâmico" também é encontrado como sinônimo de escalonamento de processos em tempo de execução e sem migração [Ferrari; Zhou (1987), Mehra; Wah (1993) e Shirazi et al. (1995a)]. Embora essa inclusão do termo "dinâmico" pareça ter algum sentido, em virtude do escalonamento ser feito em tempo de execução, ela não permite determinar se o algoritmo faz ou não a migração dos processos em execução [Ferstl (1996), Harchol-Balter; Downey (1997)].

Os trabalhos publicados por Russ et al. (1996, 1997 e 1998) utilizam os termos: alocação de tarefa automatizada, migração dinâmica de tarefa, alocação de recursos, balanceamento de carga dinâmico e algumas outras variações como sinônimos para migração de processos.

Feitelson et al. (1997) utilizam corretamente o termo escalonamento e preempção. Para esses autores existem tipos diferentes de preempção, dos quais destacam-se: a preempção local (*local preemption*) e a preempção migratória (*migratable preemption*). A primeira representa a preempção feita em processos que retornarão a executar no mesmo processador. A segunda refere-se à preempção feita em processos que podem ser executados em outro processador. Embora sejam introduzidos novos termos, estes são bem explicados e objetivos.

Casavant; Kuhl (1988), utilizam os termos "atribuição única" (*one-time assignment*) e "reatribuição dinâmica" (*dynamic reassignment*) para designar o escalonamento feito respectivamente sem e com a migração de processos. Esse trabalho, provavelmente por apresentar uma taxonomia básica para a área, também utiliza os termos: migração de processos, escalonamento não preemptivo e preemptivo.

2.4.2. Razões para as Divergências na Terminologia

Analisando-se a literatura disponível, encontram-se alguns artigos que apontam e criticam a existência dessa terminologia confusa na área de escalonamento de processos, no entanto, não foi encontrado um trabalho que apontasse os possíveis motivos que levam à essa discordância.

Genericamente, o que pode explicar essa situação é a falta de uma teoria básica comum sobre escalonamento, assim como a grande abrangência dessa área. As publicações sobre escalonamento na computação estão direcionadas para: sistemas operacionais (normalmente distribuídos), computação paralela e ferramentas de software responsáveis pelo escalonamento em plataformas distribuídas (implementadas no espaço do usuário). Assim, trabalhos em qualquer uma dessas classes apresentam implementações, terminologias e/ou classificações próprias, as quais nem sempre aplicam-se perfeitamente a outras classes.

Um dos fatores que podem estar implícitos nos trabalhos disponíveis é que as pessoas costumam utilizar uma terminologia que lhes é usual. Os pesquisadores acostumados a trabalhar diretamente com a computação paralela ou com os sistemas operacionais (distribuídos ou não), utilizam seus próprios termos, muitas vezes, para designar a mesma atividade. Novamente, o que falta é uma teoria única e básica para a área de escalonamento, independente da plataforma computacional e do pesquisador.

Exemplos desse caso são os trabalhos voltados para os sistemas operacionais distribuídos de Tanenbaum (1992 e 1995). Para definir o escalonamento feito sobre vários processadores em um sistema distribuído, Tanenbaum utiliza o termo "alocação de processadores", destacando a área sob o ponto de vista da plataforma e não da aplicação. Embora os termos utilizados acima sejam direcionados à plataforma, a definição dada é direcionada à aplicação ("quais processos serão atribuídos para quais processadores"). A adoção dessa terminologia, embora justificada pelo autor como "... seguindo a tradição ...", é creditada (neste trabalho) ao fato do termo "escalonamento de processos" já ter sido utilizado anteriormente para definir exclusivamente o escalonamento feito em um processador. Isso pode ser verificado na seção 12.4 de Tanenbaum (1992) e na seção 4.4 de Tanenbaum (1995), onde é citada a expressão "... não há realmente muito a dizer sobre escalonamento em um sistema distribuído. Normalmente cada processador faz o seu próprio escalonamento local...".

As explicações para as divergências podem ser encontradas em dois grandes grupos de trabalhos. No primeiro grupo os trabalhos divergem por serem voltados às necessidades ou da

computação paralela ou dos sistemas distribuídos (escalonamento na computação paralela x escalonamento em sistemas distribuídos). O segundo grupo representa os trabalhos que divergem em função do local da sua implementação (implementações no *kernel* x no espaço do usuário). Os próximos parágrafos discutem o por quê das divergências nesses dois grupos e demonstram alguns casos onde as mesmas ocorrem.

Escalonamento na Computação Paralela x Escalonamento em Sistemas Distribuídos: Rapine et al. (1998) citam que o escalonamento na computação paralela é um problema de otimização da utilização dos recursos pelos processos e nos sistemas distribuídos é um problema de gerenciamento de requisições de vários usuários.

As duas áreas, computação paralela e sistemas distribuídos, embora apresentem similaridades no que se refere à atividade de escalonamento, apresentam também diferenças expressivas, tornando muitos dos algoritmos desenvolvidos para uma área impróprios para a outra.

Saphir et al. (1995) destacam porque os algoritmos de escalonamento desenvolvidos para aplicações paralelas não são apropriados para aplicações seqüenciais. Para esses autores o escalonamento feito para computadores paralelos com memória distribuída são significativamente mais complexos que aqueles feitos para plataformas com um processador. A maior complexidade deve-se, por exemplo: às solicitações por processadores nas aplicações paralelas, que variam o número e o tipo de nós; às exigências por outros recursos, como grande quantias de memória; à escolha do método para *time sharing*, o qual afeta diretamente o desempenho da política de escalonamento e à maior dependência entre os processos pertencentes a uma aplicação, os quais, por normalmente serem gerados através de um processo mestre, devem ser finalizados também através deste.

O trabalho de Islam et al. (1996) é um exemplo de trabalho voltado exclusivamente para a computação paralela. É apresentada uma análise sobre diferentes algoritmos pertencentes à política de escalonamento chamada *space sharing*. Essa política refere-se aos algoritmos de escalonamento que dividem os processadores em grupos e permitem que somente os processos de uma aplicação paralela sejam atribuídos a esses grupos. Não encontrou-se na literatura pesquisada, nenhum escalonamento *space sharing*, nem tão pouco diferentes tipos de partições de processadores (*processor partitioning*) feitas para "sistemas distribuídos".

O trabalho de Wang; Morris (1985) é um exemplo de trabalho voltado exclusivamente aos sistemas distribuídos. Os autores citam que a proposta inclui apenas processos

independentes, seqüenciais, sem migração e em plataformas homogêneas. Lendo-se a proposta feita nesse trabalho percebe-se que ele apresenta uma abordagem completamente distinta em relação às abordagens apresentadas anteriormente nesta seção. Os trabalhos de Graham; Barker (1994), Lim; Kim (1995), Shivaratri et al. (1992), também são exemplos direcionados aos sistemas distribuídos. As questões abordadas nesses trabalhos envolvem o escalonamento de um processo independente sobre os processadores, sempre com o objetivo de balanceamento das cargas. A preocupação constante nesses trabalhos é encontrar processadores ociosos que recebam um processo. Apesar de se encontrar na literatura alguns poucos trabalhos que se preocupam com o escalonamento e com a sincronização de vários *threads* em um sistema distribuído [Mullender (1993), Tanenbaum (1995)], a ênfase dada a esses problemas é pequena, quando comparada com a ênfase desses mesmos problemas na computação paralela. Isso deve-se, principalmente, à preocupação constante da computação paralela pelo desempenho (com todos os seus significados) e não tanto pelo compartilhamento.

Implementações no *kernel* e no espaço do usuário (*user space*): mesmo com objetivos semelhantes, as diferentes implementações permitem que seus autores criem nomenclaturas próprias. Uma das possíveis variações na implementação pode ser o local onde o escalonamento é implementado: no *kernel* do sistema operacional ou no espaço do usuário.

Para Feitelson; Rudolph (1995), cujo trabalho é voltado exclusivamente para a computação paralela, o escalonamento dinâmico pode ser feito no sistema operacional ou então em *runtime systems*. Esses *runtime systems* são ferramentas no espaço do usuário responsáveis pelo escalonamento, as quais têm por objetivo satisfazer as necessidades individuais de uma aplicação. Quando as necessidades passam a ser coletivas (não apenas de uma aplicação, mas da plataforma computacional como um todo) o escalonamento deve ser feito no sistema operacional, segundo os autores citados.

Ainda segundo Feitelson; Rudolph (1995), essa divisão tem contribuído para "impedir o progresso" das pesquisas em escalonamento, por não deixar claro quem é responsável pelo problema. As opiniões, nesse caso, variam desde aquelas que acham que o escalonamento deve ser feito pelos *runtime systems* até aquelas que defendem que o *kernel* do sistema operacional deve ser o responsável por todo o trabalho. Nesse trabalho os autores defendem a realização do escalonamento pelo sistema operacional, alegando que algumas atividades, tais como: interrupções, chaveamento de contextos e gerenciamento da memória virtual são vitais

para atingir bons desempenhos nos computadores paralelos. Quando essas atividades são feitas no espaço do usuário há uma perda de desempenho.

O trabalho de Paindaveine; Milojicic (1996), baseado no sistema operacional Mach, é um exemplo de como nomenclaturas próprias podem surgir em função do nível da implementação do escalonamento. A principal diferença está na comparação feita entre a migração para processos e a migração para tarefas. Essa separação (processos x tarefas), não é usual e deve-se à especificidade desse trabalho, onde são apresentadas questões sobre o escalonamento com preempção em diferentes níveis, desde o *kernel* do sistema operacional até a aplicação. Segundo esse trabalho, se a migração é feita em alto nível (espaço do usuário), a implementação é mais simples, mas o desempenho, a transparência e a reusabilidade são reduzidos. Essa redução deve-se à necessidade de duplicação da maioria dos itens que compõem o estado de um processo cada vez que este muda de processador. Barak et al. (1996) na apresentação do sistema operacional MOSIX, defendem a mesma posição apresentada por Paindaveine; Milojicic (1996).

Ferstl (1996) apresenta uma visão completamente diferente nesse aspecto, defendendo que o escalonamento deve ser feito por um sistema de gerenciamento de processos e recursos, independente do sistema operacional. Essa separação apresenta como vantagens: (1) permitir a utilização dos recursos de uma maneira estruturada, planejada e controlada pela administração; (2) oferecer os recursos de uma plataforma computacional para os seus usuários de uma maneira transparente e de fácil utilização e compreensão; (3) fornecer uma interface ao usuário independente da plataforma computacional (portabilidade).

Análogos ao pensamento apresentado por Ferstl (1996), vários trabalhos têm sido desenvolvidos no espaço do usuário, tais como: Condor [Epema et al. (1996)], MPVM [Song et al. (1997)], MMPVM [Casas et al. (1995a, 1995b)] e S-MPI [Dantas; Zaluska (1998)].

Embora os defensores do escalonamento no *kernel* do sistema operacional aleguem a falta de transparência no escalonamento feito no espaço do usuário, os trabalhos publicados nesse nível demonstram várias alternativas que implementam o escalonamento de modo completamente transparente à aplicação.

Segundo Eskicioglu (1989) há a possibilidade de implementar o escalonamento nos dois níveis: *kernel* e espaço do usuário. Implementando-se os mecanismos no *kernel* e as políticas no espaço do usuário, poder-se-ia minimizar a sobrecarga extra, gerenciando certas

informações diretamente no *kernel*. Segundo o autor a desvantagem está no "chaveamento" necessário para a troca das informações entre os dois níveis.

Segundo Krone et al. (1998) quando o escalonamento é feito em plataformas distribuídas, heterogêneas e multiusuárias, o escalonamento deve ser feito fora do sistema operacional e integrado à aplicação, em função da heterogeneidade em diferentes níveis (tanto no hardware quanto no software). Segundo os autores, "é praticamente impossível" adaptar todos sistemas operacionais dos diferentes computadores existentes em tais plataformas, para os objetivos do escalonamento. Ainda segundo Krone et al. (1998), uma das vantagens de se fazer o escalonamento no espaço do usuário é justamente o desempenho, qualidade destacada por outros autores como sendo do escalonamento no *kernel*.

2.5. Taxonomias da Área de Escalonamento de Processos

Para Baumgartner; Wah (1991) o objetivo de uma taxonomia deve ser aumentar e organizar o "conhecimento total" sobre uma classe de problemas, especificando-a e mostrando o relacionamento entre problemas. Os autores apontam ainda, no mínimo quatro características desejáveis de uma taxonomia: (1) identificar características significantes: pois contribuem para uma solução eficiente; (2) mostrar claramente o relacionamento entre os problemas: a solução de um problema pode ser semelhante à solução de outro problema que esteja relacionado com o primeiro; (3) capaz de expandir e contrair: permite especializar a taxonomia a determinadas situações, onde pode haver a necessidade de um maior número de detalhes ou não, permitindo assim, que a taxonomia reduza a sua complexidade, caso seja possível; e (4) separar a especificação do problema da sua solução: essa separação permite comparar diversas soluções para o mesmo problema. A questão aqui é separar a política empregada para solucionar um problema do problema em si, separando dessa maneira o "como é resolvida uma questão" do "o que deve ser resolvido".

A área de escalonamento de processos possui várias taxonomias, muitas delas criadas sem as características descritas acima por Baumgartner; Wah (1991) [Chen; King (1996), Gibbons (1997), Hwang; Xu (1998); Kemelmakher; Kremien (1998), Mullender (1993) e Russ et al. (1996)]. Em função disso, esta seção descreve algumas taxonomias propostas, escolhidas em função da sua abrangência e também da sua aceitação por outros trabalhos.

2.5.1. Taxonomia de Casavant; Kuhl (1988)

O trabalho de Casavant; Kuhl (1988) teve por objetivo fornecer uma estrutura adequada para que os trabalhos até então desenvolvidos (1988) pudessem ser comparados, e também para que trabalhos futuros pudessem ser classificados e discutidos. Essa taxonomia não é a mais antiga a ser apresentada neste trabalho, mas em função de sua abrangência em relação a outras, preferiu-se colocá-la primeiro para que os termos nela definidos servissem como referencial para os utilizados por outras taxonomias.

A taxonomia proposta é uma taxonomia híbrida entre hierárquica e plana. São definidos dois conjuntos, um com termos que relacionam-se entre si (hierárquica) e outro conjunto de termos que podem relacionar-se com quaisquer outros, sem que haja obrigatoriamente uma hierarquia. Segundo os autores, essa taxonomia além de ser direcionada ao escalonamento de processos também é aplicável para qualquer conjunto de ferramentas de software de gerenciamento de recursos.

A classificação hierárquica é mostrada na Figura 3 e a seguir são feitas algumas discussões tanto sobre a taxonomia hierárquica quanto sobre a plana.

Taxonomia Hierárquica.

Local & Global: as diferenças entre o escalonamento local e global propostas por Casavant; Kuhl (1988) já foram apresentadas no início deste trabalho. Lendo-se essa taxonomia, percebe-se que não houve nenhum comentário a respeito de um escalonamento global integrado com o escalonamento local, sendo citado apenas que o escalonamento global e local deveriam ficar separados, com o escalonamento local a cargo do sistema operacional. Não houve nessa classificação, nenhuma preocupação maior com uma sincronização mais rigorosa entre os diferentes processos pertencentes à aplicação. Essa preocupação com a sincronização é amplamente discutida nos trabalhos que envolvem a computação paralela.

Estático & Dinâmico: as políticas de escalonamento estático e dinâmico pertencem ao escalonamento global e indicam o tempo em que o escalonamento ou as decisões de atribuições são feitas e também a quantia de informação que se dispõe para o escalonamento.

A característica estática ou dinâmica de um escalonamento é uma das mais importantes e destacadas pela literatura, embora não haja um consenso a respeito desses termos. No escalonamento estático, o máximo de informações a respeito dos processos e dos processadores devem estar disponíveis em tempo de compilação/"linkedição". Assim, quando

a aplicação for executada o escalonamento já está determinado e não mudará em tempo de execução.

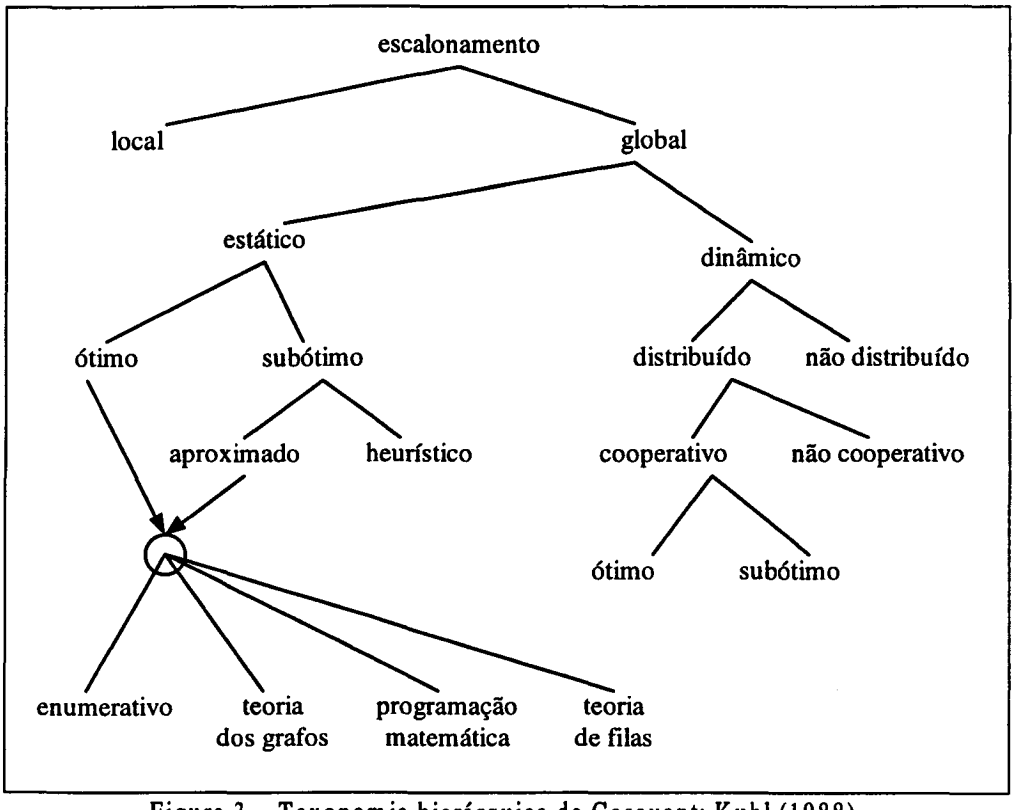

Figura 3 - Taxonomia hierárquica de Casavant; Kuhl (1988).

O escalonamento estático é integrado à aplicação, onde, no máximo durante a compilação/*linkedição* da mesma é que o escalonamento é determinado. A expressão "no máximo" indica que o escalonamento estático pode ser determinado ou pelos responsáveis pelo desenvolvimento da aplicação paralela, durante a sua construção e antes da compilação; ou então por uma ferramenta de software que analise sintática e semanticamente o código desenvolvido durante a fase de compilação.

As principais vantagens do escalonamento estático, também conhecido como determinístico, *off-line* ou de tarefa, são: (1) ausência de sobrecarga para determinar o escalonamento em tempo de execução e (2) maior controle da aplicação em plataformas monousuárias, em função da grande quantidade de informações necessárias (tanto de hardware quanto de software) para que esse tipo de escalonamento atinja seus objetivos satisfatoriamente. As desvantagens do escalonamento estático ficam por conta da sua falta de: flexibilidade, portabilidade, capacidade escalar (*scalability*), tolerância a falhas, adequabilidade às plataformas multiusuárias e também pela exigência de um alto

conhecimento prévio a respeito do comportamento da aplicação em relação à plataforma usada. Shirazi et al. (1995a) também citam que o escalonamento estático possui os seguintes problemas: (1) as políticas estáticas freqüentemente ignoram a distribuição dos dados entre os processos e (2) os mecanismos estimam os tempos de execução e comunicação não são eficientes, causando sérias degradações no desempenho esperado.

O escalonamento dinâmico é feito em tempo de execução da aplicação e possui muito pouco ou nenhum conhecimento prévio sobre as necessidades dos processos que serão executados, nem tão pouco da plataforma utilizada. Essa classe de escalonamento é amplamente pesquisada e possui inúmeras implementações, principalmente em função das suas vantagens, em relação ao escalonamento estático que são, potencialmente: flexibilidade, portabilidade, tolerância a falhas, capacidade escalar e adequação às plataformas multiusuárias. Considerando as plataformas multiusuárias como as NOWs, o escalonamento dinâmico é necessário para se atingir um desempenho melhor. Além da dificuldade de se antever o impacto causado pelos possíveis usuários simultâneos, essas plataformas computacionais podem ser alteradas mais facilmente quando comparadas, por exemplo, com os computadores paralelos. Assim, quando houver alterações na plataforma usada, o escalonamento pode, dinamicamente, refletir essas mudanças. A maior desvantagem do escalonamento dinâmico é a sobrecarga gerada, devido às decisões tomadas sobre o escalonamento em tempo de execução.

Ótimo & Subótimo: um escalonamento ótimo pode ser realizado quando se possui todas as informações necessárias a respeito dos processos que serão escalonados, assim como da plataforma computacional utilizada. Entretanto, a geração de uma solução ótima é, na maioria dos casos, um problema NP-completo, inviabilizando, portanto, a sua realização. Segundo Shirazi et al. (1995a) escalonamentos ótimos são possíveis apenas em casos específicos, como por exemplo quando o tempo de execução de todos os processos é o mesmo e apenas dois processadores são utilizados. Uma alternativa para o problema NP-completo das soluções ótimas são as soluções subótimas, as quais não se prendem apenas à melhor solução, ficando satisfeitas se "um bom escalonamento" for realizado. O problema aqui é determinar o que é "um bom escalonamento" e qual métrica avalia a solução encontrada.

Aproximado & Heurístico: o escalonamento aproximado utiliza algum modelo computacional formal para o algoritmo, porém, tem por objetivo alcançar um escalonamento aceitável, ao invés de buscar a melhor solução para o escalonamento. O escalonamento

heurístico, por sua vez, utiliza parâmetros "especiais" os quais afetam a plataforma computacional de maneiras indiretas. Normalmente os parâmetros estão relacionados indiretamente com o desempenho e são mais simples para monitorar e calcular. Casavant; Kuhl (1988) citam como um exemplo de heurística grupos de processos I/O-Bound que são escalonados no mesmo processador e grupos de processos CPU-Bound que são escalonados em processadores distintos. Essa heurística diminui diretamente a sobrecarga envolvida na troca de informações entre os processos, enquanto se beneficia do paralelismo. A intuição indica que esse escalonamento melhorará o desempenho da plataforma computacional, mas afirmar que existe um relacionamento direto entre a política empregada e os resultados desejados é uma tarefa um pouco mais complexa.

Ótimos & Subótimos Aproximados: independente se a política de escalonamento é ótima ou uma subótima aproximada, há quatro categorias básicas para os algoritmos de escalonamento: (1) enumeração do espaço da solução e busca, (2) teoria dos grafos, (3) programação matemática e (4) teoria das filas.

Distribuído & Não Distribuído: quando um escalonamento dinâmico é realizado, a responsabilidade por essa atividade pode ser apenas de um processador (escalonamento não distribuído) ou fisicamente distribuída entre os processadores (escalonamento distribuído). Casavant; Kuhl (1988) distinguem os termos distribuído e descentralizado, através de dois componentes: a responsabilidade e a autoridade. Quando a responsabilidade pelo escalonamento é compartilhada entre os diferentes processadores, o escalonamento é distribuído. Quando a autoridade para fazer o escalonamento é distribuída entre esses processadores, o escalonamento é descentralizado. As plataformas que possuem autoridade descentralizada devem possuir a responsabilidade distribuída, no entanto, é possível distribuir responsabilidade para, por exemplo, coletar informações e efetuar determinadas políticas, sem fornecer a autoridade necessária para mudar nenhuma decisão já tomada nem tão pouco realizar futuras decisões.

Cooperativo & Não Cooperativo: o escalonamento distribuído pode ser dividido considerando o grau de autonomia que cada processador possui para determinar como o escalonamento deve ser feito. No caso do escalonamento não cooperativo, um processador toma as suas decisões de maneira independente das ações dos outros elementos de processamento, não se preocupando com os efeitos das suas decisões sobre o restante da plataforma. No escalonamento cooperativo cada processador é responsável por realizar a sua

porção do escalonamento, onde todos os processadores agem seguindo um único objetivo global. Assim, com o escalonamento cooperativo, os processadores não se preocupam apenas em melhorar o seu desempenho local, mas sim o desempenho do sistema como um todo.

Taxonomia Plana:

Adaptável & Não Adaptável: quando os algoritmos e/ou parâmetros usados para implementar a política de escalonamento mudam dinamicamente em função do comportamento atual da plataforma computacional, diz-se que essa política é adaptável. Uma solução "não adaptável" é aquela onde a política de escalonamento, a qual pode ser dinâmica, considera sempre as mesmas regras e os mesmos parâmetros para realizar o escalonamento. Toda solução adaptável é uma solução dinâmica, mas uma solução dinâmica não necessariamente é adaptável. Uma solução dita dinâmica percebe as mudanças que estão ocorrendo na plataforma computacional em tempo de execução e emprega o seu algoritmo para escalonar os processos. Uma solução adaptável vai além e modifica a sua própria política de escalonamento em função das variações ocorridas na plataforma. Casavant; Kuhl (1988) citam como exemplo de políticas adaptáveis aquelas que permitem alterar o peso dado para cada parâmetro obtido da plataforma, em função de algumas variáveis como: classe da aplicação, grau de utilização da plataforma, entre outros.

Balanceamento de Carga: os autores colocam o balanceamento de carga como uma possível política para o escalonamento, no mesmo nível de características como: adaptável, migração de processos, entre outros. Apesar dessa taxonomia ser possível, preferiu-se neste trabalho designar o balanceamento de cargas como um objetivo do escalonamento e não como uma característica de implementação. A questão aqui é como o escalonamento é feito e não o que o escalonamento faz. Nesse item, balanceamento de carga, Casavant; Kuhl (1988) apontam algumas questões relacionadas, tais como: (1) que o balanceamento é mais adequado aos processadores homogêneos (fato já defasado), (2) que há a necessidade de uma troca de informações entre os processadores, normalmente feita periodicamente ou sob demanda (não foi citada a troca de informação por mudança de estado) e (3) a necessidade de uma métrica para designar a carga nos processadores (um índice de carga).

***Bidding* (Licitação)**: o escalonamento através de licitação é feito por uma negociação (troca de informações) entre os processadores que possuem processos para serem executados (remotamente) e processadores que estão aptos a executarem processos. Casavant; Kuhl (1988) citam como exemplo um escalonamento formado por um processador gerente

(*manager*) e por processadores contratantes (*contractors*). O gerente representa o processador que possui processos que serão ofertados para execução. Os contratantes representam os processadores aptos a receberem os processos. Após receberem o pedido para execução, os contratantes oferecem-se para executar os processos solicitados, indicando as suas características locais. Dessa maneira, o gerente escolhe a melhor opção de processador no momento, ao mesmo tempo que permite a completa autonomia dos contratantes (participam da negociação apenas os processadores que estiverem aptos).

Probabilístico: essa política tem por objetivo reduzir a sobrecarga existente com a geração de todas as possíveis soluções para o escalonamento de processos. A política probabilística realiza o escalonamento dos processos "randomicamente" (*randomly*), segundo alguma distribuição. Executando essa política repetidamente, diferentes escalonamentos são gerados, os quais, analisados posteriormente, permitem que o melhor escalonamento seja escolhido dentre aqueles gerados.

Atribuição Inicial & Reatribuição Dinâmica: o escalonamento feito apenas através de uma atribuição inicial (não preemptivo ou sem migração), é aquele que considera apenas uma única vez questões como onde e quando os processos devem ser executados. Para os autores, esse escalonamento deve considerar apenas as informações fornecidas pela aplicação, como: tempo estimado para execução ou demanda por recursos. No entanto, essa limitação não é apontada por outros autores, onde, mesmo com apenas o escalonamento inicial podem ser consideradas questões outras, tais como: a carga atual de um processador e as diferenças relativas às potências computacionais. O escalonamento feito com reatribuição dinâmica (preemptivo ou com migração de processos), por sua vez, tem a possibilidade de corrigir uma "atribuição inicial" já realizada, a qual, por algum motivo, tornou-se incorreta. A correção, através da transferência do processo de um processador para outro, possui uma sobrecarga alta quando comparada com o escalonamento não preemptivo e deve ser bem parametrizada para que se consiga avaliar os ganhos reais com a migração. Todo o escalonamento que realiza a "migração de processos" é um escalonamento dinâmico, mas nem todo escalonamento dinâmico realiza a migração de processos.

A questão da migração de processos é uma das mais complexas dentro da área do escalonamento, com várias propostas já desenvolvidas. Sozinha, a migração de processos constitui-se uma ampla linha de pesquisa, por ser um dos recursos mais eficientes que o

escalonamento dinâmico possui para se adaptar às possíveis variações em uma plataforma computacional multiusuária e tolerante a falhas.

Vantagens & Desvantagens: a taxonomia proposta por Casavant; Kuhl (1988) é uma das mais completas, por apresentar e agrupar inúmeros termos espalhados pelos trabalhos. Em relação à taxonomia hierárquica, a proposta é coerente e destaca-se das demais porque, apesar de existirem outras poucas classificações hierárquicas, essa estabelece um relacionamento mais coerente entre termos muitas vezes conhecidos. A taxonomia plana é válida no sentido de agregar características adicionais aos trabalhos propostos, sendo que essas são encontradas em diferentes níveis dentro da taxonomia hierárquica.

A proposta de Casavant; Kuhl (1988) apresenta alguns problemas que não comprometem o mérito do trabalho. Os autores utilizam os termos mecanismo e política de escalonamento como sinônimos, o que sob a ótica deste trabalho e de outros autores como Tanenbaum (1992) e Song et al. (1997), devem representar atividades distintas.

Em alguns pontos houve uma limitação tecnológica da taxonomia, como por exemplo no item "balanceamento de carga". Nesse item é afirmado que o balanceamento é mais eficaz quando os processadores são homogêneos, em função de todos os processadores saberem a estrutura e potência dos outros. Isso poderia ser uma realidade em 1988, porém, com as plataformas heterogêneas atuais essa afirmação está defasada.

O autores omitiram certas características, como por exemplo, quando citam como políticas para troca de informação entre os processadores, apenas as políticas periódica e sob demanda, quando na realidade existe também a por mudança de estado.

No item que aborda a migração de processos, os autores afirmam que para o escalonamento sem migração (chamado como atribuição inicial) as únicas informações possíveis de serem usadas para a tomada de decisão devem ser as informações fornecidas pela aplicação através, por exemplo, do usuário. Assim, não devem ser consideradas outras informações sobre as características atuais da plataforma, como o nível de utilização dos processadores. Essa afirmação não é válida, porque é possível ter-se um escalonamento sem migração e que considera a situação da plataforma no momento do escalonamento.

A taxonomia plana apresenta um número muito reduzido de termos. Podem ser incluídos alguns itens nessa taxonomia, como: quem é o responsável por iniciar o escalonamento (processadores transmissores, receptores ou iniciado simetricamente), se o escalonamento é feito no sistema operacional ou não, quanto há transparência para a aplicação, entre outros.

Falta estabelecer um relacionamento entre a taxonomia apresentada nesse trabalho com outras facilmente encontradas na literatura, como a taxonomia *space sharing* e *time sharing*. Mesmo com os autores citando que é possível adaptar outras propostas, as quais enfocam mais o gerenciamento de recursos que o escalonamento de processos, essa analogia não é trivial e não foi feita no trabalho.

2.5.2. Taxonomia de Wang; Morris (1985)

Wang; Morris (1985) propõem uma taxonomia direcionada apenas ao compartilhamento de carga em sistemas distribuídos locais, apresentando algumas limitações como: o uso de plataformas homogêneas, execução de processos independentes, seqüenciais e que, uma vez iniciados não podem ser suspensos e transferidos de processador (sem migração).

Antes de apresentarem a taxonomia, os autores definem o que são algoritmos de escalonamento locais, globais, dinâmicos, estáticos e distribuídos de maneira semelhante a que será usada posteriormente por Casavant; Kuhl (1988).

A taxonomia divide os algoritmos de escalonamento através de duas características. A primeira distinção é em relação a quem tem a iniciativa de iniciar o escalonamento. Para tanto, os processadores foram classificados ou como origem ou servidores (Figura 4). Os processadores origem são aqueles que possuem processos para serem executados, enquanto os processadores servidores representam aqueles que desejam executar os processos remotos. Um processador pode ser simultaneamente origem e servidor, também conhecido como simétrico [Shivaratri et al. (1992)]. Se um processador origem é quem toma a iniciativa de procurar onde executar um processo, essa política é classificada como "iniciada pela origem" (*source-initiative*). Se um processador servidor é quem toma a iniciativa de "procurar por trabalho" (processos), essa política é classificada como "iniciada pelo servidor" (*server-initiative*).

A segunda distinção refere-se à dependência por informação que a política de escalonamento possui. Essa distinção indica a quantidade de informação que um processador origem possui (ou necessita) sobre os processadores servidores e vice-versa. Para determinar a quantidade de dependência por informação os autores criaram sete níveis (1-7), onde o número "um" representa a menor e o número "sete" a maior dependência. Para Wang; Morris (1985) no nível um estariam as políticas de escalonamento estáticas e a partir do nível cinco estariam as políticas de escalonamento dinâmicas. Entre o nível um e o nível cinco estariam

as políticas "semidinâmicas", ou híbridas, conforme preferem alguns autores como Shirazi et al. (1995a).

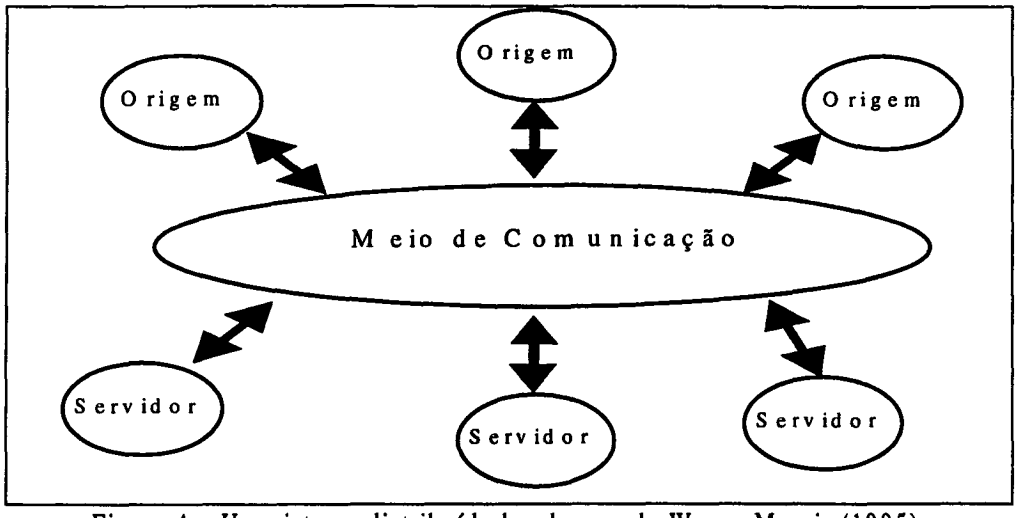

Figura 4 - Um sistema distribuído local segundo Wang; Morris (1985).

Mesmo com o uso de alguns termos semelhantes, as taxonomias de Wang;Morris (1985) e de Casavant; Kuhl (1988) possuem abordagens diferentes. As principais diferenças estão na (1) relação hierárquica, feita apenas por Casavant; Kuhl (1988), (2) na preocupação por quem inicia o escalonamento (incluída apenas em Wang; Morris (1985) e (3) na maior limitação da taxonomia de Wang; Morris (1985) a qual é voltada exclusivamente para os sistemas distribuídos.

Vantagens & Desvantagens: As grandes vantagens desse trabalho ficam por conta da utilização de termos conhecidos e pela facilidade de compreensão. A taxonomia proposta por Wang; Morris(1985) é uma das mais antigas, onde, em função da sua simplicidade e limitações, ela é pouco referenciada em outros trabalhos. Apesar de ser anterior à taxonomia proposta por Casavant; Kuhl (1988) e de utilizar termos semelhantes (e conhecidos), outros autores preferem citar esta última, provavelmente em função da sua maior abrangência.

2.5.3. Taxonomia de Baumgartner; Wah (1991)

O trabalho de Baumgartner; Wah (1991) apresenta um levantamento sobre as pesquisas em algoritmos de escalonamento, analisa algumas taxonomias propostas até então (entre elas Casavant; Kuhl (1988) e Wang;Morris(1985)) e propõe uma nova taxonomia para a área.

A primeira distinção feita é entre os escalonamentos determinísticos, não determinísticos e indecidíveis. O escalonamento determinístico tem, previamente, todas as informações necessárias para o escalonamento. Um escalonamento é determinístico se ele pode ser completamente definido e também se é possível de ser feito em função do tempo (processamento) e espaço (memória e disco). O escalonamento não determinístico pode ser completamente definido, entretanto, encontrar uma solução exata leva muito tempo ou necessita muito espaço ou ambos. Um escalonamento indecidível é aquele impossível de ser realizado, mesmo que haja infinitas capacidades de processamento e recursos de armazenagem.

Baumgartner; Wah (1991) citam que na década de 60 e início de 70 havia um grande interesse sobre o escalonamento em "ambientes de manufatura", os quais eram predominantemente determinísticos, com um conhecimento completo sobre o tempo de chegada e de atendimento. Em sistemas de computação o escalonamento deve ser visto como não determinístico, em função de raramente haver uma informação exata sobre os processos, plataforma e objetivos que devem ser considerados.

Para apresentar a nova taxonomia (chamada ESR), os autores discutem sobre dois grupos de taxonomias. O primeiro grupo relaciona algumas taxonomias existentes até então, abordando tanto escalonamentos determinísticos quanto não determinísticos. O enfoque desse primeiro grupo é dado para os algoritmos não determinísticos. O segundo grupo é voltado para o escalonamento determinístico e é a base para a taxonomia proposta nesse artigo, porque, segundo os autores, as taxonomias do primeiro grupo apresentam limitações no que tange os determinísticos.

Considerando o primeiro grupo de taxonomias, a Tabela 1 agrupa seis características dos algoritmos de escalonamento. A Figura 5 demonstra graficamente o relacionamento entre os termos apontados na Tabela 1.

O "nível do escalonamento" distingue o escalonamento em intraprocessador interprocessador. Esses termos correspondem, respectivamente aos escalonamentos local e global, destacados na taxonomia de Casavant; Kuhl (1988).

A "flexibilidade das normas" pode ser estática ou dinâmica e refere-se à flexibilidade inserida nas regras do escalonamento para reagir ao estado atual da plataforma computacional. Os termos estático e dinâmico são iguais aos usados por Casavant; Kuhl (1988).

A "localização do controle" é dividida em distribuída, hierárquica e centralizada e descreve como a responsabilidade pelas decisões do escalonamento é repartida entre os processadores pertencentes à plataforma. Não há aqui a mesma distinção feita por Casavant; Kuhl (1988) entre distribuído e decentralizado.

Tabela 1 - Revisão das taxonomias de políticas de escalonamento [Baumgartner; Wah (1991)].

Terminologia	
Característica	Valores
Nível do Escalonamento	Intraprocessador, Interprocessador
Flexibilidade das Normas	Estático, Dinâmico
Localização do Controle	Distribuído, Hierárquico, Centralizado
Cooperação	Negociação, Independente
Iniciação	Transmissor, Receptor, Ambos
Adaptabilidade	Adaptável, Não Adaptável

A "cooperação" descreve a quantia de interação realizada entre os processadores, podendo ser com negociação ou independente. Esses termos correspondem, respectivamente, aos termos cooperativos e não cooperativos da taxonomia de Casavant; Kuhl (1988). Wang; Morris (1985) também destacam essa característica na sua taxonomia.

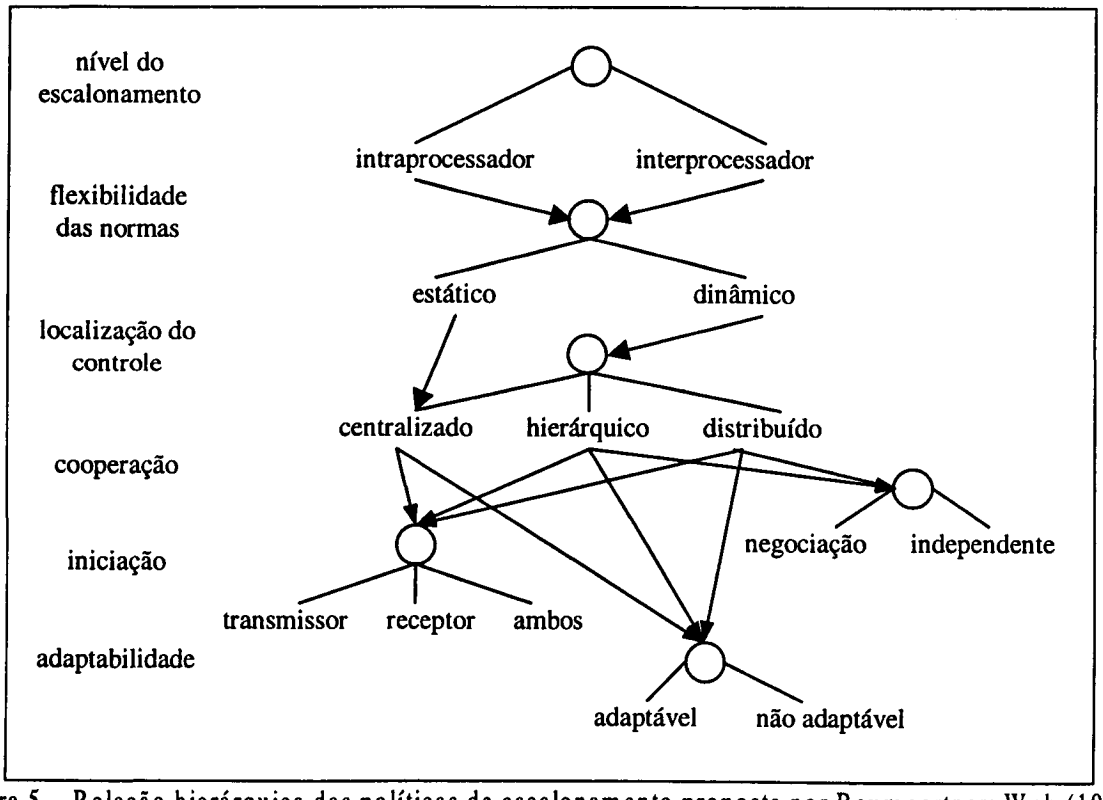

Figura 5 - Relação hierárquica das políticas de escalonamento proposta por Baumgartner; Wah (1991).

A "iniciação" destaca o algoritmo de escalonamento através de quem é o responsável por iniciar o escalonamento. Pode ser iniciado pelo transmissor (ou origem, sobrecarregado), pelo receptor (ou destino, ocioso, servidor) ou ambos. Essa característica é encontrada em várias taxonomias, dentre elas a de Wang; Morris (1985).

A "adaptabilidade" refere-se à mesma característica apontada por Casavant; Kuhl (1988), referindo-se à flexibilidade do algoritmo em mudar a sua política de atuação, baseado na situação atual da plataforma computacional usada.

Considerando o segundo grupo de taxonomias (voltado para taxonomias de escalonamentos determinísticos) os algoritmos são classificados por características como: processos (α), ambiente de execução (β) e objetivos (γ). Essas características podem ser subdivididas em outros componentes permitindo, assim uma maior especificação dos algoritmos que estão sendo classificados. Esse esquema é a base para a nova taxonomia apresentada pelos autores, onde foram acrescentados alguns itens que permitem o relacionamento de algoritmos com características estocásticas e não apenas determinísticas.

ESR – a nova proposta de taxonomia: A nova proposta de taxonomia é chamada de ESR (*Event, Environment (Surroundings) and Requirements*) e está apresentada na Tabela 2.

A primeira categoria "E" corresponde aos eventos. Os eventos nada mais são que as entidades que serão escalonadas, como por exemplo os processos. Considerando o relacionamento entre os eventos, estes podem ser: completamente independentes, com alguma ordem de precedência entre eles ou com a necessidade de trocar informações durante a execução. Em função da disponibilidade (taxa de chegada) os eventos podem ser: (1) estáticos, onde os processos chegam simultaneamente antes da sua execução, (2) periódicos, onde os processos chegam em intervalos conhecidos, mas distantes o suficiente para permitir que os processos já escalonados terminem a sua execução e (3) estocásticos, onde a chegada dos processos é determinada por uma distribuição de probabilidade. Em função da necessidade por recursos, os processos podem ser: (1) determinísticos, onde as futuras solicitações por recursos são especificadas como constantes, ou então (2) estocásticos, onde essas solicitações por recursos são regidas por alguma distribuição de probabilidade.

A segunda categoria "S" [*Environment (Surroundings)*] representa todas as demais características relativas à plataforma computacional, as quais afetam o desempenho do escalonamento. O número de recursos (processadores) pode ser: "um" indicando o

escalonamento em apenas um processador (local ou intraprocessador), *k* (um número específico) indicando uma quantidade exata de recursos, ou *n* (um número arbitrário) indicando uma quantidade variada de recursos envolvidos. A classe de recursos indica a heterogeneidade contida na plataforma computacional: uma classe indica recursos homogêneos e mais de > uma classe indica recursos heterogêneos. As características físicas das classes de recursos podem ser especificadas, se elas são decisivas para o escalonamento. A sobrecarga de comunicação, embora pudesse ser considerada uma das características das classes de recursos, indicam o peso da comunicação na plataforma utilizada. A comunicação pode ter um custo "zero", pode ser constante (determinística) ou então ser variada (estocástica). Os mecanismos de comunicação indicam como é feita a comunicação entre os eventos escalonados, os quais podem ser: fluxo de eventos, *broadcast* de informação, passagem de mensagem ou qualquer outra forma de comunicação.

Tabela 2 - Taxonomia ESR proposta por Baumgartner; Wah (1991).

Categoria	Atributo	Valores
Evento (E)	Relação entre os eventos	Independente, precedente, comunicação
	Disponibilidade (taxa de chegada)	estática, periódica, estocástica
	Necessidades por recursos	determinística, estocástica
Ambiente (S)	Número	1, k, n
	Classes de recursos	recursos homogêneos (1 classe) recursos heterogêneos (< 1 classe)
	Características físicas	velocidade, tamanho da memória principal, capacidades especiais, etc. topologia
	Sobrecarga de comunicação	nenhuma, determinística, estocástica
	Mecanismo de comunicação	fluxo de eventos, broadcast, passagem de mensagem, outros
Objetivo (R)	Desempenho	qualquer solução, *deadlines (real time)*, solução boa (com melhoria) e solução ótima.

A terceira e última categoria "R", representa os requisitos do escalonamento, chamados neste trabalho de objetivos. O objetivo básico e mais simples é encontrar qualquer solução que execute os processos. Outros objetivos podem ser: redução do tempo de execução, executar um processo dentro de um prazo máximo (*deadline*) ou qualquer outra métrica de desempenho.

Vantagens & Desvantagens: a principal vantagem da taxonomia ESR é a sua capacidade de representar os algoritmos de escalonamento de uma maneira flexível. Dessa forma, a taxonomia pode conter maiores detalhes ou então ater-se apenas a detalhes mais genéricos (capacidade de expandir ou diminuir a taxonomia). Outra vantagem é a preocupação com o "como" o escalonamento é feito e não com "o que" ele deve resolver. As desvantagens ficam por conta do relacionamento entre as diferentes propostas (não há uma relação hierárquica entre as características da Tabela 2). Com a Tabela 1 e com a Figura 5, as quais representam não uma nova taxonomia, mas sim uma revisão de outras propostas, é possível estabelecer esse relacionamento entre diferentes algoritmos. Analisando-se a Figura 5 pode-se concluir, erroneamente, que haja um escalonamento interprocessador, estático, centralizado e adaptável (para um escalonamento ser adaptável ele deve ser dinâmico e não estático); ou então um escalonamento intraprocessador, dinâmico, distribuído e com negociação (para um escalonamento ser distribuído ele deve ser interprocessador e não intraprocessador). O relacionamento ilustrado pela Figura 5 deveria limitar essas e outras possíveis conclusões errôneas ou então chamar a atenção do leitor para que determinadas conclusões não fossem feitas.

2.5.4. Taxonomia de Lüling et al. (1993)

Lüling et al. (1993) introduzem uma nova taxonomia para algoritmos de escalonamento dinâmicos com o objetivo exclusivo de balanceamento de carga. Os termos estático, dinâmico, distribuído e adaptável são (apenas) utilizados de maneira análoga aos utilizados por Casavant; Kuhl (1988).

Para os autores, as taxonomias propostas até então, foram consideradas complexas para uma comparação genérica entre os algoritmos para balanceamento de carga propostos. Assim sendo, os algoritmos de escalonamento foram separados em duas partes: uma parte de decisão e outra de migração. Como o próprio nome diz, a parte de decisão é "responsável pela decisão de migrar a carga de trabalho". A parte de migração é a responsável por enviar a carga para outro processador, com o intuito de balancear a plataforma computacional usada.

A parte de decisão pode ser baseada na situação local do processador e/ou no máximo dos seus vizinhos (decisão local) ou então depender de qualquer subconjunto dos processadores (decisão global). A parte de migração pode enviar cargas apenas para seus vizinhos diretos

(migração local) ou então enviar essas cargas para quaisquer processadores da plataforma (migração global).

Essa distinção entre local e global é a base da taxonomia proposta por Lüling et al. (1993). Outra característica inserida nessa taxonomia (com menor ênfase) é a característica de quem inicia a atividade de escalonamento (transmissor (t), receptor (r) ou combinado (tr)), conforme já abordada nas taxonomias de Wang; Morris(1985) e Baumgartner; Wah (1991). A Figura 6 mostra graficamente o relacionamento entre as partes de decisão e de migração (locais e globais), assim com os responsáveis pela iniciativa do escalonamento.

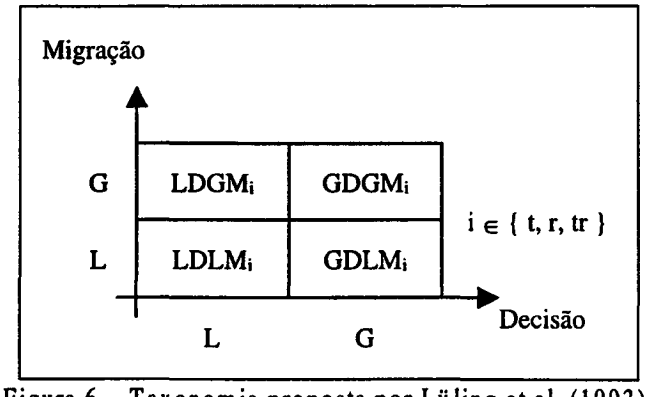

Figura 6 - Taxonomia proposta por Lüling et al. (1993).

Se os termos decisão e migração fossem substituídos por políticas de escalonamento a taxonomia proposta permaneceria a mesma. A parte de decisão compreende um conjunto de políticas decidindo *onde* obter informação, *quando* executar uma transferência e *qual* carga (processo) transferir, respectivamente políticas de informação, transferência e de seleção. A parte de migração envolve, além dos mecanismos responsáveis pela transferência da carga em si, a política de localização, encarregada de encontrar parceiros para a transferência da carga (*onde* transferir).

Vantagens & Desvantagens: um dos fatores que diferenciam positivamente a taxonomia proposta é a preocupação com a topologia da plataforma na classificação dos algoritmos de escalonamento. A topologia é uma característica decisiva, principalmente em computadores paralelos cujos processadores não conseguem ter acesso a todos os outros processadores em apenas um "salto".

Alguns dos problemas dessa taxonomia são: apresentar termos novos para designar atividades já conhecidas, como é o caso dos termos "parte de decisão e parte de migração", e também definir com significados diferentes, termos já utilizados em outras taxonomias, como é o caso de "local" e "global", usados por exemplo em Casavant; Kuhl (1988) e Wang; Morris

(1985). A simplicidade da taxonomia exclui algumas características importantes como, por exemplo, algoritmos de escalonamento estáticos, adaptáveis, determinísticos, entre outros. Taxonomias como essa são difíceis de serem relacionadas e dificultam uma comparação de maior qualidade entre propostas de escalonamento já feitas.

2.5.5. Taxonomia de Xu; Lau (1997)

Xu; Lau (1997) utilizam o termo balanceamento de carga como sinônimo de escalonamento de processos em todo o trabalho, portanto, as políticas de escalonamento apresentadas aqui têm o objetivo de balancear a carga de trabalho da plataforma computacional.

Antes de apresentarem a taxonomia, os autores também citam que o escalonamento possui algumas "questões chave" (*key issues*). Dentre essas questões aparece a regra de iniciação, a qual pode ser iniciada pelo transmissor, pelo receptor, simetricamente ou ainda periodicamente. As três primeiras (transmissor/receptor/simétrica) foram apresentadas de maneira semelhante aos trabalhos anteriores. A iniciação periódica (ou remapeamento) induz todos os processadores a participarem de um reescalonamento periódico com a intenção de atingir o objetivo do escalonamento.

Para fundamentar a taxonomia, os autores definem o "domínio do escalonamento", o qual representa o conjunto de processadores envolvidos no escalonamento dinâmico. O domínio do escalonamento pode ser global ou local. O domínio global permite que todos os processadores pertencentes à plataforma façam parte do escalonamento, enquanto o domínio local restringe o escalonamento apenas a um subconjunto de processadores da plataforma computacional. O domínio global ou local indica não somente quais são os possíveis parceiros para o escalonamento, mas também determina a quantia de informação trocada entre os processadores. Dessa forma, há duas questões envolvidas no domínio do escalonamento: as regras de localização de um parceiro para o escalonamento e as regras para a troca de informações. Algoritmos de escalonamento que trocam informações e procuram parceiros apenas no domínio local são conhecidos como algoritmos de escalonamento dinâmicos baseados nos vizinhos mais próximos. Os algoritmos baseados nos vizinhos mais próximos também são conhecidos como iterativos (repetitivos), porque transferem processos de um processador para seus vizinhos e assim sucessivamente até atingir os objetivos propostos para o escalonamento. Cada salto feito pelos processos é determinado por uma decisão local, no

processador transmissor dos processos. Assim, em um escalonamento iterativo o processador origem (aquele que estava inicialmente com os processos) não sabe qual será o processador destino dos processos, por não fazer essa atribuição (origem/destino) diretamente.

Os algoritmos de escalonamento dinâmicos que localizam seus parceiros e que utilizam informações através do domínio global, são denominados algoritmos de escalonamento globais (ou diretos). O termo direto deve-se ao fato desses algoritmos determinarem o destino dos processos diretamente entre o processador transmissor e o receptor.

Segundo os autores, os algoritmos globais são mais indicados em plataformas que possuem *broadcast* ou políticas de escalonamento centralizadas, em função do custo para determinar e para distribuir de maneira eficiente a carga de trabalho de todos os processadores. Já os algoritmos iterativos são mais flexíveis, onde a cada salto dos processos uma nova decisão sobre o escalonamento é tomada por um processador. Os algoritmos iterativos também são mais apropriados para plataformas que possuem a rede de comunicação direta (ou ponto-a-ponto) e com o roteamento do tipo *store-and-forward*, devido ao fato de cada processador receber a incumbência de executar processos e ainda assim poder enviá-los para outro processador, se for o caso.

Xu; Lau (1997) citam que há na literatura várias taxonomias sobre algoritmos de escalonamento em plataformas de computação distribuída baseadas em redes locais, fazendo referência aos trabalhos de Baumgartner; Wah (1991), Casavant; Kuhl (1988) e Wang; Morris (1985). Ainda segundo os autores, esses trabalhos são incompletos por excluírem os computadores paralelos baseados em uma rede de conexão direta. Essa topologia é mais flexível na troca de informações, na localização de parceiros e na distribuição da carga de trabalho, em função do escalonamento estar "dividido" nos diferentes saltos realizados durante a escolha do melhor processador hospedeiro para os processos. Dessa forma, a inclusão de uma taxonomia envolvendo algoritmos de escalonamento dinâmicos iterativos e/ou diretos complementaria as taxonomias existentes.

A Figura 7 ilustra graficamente a taxonomia proposta pelos autores, onde, os itens "Vizinho mais Próximo (iterativo)" e "global (direto)" já foram comentados. Os algoritmos determinísticos e estocásticos indicam quais serão as regras de distribuição aplicadas ao escalonamento. Os algoritmos de escalonamento iterativos determinísticos são caracterizados por possuírem regras predefinidas, as quais determinam para qual(is) processador(es)

vizinho(s) a carga de trabalho deve ser transferida, além de também determinar qual a quantia dessa carga deve ser transferida. Os algoritmos iterativos estocásticos têm por objetivo conduzir a plataforma utilizada a um estado que possua alta probabilidade de se atingir os objetivos propostos (no caso o balanceamento de carga), sem utilizar para isso regras e parâmetros predefinidos pela política de escalonamento. Essas definições de determinístico e de estocástico não são as mesmas citadas anteriormente por Casavant; Kuhl (1998) e Baumgartner; Wah (1991).

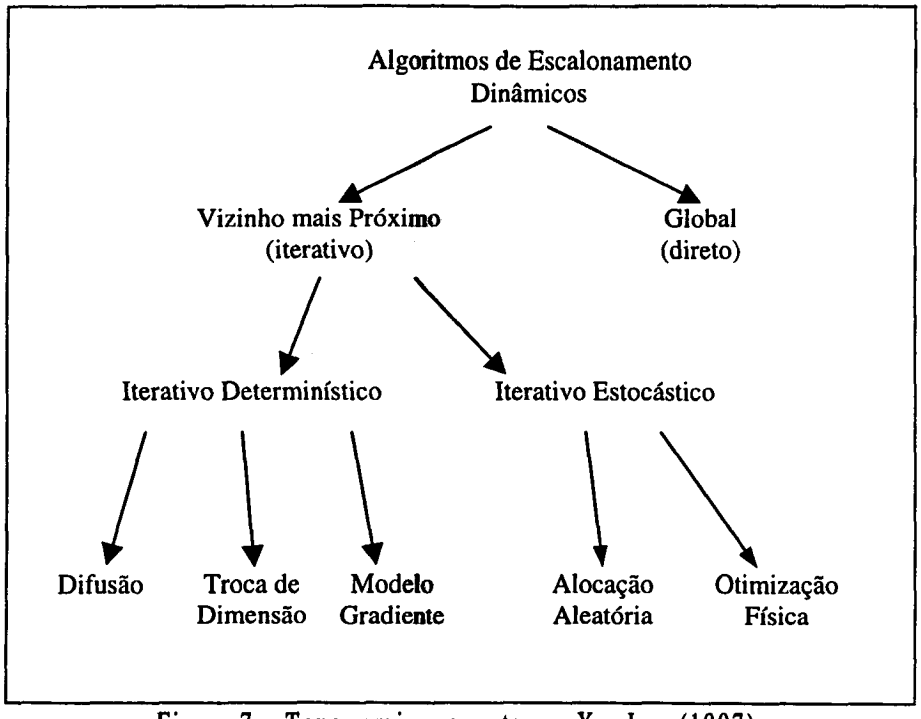

Figura 7 - Taxonomia proposta por Xu; Lau (1997).

Os três tipos de algoritmos de escalonamento pertencentes ao escalonamento iterativo e determinístico são: difusão (*diffusion*), troca de dimensão (*dimension exchange*) e modelo gradiente (*gradient model*). Os algoritmos por difusão e por troca de dimensão são parecidos, em função de examinarem todos os seus vizinhos em toda operação de escalonamento. Porém, um algoritmo por difusão permite que, em cada operação de escalonamento, um processador contate todos os seus vizinhos ao mesmo tempo, propagando porções da sua carga de trabalho para um ou mais processadores, enquanto solicita carga de trabalho de outros vizinhos. Já com o algoritmo troca de dimensão, um processador também realiza o escalonamento com seus vizinhos e entra em contato com apenas um processador vizinho por vez. Depois de uma operação de transferência e/ou recebimento de carga de trabalho com um vizinho, o

processador comunica a sua carga atual para o próximo vizinho e então uma nova distribuição de carga é iniciada. Maiores detalhes sobre a política de escalonamento baseada em difusão podem ser encontrados no trabalho de Corradi et al. (1997).

Os algoritmos de escalonamento baseados no modelo gradiente, propostos inicialmente por Lin; Keller (1987), restringem o escalonamento apenas ao vizinho que apresenta a menor carga de trabalho. Segundo Lüling et al. (1993) gradientes são vetores que possuem a menor distância (em saltos) entre o processador atual e um processador ocioso (com carga de trabalho baixa ou nenhuma), permitindo assim, que os processos a serem escalonados saibam a distância e saltem sobre os processadores até encontrarem esse processador ocioso.

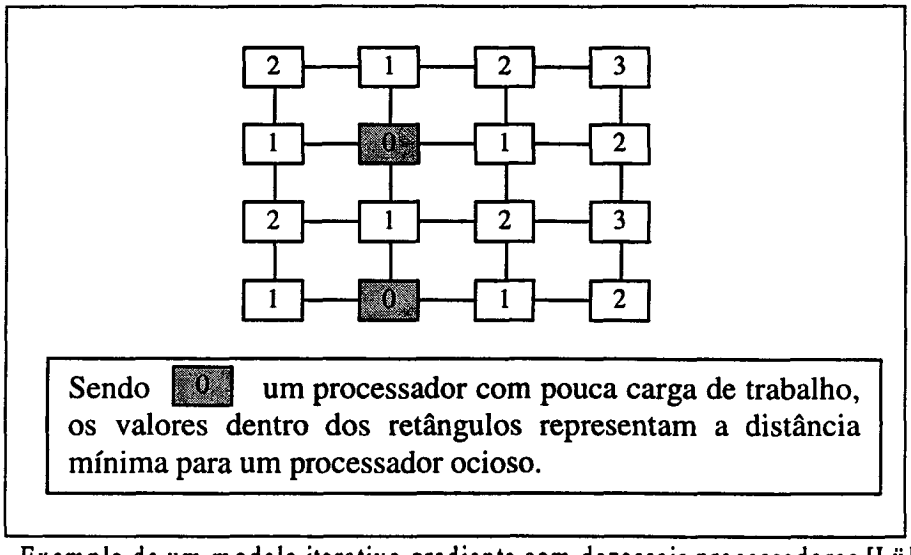

Figura 8 - Exemplo de um modelo iterativo gradiente com dezesseis processadores [Lüling et al. (1993)].

A Figura 8 demonstra graficamente uma superfície gradiente (*gradient surface*), a qual representa a menor distância de cada processador para o processador ocioso mais próximo. Para se determinar essa menor distância entre os processadores, a carga local é coletada por todos os processadores e enviada sistematicamente para os seus vizinhos mais próximos. Com isso, o escalonamento usando o modelo gradiente pode ser feito sucessivamente com o objetivo de se atingir um balanceamento de carga global para a plataforma.

Os dois tipos de algoritmos de escalonamento pertencentes ao escalonamento iterativo e estocástico são: alocação aleatória e otimização física. A alocação aleatória é a mais simples das duas políticas, onde os processos são transferidos para um processador vizinho escolhido de maneira arbitrária. Se o processador escolhido, determinar que ficou sobrecarregado após receber os processos, pode transferir os mesmos para outro processador vizinho, também

escolhido de maneira arbitrária. Os algoritmos pertencentes ao tipo "otimização física" são baseados em analogias com sistemas físicos. Segundo Xu; Lau (1997) os algoritmos de otimização física fazem o mapeamento do escalonamento de processos sobre sistemas físicos e então solucionam esse escalonamento através de simulação ou técnicas advindas da física experimental ou teórica. A grande vantagem desses algoritmos de otimização física sobre os que utilizam a alocação aleatória é que os primeiros melhoram o controle sobre a maneira aleatória da redistribuição dos processos.

Vantagens & Desvantagens: a taxonomia apresentada tem a vantagem de anexar a algumas taxonomias anteriores a questão da localização dos processadores, considerando também a topologia da plataforma computacional. Embora a topologia também tenha sido considerada pelo trabalho de Lüling et al. (1993), a taxonomia de Xu; Lau (1997) possui uma especificação maior e também uma relação hierárquica entre os diferentes tipos de algoritmos de escalonamento. A terminologia usada nas taxonomias de Lüling et al. (1993) e de Xu; Lau (1997) não divergem, fato observado com os termos "local" e "global" utilizados por ambas e com o mesmo significado. Algumas das desvantagens dessa taxonomia coincidem com as já apontadas no trabalho de Lüling et al. (1993). O fato da taxonomia ser apenas parcial, estando vinculada ao escalonamento dinâmico e detalhando somente o escalonamento iterativo, restringe a sua abrangência (mesmo com objetivos restritos de apenas complementar algumas taxonomias anteriores).

2.5.6. Taxonomia usada por Quinn (1994)

O livro de Quinn (1994) destaca três grupos de escalonamento feitos para a computação paralela: mapeamento em multicomputadores e em *arrays* de processadores, balanceamento de carga dinâmico em multicomputadores e escalonamento estático em multiprocessadores com acesso uniforme à memória.

O primeiro grupo, mapeamento em multicomputadores e *arrays* de processadores, refere-se ao problema de realizar o mapeamento prévio de uma aplicação paralela (representada por intermédio de um grafo) especificamente sobre uma plataforma computacional distribuída e com acesso à memória não uniforme. O acesso não uniforme à memória (NUMA – *NonUniform Memory Access*) indica que cada processador é capaz de obter dados locais muito mais rápido que se estivesse acessando dados vindo de uma memória remota (em outro processador). Assim, com informações detalhadas sobre o hardware e sobre a aplicação

paralela, as decisões sobre o escalonamento levam em consideração principalmente fatores como a localização dos dados, de modo que seja otimizado o acesso aos mesmos pelos processos escalonados. O escalonamento discutido para esse grupo, embora restrinja a plataforma utilizada, é semelhante ao definido como "escalonamento estático" por Casavant; Kuhl (1988), feito antes da execução.

O segundo grupo, balanceamento de carga dinâmico em multicomputadores, refere-se "à atividade de mudar a distribuição do trabalho entre os processadores durante a execução". Essa definição indica que esse grupo utiliza a migração de processos para mudar o escalonamento durante a execução, o que segundo a taxonomia de Casavant, Kuhl (1988) refere-se ao escalonamento dinâmico e com preempção (com migração).

Nesse segundo grupo Quinn (1994) classifica os algoritmos em: centralizados, completamente distribuídos e semidistribuídos. Os algoritmos centralizados foram definidos de maneira análoga à definição apresentada em Casavant; Kuhl (1988). Os algoritmos completamente distribuídos permitem que cada processador possua a sua própria imagem do estado da carga da plataforma, onde os processadores trocam informações e processos (carga de trabalho) com seus vizinhos mais próximos. Para a taxonomia de Xu;Lau (1997), apresentada anteriormente, esses algoritmos que trocam informações com seus vizinhos são chamados de iterativos.

Os algoritmos semidistribuídos dividem os processadores em grupos, onde cada grupo possui um coordenador que trata o escalonamento de maneira centralizada. Segundo os autores, um "mecanismo de escalonamento" de mais alto nível realiza o balanceamento entre os diferentes grupos.

Além da questão da distribuição da responsabilidade pelo escalonamento, esse segundo grupo também destaca a iniciação do escalonamento, a qual pode ser: iniciada pelo transmissor ou pelo receptor. As definições fornecidas para a iniciação do escalonamento são análogas as apresentadas anteriormente como por em Wang; Morris (1985) e Lüling et al. (1993).

O terceiro grupo de escalonamento é chamado de estático em multiprocessadores e com acesso uniforme à memória (UMA – *Uniform Memory Access*). Esses são os escalonamentos realizados antes da execução e em plataformas com memória compartilhada, onde todos os processadores têm acesso físico e lógico a mesma porção de memória principal. Como a velocidade de acesso à memória é considerada uniforme para todos os processadores, a

discussão desses algoritmos enfoca principalmente o tempo de execução dos processos escalonados e não a distribuição dos dados entre os processos, como foi o caso do primeiro grupo de escalonamentos.

A definição de "escalonamento estático", apresentada nesse terceiro grupo, mesmo restringindo a plataforma utilizada, também é semelhante à definição fornecida por Casavant; Kuhl (1988).

Vantagens & Desvantagens: embora a separação do escalonamento em três grupos apresentada por Quinn (1994) não tenha a pretensão de ser uma nova taxonomia para a área, ela induz a separação dos algoritmos de escalonamento dentro dos três grupos apresentados. A separação feita considera o hardware utilizado em multicomputadores e em multiprocessadores, fato que pode ser visto como uma vantagem.

Entretanto, a taxonomia usada não é devidamente situada dentro das demais já publicadas na área, impedindo assim um relacionamento direto desse trabalho com outros já existentes. A separação do escalonamento em: feitos para multicomputadores (primeiro grupo) e para multiprocessadores (terceiro grupo) não informa ao leitor que ambos são feitos antes da execução da aplicação, apenas utilizando políticas e recursos diferentes (grafos, gráfico de Gantt ou outro) para decidir onde escalonar os processos. A utilização do termo balanceamento de carga dinâmico no segundo grupo, novamente confunde o objetivo de balancear a carga de trabalho da plataforma com a atividade de escalonar processos de maneira dinâmica e com preempção.

2.5.7. Taxonomia de Feitelson; Rudolph (1995)

O trabalho de Feitelson; Rudolph (1995) apresenta um resumo sobre escalonamento de processos, direcionado à computação paralela. Os autores classificam os algoritmos de escalonamento em duas dimensões (tempo e espaço) e em dois níveis (*jobs* e *threads*). O termo "nível" descrito aqui, não tem o mesmo sentido do "nível do escalonamento" apresentado por Baumgartner; Wah (1991).

Antes de ser apresentada a taxonomia em relação às dimensões (tempo e espaço), os algoritmos de escalonamento são classificados em: estáticos, em tempo de execução (*runtime system*), feitos pelo sistema operacional e os escalonamentos administrativos. Os algoritmos estáticos têm o mesmo sentido indicado por Casavant; Kuhl (1988).

Os algoritmos de escalonamento feitos em tempo de execução compreendem os algoritmos dinâmicos, embora estejam restritos aos feitos exclusivamente fora do sistema operacional e preocupam-se com as necessidades individuais em detrimento às coletivas.

O escalonamento feito pelo sistema operacional (o qual também pode ser dinâmico) é o escalonamento preocupado com as necessidades coletivas e desenvolvido especialmente para a plataforma utilizada.

O escalonamento administrativo agrupa os algoritmos que permitem decisões administrativas em detrimento às técnicas. Essa classe de escalonamento deve, por exemplo, permitir que determinadas aplicações utilizem exclusivamente os recursos computacionais, em função de algum objetivo maior da organização. Essa classe de "questões administrativas" novamente mistura os objetivos do algoritmo com a maneira que eles são implementados.

Considerando as duas dimensões: tempo e espaço, os algoritmos de escalonamento podem ser, respectivamente: *time sharing* ou *space sharing*[3] (ou ainda uma combinação de ambas). Na classe *time sharing* as aplicações paralelas compartilham o uso dos mesmos processadores com outras aplicações, enquanto que na classe *space sharing* um (sub)conjunto específico de processadores são separados e a eles são atribuídas aplicações paralelas, as quais usam-nos exclusivamente até o final da execução. A classe *space sharing* também é citada como *space partitioning* por alguns autores como Tripathi; Karnik (1995).

Os algoritmos *time sharing* são utilizados em casos onde há alta variação na carga computacional e onde há também pouco ou nenhum conhecimento sobre as aplicações que serão executadas. Ainda segundo os autores, somente com *time sharing* é possível garantir o tempo de resposta das aplicações interativas. No entanto, *time sharing* nem sempre é uma opção viável. Em algumas plataformas, garantir domínios de proteção às aplicações pode ser uma atividade altamente custosa ou mesmo impossível; além do fato de algumas aplicações paralelas manipularem volumes significativos de dados na memória principal, o que tornaria o "chaveamento de contexto" uma atividade onerosa.

Os algoritmos *space sharing* por sua vez, são utilizados com o objetivo de se obter melhores desempenhos com a utilização "exclusiva" dos recursos computacionais [Feitelson et al. (1997), Hwang; Xu (1998)]. S*pace sharing* também pode tornar-se inviável se a plataforma utilizada não dispõe dos recursos de software necessários para garantir essa

divisão; se a proteção do (sub)conjunto de processadores não pode ser garantida ou se as aplicações executadas necessitam da plataforma computacional como um todo. Outros autores como Harchol-Balter et al. (1997), citam que na computação paralela há a preferência pelo uso das políticas *space sharing* em função do alto custo para o chaveamento de contexto existente nas políticas *time sharing*.

Feitelson; Rudolph (1995) descrevem duas políticas de escalonamento pertencentes a cada tipo de compartilhamento (tempo e espaço). Para a classe *space sharing* são descritas as políticas chamadas: partição variável (*variable partitioning*) e partição dinâmica (*dynamic partitioning*). Para a classe *time sharing* são descritas as políticas chamadas: fila global (*global queue*) e *gang scheduling*.

Space sharing: embora os autores tenham enfatizado duas políticas de escalonamento (partição variável e partição dinâmica), as políticas *space sharing* podem ser divididas em quatro outras políticas: partição fixa, variável, adaptável e dinâmica (Tabela 3). Na partição fixa o número de processadores de cada partição é determinado previamente pelo administrador da plataforma. Somente um "reboot" permite a repartição. Na partição variável, também conhecida como *space sharing* pura (*pure space sharing*) ou partição estática (*static partitioning*), o tamanho de cada partição é definido previamente em função das necessidades de cada aplicação. Depois que cada aplicação termina a sua execução a partição é desfeita. Na partição adaptável o número de processadores de cada partição é definido em função da utilização da plataforma (carga computacional atual), durante a submissão da aplicação. Depois de feita, a partição não é alterada até o final da execução da aplicação paralela. Na partição dinâmica o tamanho de cada partição, além de também considerar a carga computacional na submissão, também permite a alteração durante a execução da aplicação.

Tabela 3 - Relação descrevendo a taxonomia das políticas *space sharing* proposta por Feitelson;Rudolph (1995).

	Parâmetros Considerados		
Políticas *Space sharing*	Solicitada pelo Usuário	Carga Computacional	Alterações na Partição
Partição Fixa	não	não	não
Partição Variável	sim	não	não
Partição Adaptável	sim	sim	não
Partição Dinâmica	sim	sim	sim

[3] Feitelson; Rudolph (1995) foram os únicos autores, encontrados na literatura de escalonamento de processos, que utilizam o termo *space* e *time slicing*ao invés de *space* e *time sharing*.

A divisão das políticas *space sharing* em quatro classes de políticas não foi encontrada em outros trabalhos (também voltados para a computação paralela). Islam et al. (1996), Squillante (1995), por exemplo, dividem as políticas *space sharing* apenas em três políticas: partição estática (ou variável), adaptável e dinâmica. Os significados dessas três políticas coincidem com o trabalho aqui apresentado.

A principal vantagem das políticas *space sharing* é permitir que o usuário tenha uma impressão mais "concreta" que está usando uma máquina dedicada, por possuir uma partição dos processadores exclusiva para ele. Dependendo da topologia utilizada, a aplicação não terá a interferência da rede de comunicação, porque o tráfego na sub-rede de comunicação pode ser exclusivo à partição feita.

A utilização dessas políticas *space sharing* depende diretamente da habilidade da aplicação paralela em adaptar-se a esses esquemas de divisão em grupos de processadores. Por exemplo, se uma aplicação não é capaz de executar com um número flexível de processadores, necessitando uma quantia predeterminada de elementos de processamento, de nada adianta uma partição dinâmica ou adaptável. A "habilidade" citada pode estar na própria aplicação, onde a aplicação determina quantos serão os seus processadores e eventuais mudanças nesses, sem nenhuma transparência. Essa "habilidade" também pode estar encapsulada em uma ferramenta de software que esconda da aplicação essas atividades e forneça-lhe maior flexibilidade e também (principalmente) transparência. Em outro trabalho, Feitelson; Rudolph (1996) dividem as aplicações paralelas em quatro classes, classificando-as através da necessidade por recursos e da habilidade de adaptação a diferentes partições (vide Tabela 4).

Tabela 4 - Classificação dos tipos de aplicações paralelas, baseada na especificação por recursos e também na habilidade de se adaptar a diferentes números de processadores [Feitelson; Rudolph (1996)].

Quem decide	Quando é decidido	
	na submissão	durante a execução
Usuário	Rígida (*rigid*)	Desenvolvimento Gradual (*evolving*)
Sistema	Modular (*moldable*)	Maleável (*malleable*)

As aplicações paralelas ditas "rígidas" são aquelas que necessitam um número exato de recursos (processadores, memória, e outros), os quais são especificados previamente pelo usuário. O fato de existir um número maior de recursos disponíveis no momento da execução

da aplicação, não melhora o desempenho da mesma. Porém, se os recursos solicitados não estiverem disponíveis, não será possível a execução da aplicação.

As aplicações classificadas como "desenvolvimento gradual" são aquelas que, além de especificarem as suas necessidades por recursos, podem alterá-las durante a execução, refletindo assim mudanças ocorridas na própria aplicação e também na plataforma computacional. Ambos, o levantamento das necessidades em termos de recursos e a própria mudança, são feitos através da aplicação e não do algoritmo de escalonamento. Essas aplicações contam apenas com o apoio de alguma ferramenta de software que permita a elas adaptarem-se as alterações ocorridas. As versões 3.3 e 3.4 do PVM são bons exemplos de "ferramentas" que possibilitam uma aplicação paralela inserir/retirar processadores da plataforma computacional, assim como alterar o número de processos durante a execução.

As aplicações modulares são aquelas em que a aplicação é mais flexível em relação às suas necessidades. Assim, é o algoritmo de escalonamento (ou ferramenta de software que o valha) que especifica os recursos desejáveis para executar a aplicação, durante a submissão. Depois de iniciada a execução, os recursos utilizados não são mais alterados. Baseando-se em informações da situação atual da plataforma computacional, o algoritmo de escalonamento determina o número de recursos que serão destinados à aplicação. Essa mesma atividade já foi definida como partição adaptável, a diferença está apenas no enfoque dado: para a aplicação ou para o processador. Normalmente as aplicações modulares são projetadas para executarem em um número mínimo de processadores. Acima dessa quantia, a aplicação terá o seu desempenho beneficiado até algum ponto de saturação, onde a inclusão de mais elementos de processamento não melhorará mais o desempenho da aplicação.

As aplicações maleáveis representam as mais flexíveis das aplicações descritas pelos autores, onde as necessidades por recursos podem ser alteradas em tempo de execução e pelo algoritmo de escalonamento, de maneira independente à aplicação. Novamente, se as aplicações maleáveis forem analisadas sob o ponto de vista dos processadores, essas atividades podem ser consideradas as mesmas descritas para a partição dinâmica.

As desvantagens das políticas *space sharing* dependem diretamente de qual política está sendo utilizada. Um dos maiores problemas é a fragmentação dos processadores disponíveis, principalmente com as políticas de partição fixa e variável (ou estática). A fragmentação ocorre quando o número de processadores disponíveis é insuficiente para executar quaisquer aplicações paralelas, deixando-os, portanto, ociosos. Outro problema é o tempo de espera

imposto às aplicações, na tentativa de obter acesso aos processadores solicitados, os quais não estão disponíveis no momento da submissão. Esses problemas tornam-se maiores quando são executadas aplicações com maior processamento e que necessitam de todos os processadores disponíveis. Essas aplicações "pesadas", por solicitarem todos os processadores, não executam enquanto houver outras aplicações em execução e, uma vez submetidas, não permitem que outras aplicações sejam escalonadas.

A partição adaptável é uma alternativa viável para as desvantagens apontadas, porque evita que processos "absorvam" muitos processadores quando a plataforma está com alta utilização no momento da submissão. A desvantagem da partição adaptável é não permitir alterações nas partições durante a execução, limitando, assim, seus benefícios quando há uma alta variação das cargas computacionais. A partição dinâmica, além de considerar a carga computacional no momento da submissão, também permite que as partições realizadas sejam alteradas durante a execução das aplicações, refletindo assim as mudanças da carga computacional.

Para os autores, as principais desvantagens da partição dinâmica são: (1) a sobrecarga para refazer as partições, como por exemplo a transferência de processadores de um domínio de proteção para outro, (2) a impossibilidade de execução de estilos de programação como o modelo SPMD (*Single Program, Multiple Data*), em função do número de processos ser fixo e freqüentemente igual ao número de processadores, (3) a coordenação extra necessária entre o algoritmo de escalonamento e a aplicação paralela, devido a alterações no número de processadores, as quais afetam o comportamento da aplicação sendo executada (a aplicação deve estar ciente e preparada para tais mudanças) e (4) a perda de portabilidade, face à maior coordenação entre algoritmo de escalonamento, aplicação paralela e plataforma.

Time sharing: a fila global é uma das mais simples políticas de escalonamento *time sharing*, onde uma única fila de execução recebe todas as submissões para a plataforma. Os processadores recebem o primeiro processo da fila, executam-no por um intervalo de tempo e depois devolvem-no à fila. A política fila global é normalmente utilizada em computadores paralelos com memória compartilhada e com pequeno número de processadores. Um dos principais problemas dessa política é a falta de capacidade escalar, onde o aumento do número de processos e processadores aumenta de maneira considerável a fila global. A maior vantagem da fila global é fornecer um compartilhamento de carga "automático", onde

nenhum processador fica parado enquanto há algum processo esperando para ser executado na plataforma.

Tripathi; Karnik (1995) citam que utilizando-se o escalonamento através de uma fila global, há a possibilidade da implementação de algumas variações tais como: (1) FCFS (*First Come First Served*), onde os primeiros processos a chegarem na fila serão os primeiros a serem executados, (2) SNPF (*Smallest Number of Process First*), a qual atribui prioridade maior para os processos de aplicações paralelas com menor número de processos não escalonados, (3) *RRprocess*, a qual implementa uma política de escalonamento *round-robin* para os processos na fila global, independente de qual aplicação paralela esses processos sejam e, finalizando, (4) *RRjob*, onde as aplicações paralelas (citadas como *jobs*) são escalonadas através da política *round-robin*. Para Feitelson;Rudolph (1995) há também a possibilidade de se incluir na política fila global o objetivo de balancear a carga do sistema, considerando assim a diferença entre as cargas computacionais dos processadores.

Dentre os possíveis problemas com a fila global, a falta de coordenação com o escalonamento dos diferentes processos que pertencem a uma aplicação paralela é um dos mais significativos. Considerando que em muitas aplicações paralelas os seus processos interagem e sincronizam com os demais, a falta de um escalonamento coordenado entre esses processos implica que, enquanto um processo está tentando sincronizar/comunicar com outro processo, este segundo pode não estar executando. Essa falta de coordenação impedirá que o(a) sincronismo/comunicação seja efetuado(a) momentaneamente, necessitando para tanto "chaveamentos de contextos" extras e, conseqüentemente, aumentando a sobrecarga.

A política de escalonamento *gang scheduling*, criada por J.K. Ousterhout em 1982 [Ousterhout (1982) *apud* Tanenbaum (1995)] é, segundo Feitelson; Rudolph (1995), a maneira mais usual de se implementar escalonamento *time sharing*. Com a *gang scheduling* todos os processos pertencentes a uma aplicação paralela são escalonados e suspensos simultaneamente, de uma maneira coordenada. A política de escalonamento *gang scheduling* tem, portanto, o objetivo de sincronizar o escalonamento dos vários processos pertencentes à aplicação paralela, para melhorar a sincronização entre os mesmos. Com isso a *gang scheduling* melhora o desempenho no paralelismo com menor granulosidade, onde há alta interação entre os processos (*I/O-Bound*). Com o escalonamento coordenado haverá uma melhor interação entre os processos, além destes serem escalonados permanentemente para os

mesmos processadores, retirando-se a necessidade de transferência do estado dos processos de uma memória para outra.

Analisando-se a freqüência com que os processos interagem, a *gang scheduling* pode apresentar algumas variações. *Coscheduling* é uma dessas variações, na qual são escalonados subconjuntos de processos (os que mais interagem), principalmente se é impossível escalonar todos os processos ao mesmo tempo. *Family scheduling* permite que sejam escalonados mais processos que processadores, utilizando para tanto um segundo nível de *time sharing*. Os termos *hard* e *soft scheduling* também são encontrados na literatura como sinônimos de *gang* e *coscheduling*, respectivamente [Feitelson; Rudolph (1996)]. Os termos *group*, *global* e o próprio *coscheduling* também podem ser encontrados como sinônimos de *gang scheduling* [Saphir et al. (1995), Schnor (1995), Schwiegelshohn (1996), Hwang; Xu (1998)].

Para Saphir et al. (1995) a política *gang scheduling* não é necessária quando os processos escalonados podem ser migrados entre os processadores (partição dinâmica), porque a migração possibilita maior flexibilidade ao escalonamento e conseqüentemente melhor desempenho.

As desvantagens da política *gang scheduling* ficam por conta: (1) da sobrecarga com o "chaveamento de contexto" coordenado, (2) da possibilidade de interferência com o *cache* de cada processador, devido às trocas constantes de processos e (3) da possível fragmentação dos processadores, problema considerado de menor importância devido à natureza *time sharing* dessa política.

Vantagens & Desvantagens: a taxonomia apresentada pelos autores segue uma linha voltada à computação paralela, fato que também ocorre em vários outros trabalhos disponíveis na literatura, como por exemplo: Islam et al. (1996), Krueger et al. (1994), Saphir et al. (1995), Schnor (1995), Schwiegelshohn (1996) e Squillante (1995). Dentro desse contexto (computação paralela) a taxonomia apresentada é coerente com os demais artigos encontrados na literatura, usando uma terminologia semelhante. A principal falha, não só desse trabalho mas também dos demais, é a ausência de um relacionamento explícito dessa taxonomia (voltada mais para os processadores), com as outras já discutidas neste trabalho (direcionadas mais para os processos). O leitor que não possui uma visão mais abrangente sobre a área de escalonamento, tem sérias dificuldades de se situar dentro desses contextos apresentados sob óticas diferentes.

2.5.8. Taxonomia de Milojicic et al. (1993)

O trabalho de Milojicic et al. (1993) classifica os algoritmos de escalonamento segundo as características expostas na Tabela 5. Os algoritmos de escalonamento são classificados em implementações feitas no: espaço do usuário em *kernels* monolíticos com interface UNIX, *kernel* dos sistemas operacionais monolíticos com interface UNIX, *kernel* dos sistemas operacionais baseados em mensagens, *kernel* dos sistemas operacionais com *µkernels* e, finalizando, espaço do usuário no topo de *µkernels*. Para comparar essas cinco classes foram utilizados os seguintes critérios: (1) funcionalidade, descrevendo se existe alguma restrição no escalonamento, como por exemplo, *systems calls* que não podem ser executadas; (2) transparência, representando a independência da aplicação em relação ao escalonamento, como por exemplo, a necessidade de recompilação ou "relinkedição" das aplicações para realizar uma migração de processos; (3) extensibilidade, descrevendo o quanto é possível modificar o escalonamento, sem a necessidade de alterar o restante do sistema operacional; (4)desempenho, representando a razão dos tempos de execução remoto e local e (5) capacidade escalar, representando o quanto o algoritmo de escalonamento é afetado com o aumento/diminuição da plataforma utilizada. A capacidade escalar depende também, segundo os autores, diretamente do sistema operacional. As cinco classes propostas são classificadas nesses critérios como: pobre, satisfatória, boa e excelente.

Tabela 5 - Taxonomia proposta por Milojicic et al. (1993).

	Funcionalidade	Transparência	Extensibilidade	Desempenho	Capacidade Escalar
Espaço usuário, UNIX	pobre	pobre	excelente	pobre	pobre
kernel, UNIX	boa	boa	pobre	excelente	satisfatória
Int. baseada em mensagens	boa	boa	pobre	boa	boa
µkernels	boa	boa	satisfatória	boa	boa
topo dos *µkernels*	excelente	boa	excelente	boa	boa

Vantagens & Desvantagens: a taxonomia proposta pelos autores, embora seja simples e bem específica, atende às suas necessidades. Mas para realizar uma comparação mais genérica, que saia do escopo do trabalho em questão, a taxonomia mostra-se muito limitada. Esse trabalho de Milojicic et al. (1993) é um exemplo de como os algoritmos de escalonamento podem ser vistos sob óticas distintas e, conseqüentemente, classificados de maneira diferente. Outro fator que pode-se concluir é a falta de uma preocupação maior em se situar as novas propostas dentro de taxonomias já existentes. Utilizando-se taxonomias mais abrangentes poder-se-ia comparar com justiça um maior número de outros trabalhos.

2.5.9. Taxonomia usada por Tanenbaum (1995)

A primeira classificação citada refere-se aos algoritmos migratórios e não migratórios, indicando respectivamente os algoritmos que realizam migração de processos (preemptivos) e aqueles que não a realizam (não preemptivos). O segundo grupo de taxonomias, as quais são apresentadas como "questões de projeto" de algoritmos de escalonamento, são: determinísticos x heurísticos, centralizados x distribuídos, ótimos x subótimos, locais x globais e iniciados pelo transmissor x iniciados pelo transmissor.

Os algoritmos determinísticos indicam o escalonamento que possui conhecimento total sobre o comportamento de todos os processos a serem escalonados e sobre a plataforma como um todo. Os algoritmos determinísticos são aqueles que podem teoricamente analisar todas as possíveis variações para o escalonamento e escolher a melhor. Os algoritmos de escalonamento heurísticos, por outro lado, são os que estão preparados para variações arbitrárias e significativas na plataforma e nos processos em tempo de execução. Pela definição dada para o escalonamento determinístico e para o heurístico, percebe-se que ambos representam respectivamente os algoritmos estático e dinâmico definidos em Casavant; Kuhl (1988).

Os algoritmos centralizados e os distribuídos foram definidos de maneira similar às definições apresentadas por Casavant; Kuhl (1988) para não distribuídos e distribuídos.

A questão referente aos algoritmos ótimos e subótimos estão relacionadas ao custo na obtenção de soluções ótimas e em como determinar uma solução subótima. Essa característica já foi abordada em taxonomias anteriores, como as taxonomias de Casavant; Kuhl (1988) e Baumgartner; Wah (1991).

O termo local para Tanembaum (1996) refere-se aos algoritmos que utilizam apenas as informações pertencentes ao processador que enviará/receberá processos (usa informações locais ao processador). O termo global refere-se aos algoritmos que obtêm informações de outros processadores para tomar decisões sobre a transferência de processos. Essa definição de local e global não é a mesma usada por Casavant;Kuhl (1988) e nem é a mesma usada por Lüling et al. (1993) e Xu; Lau (1997). Outra questão é a afirmação do autor de que a questão "local x global" é freqüentemente chamada de política de transferência (determina se um processador é transmissor ou receptor de processos). As definições de Tanenbaum (1995) não coincidem com as expostas por Shivaratri et al. (1992). Embora a política de transferência

realmente seja afetada por decisões locais e/ou globais, é a política de informação (onde, qual, quando e como buscar informações) que possui o maior peso para determinar se um algoritmo será "global ou local" nos termos apresentados no trabalho de Tanenbaum (1995). Dessa maneira, a afirmação em questão seria mais apropriada se colocasse que a questão "local x global" refere-se à política de informação.

Os algoritmos iniciados pelo transmissor e pelo receptor foram definidos de maneira semelhante às outras definições já apresentadas. No entanto, Tanenbaum (1995) afirma que a regra de iniciação (transmissor ou receptor) é de responsabilidade da política de localização (localizar um parceiro para o escalonamento). Mesmo que a política de localização também seja afetada pela maneira como o escalonamento é iniciado (irá localizar ou um receptor ou um transmissor), essa regra de iniciação refere-se à política de transferência. É esta a responsável por determinar se um processador deve transferir e/ou receber processos e, assim, iniciar o escalonamento em tempo de execução. A política de localização entra em operação somente para localizar um parceiro e tão somente depois que a decisão de se fazer o escalonamento já está tomada.

Há ainda a afirmação de que a política de localização não pode ser local. Essa não é a opinião de vários outros autores, dentre eles Casavant; Kuhl (1988) e Shivaratri et al. (1992), onde é citado que há a possibilidade de uma política de localização local, a qual pode escolher os parceiros aleatoriamente e, portanto, não depende de informações de outros processadores.

Vantagens & Desvantagens: a taxonomia usada pelo autor é didática e de fácil compreensão. A parte desse trabalho que trata sobre o escalonamento de processos pode ser considerada uma literatura introdutória (o objetivo dessa parte do trabalho não é ser um material profundo sobre o assunto). Entretanto, algumas afirmações contradizem vários outros autores, devendo o leitor consultar outras referências (as quais também estão relacionadas na bibliografia de Tanenbaum (1995)), para formar uma definição apropriada das mesmas.

2.6. Uma Convergência para as Taxonomias

As taxonomias apresentadas neste trabalho demonstram que elas são, em alguns casos, muito divergentes e podem ser divididas em três grupos diferentes. As taxonomias de Casavant; Kuhl (1988), Wang; Morris (1985), Baumgartner; Wah (1991), Milojicjc et al. (1993) e Tanenbaum (1995) classificam os algoritmos de escalonamento sob o ponto de vista dos processos, considerando como os processos têm acesso aos processadores. Assim sendo,

essas taxonomias podem ser chamadas de baseadas nos processos. Dessas taxonomias acima, as de Milojicjc et al. (1993) e Tanenbaum (1995) podem ser diferenciadas das demais por serem ou bem específicas ou então genéricas o suficiente para não apresentarem uma característica muito bem definida. Embora haja essa diferença, as duas ficam melhor agrupadas com as taxonomias baseadas nos processos, porque também preocupam-se com a maneira com que os processos terão acessos aos processadores.

As taxonomias de Lüling et al. (1993), Xu; Lau (1997) e Quinn (1994) apesar de também serem baseadas em processos, preocupam-se com a distância entre os processadores que farão parte do escalonamento, considerando a topologia utilizada pela plataforma computacional. Portanto, essas taxonomias podem ser chamadas de baseadas na topologia.

A taxonomia de Feitelson; Rudolph (1995) pode ser denominada de baseada nos processadores, porque divide os algoritmos de escalonamento pela maneira com que os processadores serão disponibilizados aos processos de uma aplicação paralela.

Considerando as diferentes linhas das taxonomias apresentadas, acredita-se que não seja uma tarefa trivial para um leitor (principalmente se iniciante), se orientar e ainda estabelecer com rapidez e segurança um relacionamento entre dois ou mais algoritmos de escalonamento, classificados através de duas ou mais taxonomias diferentes. Não é uma tarefa trivial analisar uma taxonomia, como por exemplo de Feitelson; Rudolph (1995), relacionando-a com as taxonomias baseadas em processos.

Para exemplificar essa situação, imagine por um instante um algoritmo de escalonamento A classificado segundo as taxonomias baseadas em processos e um algoritmo B classificado segundo as taxonomias baseadas em processadores. O algoritmo de escalonamento A, segundo a sua taxonomia, é considerado global, dinâmico, distribuído, não determinístico e iniciado pelo transmissor. O outro algoritmo de escalonamento, segundo a sua taxonomia, é classificado como *space sharing,* partição variável e quando a partição é determinada, usa também a política *gang scheduling*. Com base nas taxonomias apontadas para os dois algoritmos, essas duas propostas de escalonamento são iguais ou são diferentes? A maneira de responder essa questão é analisar um dos algoritmos de escalonamento sob o ponto de vista da outra taxonomia.

Analisando o algoritmo B, não sob o enfoque dos processadores, mas sim dos processos, tem-se que o algoritmo de escalonamento determina para onde os processos serão atribuídos antes da execução dos mesmos. Ao analisar a definição da partição variável (ou pura): "o

tamanho de cada partição é definido previamente em função das necessidades de cada aplicação (depois que cada aplicação termina a sua execução a partição é desfeita)", conclui-se que o algoritmo B é estático por determinar o escalonamento antes da execução da aplicação e também por não considerar a carga de trabalho existente na plataforma durante a execução. A política *gang scheduling*, que também está presente no algoritmo B, sincroniza a preempção entre os processos pertencentes à aplicação paralela. Mesmo com a política *gang scheduling*, o algoritmo B continua sendo estático, porque a *gang scheduling* não considera quando e onde os processos serão executados e também não se preocupa com a situação atual da carga de trabalho dos processadores. Apesar de ser preemptiva, a *gang scheduling*, preocupa-se tão somente em sincronizar as preempções locais, feitas em cada processador.

Para estabelecer uma convergência entre as diferentes taxonomias, as seções seguintes determinam uma relação entre as taxonomias baseadas nos processos com as outras duas linhas de taxonomias: baseadas na topologia e nos processadores.

2.6.1. Taxonomias baseadas nos processos x baseadas na topologia

Esses dois grupos de taxonomias são os mais fáceis de serem relacionados, porque as taxonomias baseadas na topologia podem ser consideradas como uma extensão das baseadas nos processos. Conforme citado na taxonomia de Xu; Lau (1997), os algoritmos classificados pelo domínio do escalonamento como locais ou globais são dinâmicos (segundo Casavant; Kuhl (1988)), porque determinam o escalonamento e consideram a carga computacional durante a execução. A diferença está apenas se o processador analisa somente a sua situação local e/ou no máximo dos seus processadores vizinhos; ou então analisa a situação da carga computacional de toda a plataforma.

O significado dos termos "local e global" nas taxonomias baseadas nos processos é diferente do significado usado para os mesmos termos nas taxonomias baseadas na topologia. As taxonomias de Casavant; Kuhl (1988) e de Xu; Lau (1997) exemplificam bem as diferenças.

Considerando as taxonomias baseadas em processos como referência, os algoritmos de escalonamento baseados nos vizinhos mais próximos (ou iterativos ou locais) [Xu; Lau (1997)] são equivalentes aos classificados como distribuídos [Casavant; Kuhl (1988)]. Isso acontece porque as decisões tomadas para o escalonamento são feitas por vários

processadores em tempo de execução, durante os vários "saltos" dos processos sobre os processadores.

Os algoritmos diretos (ou globais) [Xu;Lau (1997)] podem ser tanto distribuídos quanto não distribuídos. Isso depende da maneira com que eles são implementados.

Os algoritmos denominados como iterativos/determinísticos são cooperativos, porque tomam suas decisões considerando a situação de outros processos (mesmo que apenas seus vizinhos) e são também subótimos, porque a cada iteração um novo escalonamento é decidido somente entre os vizinhos (um escalonamento ótimo com essas características tornar-se-ia oneroso). Esses mesmos algoritmos podem ser iniciados pelo transmissor/receptor/simétrico, implementar ou não migração e ser ou não adaptáveis, todas essas características dependentes da maneira como seja implementado o algoritmo.

Os algoritmos denominados como iterativos/estocásticos não são cooperativos, porque determinam seus parceiros (considerando apenas os vizinhos) de maneira arbitrária (aleatória) e sem se preocupar em como está o processador que será parceiro no escalonamento. Esses mesmos algoritmos podem ser iniciados pelo transmissor/receptor/simétricos, implementarem ou não migração de processos, serem ou não adaptáveis, características essas dependentes da implementação do algoritmo.

A tese de doutorado feita por Muniz (1994) apresenta uma extensão à taxonomia baseada nos processos feita por Casavant; Kuhl (1988), unindo as características das taxonomias baseadas na topologia, como a de Lüling et al. (1993) e de Xu; Lau (1997). O trabalho de Muniz (1994) é um exemplo de trabalho na área de escalonamento que tem a preocupação de situar-se dentro de um contexto conhecido na literatura.

A Figura 9 demonstra graficamente a extensão feita por Muniz (1994), onde as partes sublinhadas e em negrito demonstram as principais alterações realizadas. A classificação em globalmente distribuído e localmente distribuído (ou semidistribuído) feita por Muniz (1994) pode ser extrapolada para a classificação global (ou direta) e em vizinhos mais próximos (ou iterativas) de Xu; Lau (1997), porque ambas as classificações preocupam-se ou com o uso de todos os processadores ou então de um subconjunto destes, respectivamente. A classificação em cooperativos e não cooperativos pode ser extrapolada para a classificação em determinísticos e estocásticos de Xu;Lau (1997). Os determinísticos cooperam entre si para atingirem um objetivo comum, enquanto que os estocásticos não. Finalizando, o modelo gradiente (estendido, no caso) está subordinado à classificação em cooperativos da mesma

forma como o modelo gradiente está subordinado à classificação determinística em Xu; Lau (1997).

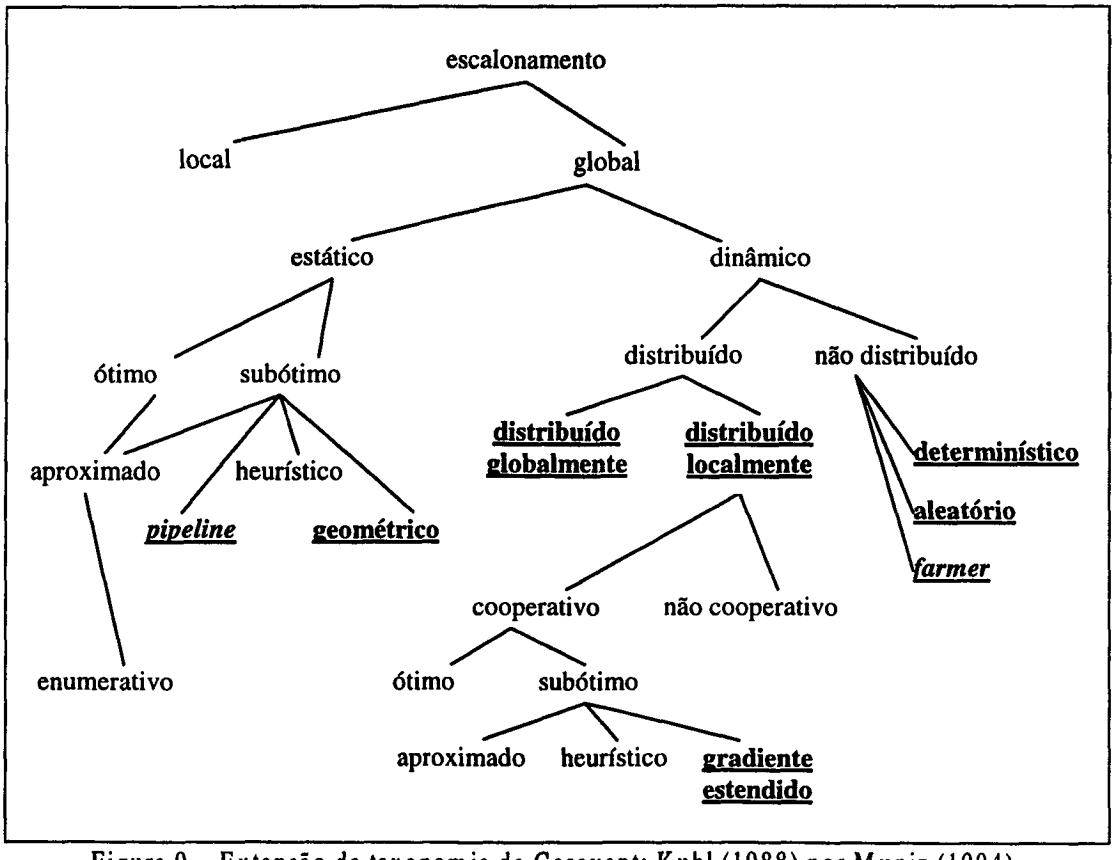

Figura 9 - Extensão da taxonomia de Casavant; Kuhl (1988) por Muniz (1994).

2.6.2. Taxonomias baseadas nos processos x baseadas nos processadores

Analisando as políticas *space sharing* com partições fixas ou variáveis e mudando o enfoque para os processos que serão escalonados nessas partições, percebe-se que o escalonamento é estático, porque os processadores são determinados antes da aplicação executar, quer seja através de informações fornecidas pelo próprio usuário/administrador, quer seja através de informações fornecidas por uma ferramenta de software que analise o código fonte e determine quais serão os processadores disponíveis à aplicação. Segundo o que está exposto na literatura, durante a execução da aplicação não é feita nenhuma consideração que pondere o destino dos processos. A principal diferença entre as duas políticas fixa e variável é que a primeira determina o número de processadores de maneira arbitrária e sem considerar (nem estaticamente) a aplicação a ser executada, enquanto que a segunda tem condições de pelo menos indicar antes da execução os seus requisitos de hardware.

Considerando as políticas *space sharing* com partições adaptáveis ou dinâmicas (sob o enfoque dos processos que serão escalonados), conclui-se que o escalonamento é dinâmico, em virtude dos processadores serem determinados durante a execução dos processos e também por considerarem a situação atual da carga trabalho da plataforma computacional. As adaptáveis, embora determinem o escalonamento durante a execução, o fazem apenas na submissão e não implementam a migração de processos. Já as dinâmicas, mesmo tendo determinado o subconjunto de processadores que irão servir à aplicação paralela durante a submissão da mesma, podem rever o escalonamento feito, alterar a partição e migrar os processos já escalonados.

Para as políticas *space sharing* com partições adaptáveis ou dinâmicas, questões como por exemplo: distribuído & não distribuído, cooperativo & não cooperativo, ótimo & subótimo e quem é o responsável por iniciar o escalonamento (transmissor/receptor/simétrico) são dependentes da implementação do algoritmo.

Considerando as políticas *space sharing* deve-se ponderar que há a possibilidade de dividir o escalonamento a ser feito em duas etapas. A primeira etapa seria a partição dos processadores através de uma das quatro políticas descritas (fixa, variável, adaptável e dinâmica) e a segunda etapa seria uma atribuição posterior dos processos aos processadores que pertencem à partição. Nesse caso está se colocando um segundo nível de escalonamento, situação bem diferente da descrita pelos autores em suas publicações. Na descrição da sua taxonomia, Feitelson; Rudolph (1995) não citam essa possibilidade, a não ser no caso de políticas *space sharing* com partição dinâmica, onde a migração de processos atua dinamicamente no escalonamento feito.

A política *time sharing* com fila global é uma política dinâmica [Casavant; Kuhl (1988)], em função da atribuição dos processos aos processadores ser feita em tempo de execução, podendo até considerar uma distribuição homogênea da carga computacional (balanceamento de carga). Se um processador torna-se ocioso, ele é o responsável (através do algoritmo de escalonamento) por retirar o processo que está na "cabeça" da fila de execução da plataforma, executá-lo por uma quantia de tempo e posteriormente retorná-lo à "cauda" da fila. Apesar desse escalonamento contar com um único ponto de entrada para os processos (a fila global), a decisão por executar processos pode ser feita de maneira distribuída, dando a cada processador receptor a responsabilidade por iniciar o escalonamento. Considerando a responsabilidade por iniciar o escalonamento, esse algoritmo é iniciado pelo receptor. As

outras questões como: cooperativo & não cooperativo, implementar ou não migração, ser ou não adaptável ficam por conta da implementação.

As políticas *time sharing* implementadas com *gang scheduling* são mais difíceis de serem relacionadas em outras taxonomias. A *gang scheduling* tem o objetivo de sincronizar o escalonamento dos vários processos pertencentes à aplicação paralela, para melhorar a sincronização entre os mesmos. Se uma política implementa *gang scheduling,* não está determinado se o escalonamento é feito em tempo de execução, quem é o responsável por iniciá-lo, se a decisão é distribuída ou não (caso seja dinâmica), entre outras questões. Assim, pode-se afirmar que a política *gang scheduling* indica uma classe de algoritmos de escalonamento globais, segundo a taxonomia de Casavant; Kuhl (1988), a qual se preocupa com a sincronização da preempção local em cada processador (aquela feita por todo sistema operacional multitarefa/multiusuário). Não obstante, essa preocupação não determina que um algoritmo deva ser dinâmico ou estático. O escalonamento pode, perfeitamente, ser determinado em tempo de compilação e durante a execução, a preempção feita em cada processador estar sincronizada pela *gang scheduling*.

2.7. Componentes de um Algoritmo de Escalonamento

Há várias maneiras de se construir um algoritmo de escalonamento, onde o mesmo pode ser dividido em partes, chamadas aqui de componentes. Cada componente de um algoritmo de escalonamento é responsável por uma porção da atividade maior que é "escalonar processos" e a combinação desses diferentes componentes é que determinam como o escalonamento será classificado.

Determinar com exatidão quais são os componentes de um algoritmo de escalonamento não é uma tarefa trivial. Há várias maneiras de se fazer essa divisão, algumas mais modulares (com um maior grau de divisão) e outras menos. Entretanto, a quantidade e a natureza dos componentes em um algoritmo de escalonamento não altera o objetivo final: escalonar processos usando a política de escalonamento que se deseja.

O escalonamento pode ser dividido de diversas maneiras, principalmente com dois, três ou quatro componentes. Henderson (1995), Mehra; Wah (1993) e Stellner; Trinitis (1997) são exemplos de trabalhos que dividem o escalonamento em dois componentes. Para Henderson (1995), o escalonamento é formado por um componente que seleciona os processos a serem escalonados e por outro que localiza os recursos de hardware (processadores, memória e

outros) que executarão os processos. Para Mehra; Wah (1993), o escalonamento é composto por um componente que indica a carga de trabalho em cada processador e por outro componente, chamado de política de decisão, o qual é responsável por duas tarefas: determinar sob quais condições o escalonamento será feito (no caso do trabalho, envolvendo migração de processos) e também por localizar processadores que irão receber os processos.

A proposta de Stellner; Trinitis (1997) divide o escalonamento em um "balanceador de carga" (*load balancer*) e em um componente de decisão. O "balanceador de carga" é responsável por obter a informação da carga de trabalho de todos os processadores e o componente de decisão determina quais processadores estão sobrecarregados ou ociosos e seleciona quais processos farão parte do escalonamento.

Shirazi et al. (1995a), Muniz (1994), Xu; Lau (1997) e Feitelson et al. (1997) são exemplos de trabalhos que dividem o escalonamento em três componentes. Para Shirazi et al. (1995a) o escalonamento é formado por uma política de informação, uma de transferência e outra de atribuição. A política de informação é responsável pela quantia de informação necessária para que as decisões sejam tomadas; a política de transferência determina sob que condições um processo deveria ser escalonado, por exemplo, a carga de trabalho atual do processador e o tamanho do processo em questão; e a política de atribuição localiza um processador apto a receber o processo a ser escalonado.

Para Muniz (1994) o escalonamento é formado por uma política de informação, outra de localização e outra de transferência. A política de informação é a responsável por monitorar a carga de trabalho do processador; a política de localização é a responsável por localizar um processador apto a participar do escalonamento; e a política de transferência é a responsável por mover os processos entre os processadores.

Para Xu; Lau (1997) o escalonamento é formado por uma regra de localização, uma de distribuição e outra de seleção. A regra de localização determina parceiros (processadores) para o escalonamento, os quais podem receber ou transmitir processos. A regra de distribuição determina como redistribuir a carga de trabalho entre os processadores que farão parte do escalonamento (responsável por coletar e analisar as informações para o escalonamento). A regra de seleção, por sua vez, determina quais são os processos aptos a fazer parte do escalonamento.

Feitelson et al. (1997) apresentam uma proposta de padronização para o desenvolvimento de algoritmos de escalonamento, feita pela NASA, IBM, entre outros. Essa tentativa de

padronização, chamada PSCHED (a sigla não é explicada), especifica três módulos: um gerente de tarefas, um gerente de recursos e um escalonador. O gerente de tarefas atua como um "elo de ligação" entre os processos pertencentes a uma aplicação paralela e os demais componentes do escalonamento, sendo o responsável por alguns serviços como gerar um processo em um processador (local ou remoto), manipular sinais entre processos de uma aplicação paralela e servir de interface para que os processos tenham as informações necessárias para solicitarem a alocação ou liberação de processadores. O gerente de recursos é o responsável por monitorar os recursos disponíveis na plataforma computacional e alocar ou liberar processadores para processos solicitantes. O escalonador é o componente que implementa a maneira como o escalonamento será efetuado. É o escalonador o responsável por implementar a política geral do escalonamento e decidir como as informações vindas do gerente de tarefas e do gerente de recursos serão utilizadas, para que o escalonamento seja efetuado. Como base no exposto por Feitelson et al. (1997), pode-se deduzir que os dois primeiros componentes (de tarefa e de recursos) possuem os mecanismos necessários à política de escalonamento, esta implementada no componente escalonador.

Shivaratri et al. (1992) citam que um algoritmo de escalonamento é "tipicamente" dividido em quatro componentes: política de transferência, política de seleção, política de localização e política de informação.

A política de transferência indica se um processador encontra-se apto a participar de uma transferência de processos, como transmissor ou como receptor. Quando a política de transferência é baseada em limites, um processador é dito transmissor se a carga de trabalho do mesmo excede um limite T_1 e, por outro lado, é dito receptor se a carga de trabalho é menor que um limite T_2. Os valores de T_1 e T_2 podem ser iguais, dependendo da implementação feita. Quando a política de transferência é relativa, a carga de um processador é comparada com a carga de outros processadores. Com uma política de transferência relativa um processador pode, por exemplo, ser considerado receptor se a sua carga é inferior a algum outro processador em, no mínimo, um valor preestabelecido. Outra opção para uma política relativa é considerar um processador receptor se a sua carga de trabalho está entre as menores da plataforma computacional.

A política de seleção, como o próprio nome indica, é a responsável por selecionar os processos que farão parte do escalonamento, uma vez que a política de transferência determinou que o processador é um transmissor. Caso a política de seleção falhe na sua busca,

o processador não deveria mais ser considerado como transmissor. A seleção dos processos mais novos, aqueles que estão chegando à plataforma e ainda não foram executados, é a maneira mais simples de se implementar a política de seleção. Isso implica na ausência de migração de processos. Para selecionar processos, a política em questão deveria considerar alguns fatores como: (1) a sobrecarga existente na transferência dos processos deve ser mínima, (2) o processo selecionado deve possuir um tempo restante de execução suficiente para compensar a sobrecarga gerada com a transferência e (3) a quantia de procedimentos dependentes da localização feitos pelos processos selecionados, tais como janelas, eventos de mouse, *clock*, deve ser mínima.

A política de localização é a responsável por encontrar um parceiro para a transferência dos processos, localizando um processador que tenha sido "rotulado" pela política de transferência ou como transmissor ou como receptor. A política de localização pode ser centralizada ou descentralizada. Um exemplo de política de localização descentralizada é aquela que localiza possíveis parceiros através de votação (*polling*). Pela votação, um processador "questiona" outros processadores na tentativa de encontrar algum que esteja apto e/ou aceite fazer parte do escalonamento (receber/transmitir processos). Em uma política de localização centralizada há um "coordenador" responsável por localizar um processador apto a tomar parte no escalonamento. O coordenador, através da política de informação, coleta informações sobre a carga de trabalho na plataforma computacional e a política de localização usa essa informação para localizar possíveis parceiros.

A política de informação tem a responsabilidade de decidir quando, onde e qual informação deve ser coletada sobre a situação da carga de trabalho de cada processador pertencente à plataforma computacional. Shivaratri et al. (1992) destacam três tipos de políticas de informação: sob demanda, periódica e por mudança de estado.

As políticas de informação sob demanda (distribuídas por natureza), obtêm a situação da carga de trabalho de outros processadores somente quando o processador, onde essa política está sendo executada, torna-se ou um receptor ou um transmissor. Essas políticas sob demanda podem ser iniciadas pelo transmissor, pelo receptor ou por ambos (no caso simetricamente iniciada), de acordo com a política de transferência adotada.

As políticas de informação periódicas, como o próprio nome indica, fazem a atualização das informações pertinentes à carga computacional de tempos em tempos e, dependendo da implementação, podem ser centralizadas ou descentralizadas. Apesar da sua maior

simplicidade, as políticas de informação periódicas, a princípio, não se adaptam à variação da carga na plataforma computacional. Como exemplo desse fato, considere uma plataforma com uma alta utilização de seus recursos, onde nenhum processador encontra-se apto a receber processos. Apesar dessa situação, uma política periódica continua atualizando as cargas computacionais, contribuindo apenas para aumentar a sobrecarga e piorar a situação na plataforma.

Nas políticas de informação por mudança de estado, os processadores enviam a situação das suas cargas de trabalho para outros processadores quando estas mudam em, no mínimo, um valor preestabelecido. As políticas de informação por mudança de estado são diferentes das sob demanda em função de apenas enviarem as suas cargas de trabalho, não requisitando, portanto, informações de outros processadores. Quando essas políticas por mudança de estado são centralizadas, as informações são enviadas para apenas um processador, denominado coordenador.

Os componentes apresentados por Shivaratri et al. (1992) serão utilizados neste trabalho em função da simplicidade e da modularidade que essa divisão proporciona ao algoritmo de escalonamento e também devido ao fato desses componentes serem utilizados e/ou referenciados por outros trabalhos como Harchol-Balter; Downey (1997), Shu; Wu (1996) e Tanenbaum (1995).

2.8. Índice de Carga

A qualidade de um escalonamento está relacionada diretamente com a quantidade/natureza das informações obtidas pela política de informação e disponíveis no momento em que as decisões precisam ser tomadas para a atribuição dos processos. Uma dessas informações é a quantidade de carga de trabalho existente na plataforma computacional, indicando a situação atual e/ou de um passado próximo, na tentativa de predizer como será o comportamento futuro das aplicações a serem escalonadas.

Para quantificar o conceito de carga de trabalho em um processador é utilizada uma métrica denominada na literatura por Índice de Carga, que é definida como sendo um valor não negativo, iniciando em 0 (zero) se o processador está ocioso e aumentando positivamente conforme a carga de trabalho aumenta [Ferrari; Zhou (1987)].

Tanenbaum (1995) cita que dimensionar a carga de trabalho não é tão simples quanto parece, porque não existe uma única maneira correta para determinar se um processador está

carregado, moderado ou ocioso. Ferrari; Zhou (1987) citam que devido ao fato da demanda por recursos variar de aplicação para aplicação, o uso do termo genérico "a carga de um processador" pode não ser representativo. Para exemplificar essa situação os autores citam o caso onde um processador pode estar "sobrecarregado" enquanto o acesso aos discos não está. Nessa situação, para processos *CPU-Bound* a carga do processador deve ser considerada alta, enquanto para processos *I/O-Bound*, que dependam predominantemente do acesso aos discos, a carga do processador deve ser considerada baixa.

Para determinar a qualidade de um índice de carga, deve-se portanto, estar ciente de quais aplicações serão executadas e também quais são os objetivos esperados para o escalonamento. Ferrari; Zhou (1987) citam que, de uma maneira geral, a escolha do índice de carga deve considerar: (1) a capacidade de representar com qualidade a carga atual de um host; (2) a capacidade de predizer a carga em um futuro próximo; (3) a estabilidade do índice, o qual deve ignorar ao máximo flutuações de carga (*noise*) e (4) a relação com a métrica que irá avaliar o desempenho do escalonamento feito, para facilitar análises de expectativas de desempenho a ser alcançado com o uso do escalonamento (vide próxima seção que aborda o desempenho de um algoritmo de escalonamento).

Outra questão relacionada à qualidade de um índice de carga é a sobrecarga gerada para a sua obtenção. Um índice de carga, além de ser eficiente na representação da carga de trabalho, deve agregar a menor sobrecarga possível ao escalonamento, tendo-se em mente sempre a melhor relação custo x benefício. A maneira como o índice de carga é formado, tem papel decisivo na sobrecarga gerada. Os índices de carga mais específicos, aqueles formados por apenas um valor obtido em cada processador, tendem a apresentar uma sobrecarga menor, além de serem mais eficientes em situações específicas. Esses devem ser utilizados corretamente em determinados tipos de aplicações e também de acordo com os objetivos esperados para o escalonamento. Exemplos de índices específicos são: tamanho da fila de execução ou tamanho da fila de acesso à memória. Os índices de carga genéricos são aqueles formados pela união de mais de um valor obtido nos processadores. Esses índices são mais indicados quando não se possui um conhecimento maior sobre as aplicações que serão executadas e nem sobre os objetivos esperados para o escalonamento. Além de apresentarem uma tendência à sobrecarga maior, os índices genéricos não devem apresentar a mesma qualidade de representação da carga de trabalho, quando comparados com os índices específicos utilizados corretamente [Mehra; Wah (1993), Ferrari; Zhou (1987)]. Um exemplo

de índice de carga genérico pode ser a união do tamanho da fila de execução com o tamanho da fila de acesso à memória.

Os índices de carga também podem ser divididos em outros grupos como: baseados no tamanho da fila de acesso ao recurso, baseados no percentual de utilização do recurso e os baseados no tempo de execução/resposta [Ferrari; Zhou (1987), Shirazi et al. (1995a)].

Exemplos de índices baseados no tamanho da fila de acesso ao recurso podem ser: fila de processos no processador (executando, suspensos, prontos) e fila de acesso à memória. Exemplos de índices baseados no percentual de utilização podem ser: percentual de utilização da CPU, percentual da CPU ociosa e percentual de ocupação da memória. Exemplos de índice de carga baseados no tempo de execução/resposta são: tempo de CPU ociosa, tempo restante de execução dos processos, tempo de execução dos processos ativos, tempo (custo) para comunicação e tempo normalizado ou *slowdown* (tempo carregado / tempo ocioso) de um processo ou grupo de processos. Os índices acima podem representar valores instantâneos (o valor exato da última obtenção) ou então representar uma média das últimas obtenções. A utilização da média ajuda a reduzir o risco do valor obtido estar representando apenas uma flutuação instantânea da carga de trabalho.

Se a plataforma utilizada é heterogênea e multiusuária, a importância do índice de carga é ainda maior. Além das alterações feitas pela própria aplicação, a presença de aplicações de outros usuários e as diferenças entre os processadores influenciam decisivamente o desempenho esperado para a aplicação. As cargas de trabalho externas à aplicação são chamadas por alguns autores como *background workload* ou *external load* [Mehra; Wah (1993), Krone et al. (1998), Russ et al. (1997)] e devem ser consideradas pelo índice utilizado. As diferentes potências computacionais existentes nas plataformas computacionais distribuídas e heterogêneas, agregam ao índice de carga a tarefa de normalizar as cargas computacionais em relação às potências em cada processador. Essa normalização não é uma tarefa trivial, porque dependendo da aplicação utilizada o impacto da heterogeneidade será maior ou menor. Considere, por exemplo, dois *hosts* onde a heterogeneidade encontra-se apenas na velocidade de processamento e considere novamente as duas aplicações CPU-Bound e I/O-Bound. O impacto dessa heterogeneidade na aplicação CPU-Bound deve ser maior por esta necessitar mais da velocidade do processador que a I/O-Bound.

Siegell; Steenkiste (1997) desenvolveram um trabalho interessante relacionado à área de índice de carga, onde são determinados quais os parâmetros que devem ser considerados para

realizar o escalonamento. Esse levantamento é feito em tempo de execução e também por ferramentas que analisam o código da aplicação paralela em tempo de compilação.

O índice de carga utilizado pelo escalonamento de processos é um item que por si só já constitui uma linha de pesquisa. Em função disso não serão discutidos outros fatores pertencentes aos índices de cargas, os quais podem ser encontrados em: Ferrari; Zhou (1987), Kunz (1991), Shivaratri et al.(1992), Mehra; Wah (1993), Milojicic et al. (1993), Kaplan; Nelson (1994), Muniz (1994), Shirazi et al. (1995b), Dantas (1996), Ferstl (1996), Schnor et al. (1996), Tanenbaum (1995), Russ et al. (1997), Krone et al. (1998), entre outros.

2.9. Desempenho Alcançado com o Algoritmo de Escalonamento

Um algoritmo de escalonamento, além de permitir a execução de processos pelos processadores (objetivo principal), também deve maximizar o desempenho esperado para a aplicação executada e/ou para a plataforma computacional como um todo. Esse desempenho que tenta-se maximizar não é único e não pode ser quantificado uniformemente para todas as possíveis situações. Para determinar se o algoritmo de escalonamento está desempenhando a contento as suas funções, deve-se saber quais são os objetivos propostos para o algoritmo e também estabelecer uma métrica capaz de avaliar o desempenho em função desses objetivos.

2.9.1. Objetivos e Métricas de Desempenho

A literatura da área demonstra que os algoritmos de escalonamento se propõem a uma grande variedade de objetivos, embora o objetivo de "balanceamento de carga" tenha um grande destaque e chegue até a ser confundido com a própria atividade de escalonamento. Mesmo com a política de buscar os processadores menos sobrecarregados, o objetivo central de um algoritmo de escalonamento não é, necessariamente, tornar as cargas computacionais homogêneas ou balanceadas (meta do balanceamento de carga).

Alguns exemplos de outros objetivos são: compartilhamento de carga, aumento da utilização do processador, redução do tempo de resposta, redução do tempo de execução, aumento do *throughput* da plataforma, questões administrativas, imparcialidade na execução dos processos, transparência no escalonamento, execução de uma aplicação dentro do prazo estabelecido (*real time*), acesso a recursos específicos de hardware ou software, tolerância a falhas, redução do tempo de espera pelos recursos ou *wait time*, redução da fragmentação do

número de processadores disponíveis, emular uma máquina dedicada, entre outros [Wang; Morris (1985), Kunz (1991), Shivaratri et al. (1992), Mullender (1993), Kaplan; Nelson (1994), Harchol-Balter et al. (1997), Feitelson;Rudolph (1996), Franklin; Govidan (1996), Martin (1996), Kwok; Ahmad (1996), Robinson et al. (1996), Tanenbaum (1995), Billard; Pasquale (1997), Xu; Lau (1997), Rapine et al. (1998)].

Alguns desses objetivos são conseqüência de outros, como por exemplo quando tenta-se reduzir o tempo de execução, normalmente reduz-se também o tempo de resposta (onde o tempo de execução é o tempo computado desde o início da aplicação até o término da mesma e o tempo de resposta é o tempo para responder um evento gerado pela aplicação, como um *click* de *mouse*). Outros objetivos podem ser contraditórios, onde a opção por um objetivo inviabiliza o alcance de outros. Um exemplo de objetivo mutuamente exclusivo é o aumento da utilização do processador, onde escalonam-se preferencialmente processos "maiores", aumentando-se conseqüentemente o tempo de resposta das aplicações.

Alguns autores como Feitelson; Rudolph (1996) classificam os objetivos como coletivos e individuais. Os objetivos coletivos são aqueles que dão ênfase às preferências administrativas, vendo as necessidades dos seus usuários como um todo. O aumento do *throughput* da plataforma é um exemplo de objetivo coletivo. Os objetivos individuais são aqueles que dão ênfase às necessidades de um usuário em especial, em detrimento às necessidades coletivas. A redução do tempo de execução de uma aplicação é um exemplo de objetivo individual.

As métricas de desempenho formalizam os objetivos propostos para o algoritmo de escalonamento e permitem avaliar se esse algoritmo está ou não atingindo os seus objetivos. Alguns objetivos, no entanto, são difíceis de se quantificar, por fazerem referência a questões mais subjetivas como as questões administrativas de uma organização [Ferstl (1996)]. Nem sempre a visão puramente técnica (e quase sempre imparcial) para o escalonamento de um processo é a melhor política a ser adotada. A visão administrativa pode e deve ser utilizada em situações onde o escalonamento de certos processos são, por exemplo, cruciais à vida de uma organização.

Independente de questões administrativas, as métricas de desempenho (ou índices de desempenho ou funções de custo) existem e são bastante discutidas na literatura. Alguns exemplos de métricas são: tempo de execução, *throughput*, tempo de resposta, percentual de utilização do processador, tempo normalizado ou *slowdown* (tempo carregado / tempo

dedicado), tempo de espera para execução, diferença entre as cargas de trabalho dos processadores, processador ocioso enquanto há processos esperando para execução ou *idle-while-waiting* [Ferrari; Zhou (1987), Mehra; Wah (1993), Krueger et al. (1994), Feitelson;Rudolph (1996), Loh et al. (1996)].

Feitelson et al. (1997) destacam também a necessidade de se flexibilizar as métricas que avaliam o desempenho em função de determinados fatores, como por exemplo o horário de utilização e a demanda pelos recursos. Em horários "não convencionais", como madrugadas e finais de semana, pode-se dar prioridade às aplicações que requeiram maior potência computacional e não apresentem alta interatividade com os usuários. Desse modo, a métrica para avaliar o desempenho do escalonamento pode ser o percentual de utilização do processador e não o tempo de resposta das aplicações.

2.9.2. Fatores que Influenciam o Desempenho

Mesmo fixando um dos possíveis objetivos discutidos anteriormente, um algoritmo de escalonamento pode apresentar desempenhos distintos ou até mesmo negativo. Afirmar que exista um único algoritmo de escalonamento capaz de produzir excelentes resultados em todas as possíveis combinações de, por exemplo, hardware, software e questões administrativas é uma tarefa extremamente difícil.

Essa variação no desempenho é conseqüência de três fatores principais: (1) a maneira como o próprio algoritmo de escalonamento é construído, (2) a plataforma computacional utilizada, considerando por exemplo, potências computacionais, topologia, heterogeneidade e multiusuários e (3) a classe (ou natureza) da aplicação executada, caracterizando a carga de trabalho que será escalonada.

O **algoritmo de escalonamento** é um fator decisivo no desempenho atingido com o escalonamento. Além dos benefícios advindos com um escalonamento melhor elaborado, deve-se considerar também os custos necessários para a obtenção desse escalonamento. Algumas vezes, um algoritmo simples ou específico possui um ganho de desempenho melhor por apresentar pouca sobrecarga, do que quando comparado a um algoritmo mais complexo ou então genérico e que possui uma sobrecarga elevada [Wang; Morris (1985), Casavant; Kuhl (1988), Ferrari; Zhou (1987), Franklin; Govidan (1996), Tanenbaum (1995)]. Um exemplo a respeito dos custos de uma política de escalonamento é o emprego de uma política com migração de processos. O tempo necessário para transferir um processo de um

processador para outro deve ser muito analisado, para se ter certeza que os benefícios alcançados com a migração irão sobrepor os custos envolvidos com a transferência de todo o estado do processo em execução [Downey; Harchol-Balter (1995)].

Além da sobrecarga de um algoritmo de escalonamento, há também a questão da estabilidade e da eficiência do mesmo. Para alguns autores como Shivaratri et al. (1992), Krueger; Shivaratri (1994), Tanenbaum (1995) e Corradi et al. (1997), um algoritmo de escalonamento é instável se ele piora a situação de carga de uma plataforma, realizando indefinidamente execuções inúteis que irão contribuir apenas para agravar ainda mais a demanda pelos recursos. Um exemplo de operação inútil é a procura de processadores ociosos em uma plataforma totalmente sobrecarregada. Um algoritmo eficiente, por sua vez, além de ser estável, também melhora o desempenho da plataforma, quando comparado à execução das aplicações sem o algoritmo de escalonamento.

Xu;Lau (1997) definem estabilidade como sendo os ganhos obtidos com o escalonamento (indicando quão bem o escalonamento foi realizado), e eficiência como sendo os custos (a sobrecarga) gerados a partir da execução do algoritmo de escalonamento. Casavant; Kuhl (1988) apresentam uma definição semelhante à apresentada por Xu; Lau (1997), utilizando o termo desempenho no lugar de estabilidade.

A **plataforma computacional** utilizada é decisiva no desempenho assim como na escolha do algoritmo de escalonamento. Características como organização da memória principal, topologia, velocidade de transmissão pela rede de comunicação e confiabilidade afetam principalmente as atividades que necessitam mais comunicação [Loh et al. (1996)].

Há várias plataformas propostas, com os mais variados esquemas de conexão entre processadores, porém, mesmo considerando uma plataforma mais genérica como uma NOW, a simples troca da rede de conexão por outra com maior capacidade de transmissão, altera o desempenho alcançado pelos algoritmos de escalonamento. O custo para a troca de informações ou o envio do estado de algum processo em execução será menor.

A heterogeneidade presente nas plataformas distribuídas atuais acrescenta um fator decisivo para o escalonamento dos processos. Além de se determinar a carga atual de um elemento de processamento, deve-se também normalizar essa carga em função das diferenças encontradas nesses elementos. O impacto causado pela heterogeneidade depende também da classe de aplicação que está sendo executada.

A heterogeneidade pode ser configuracional ou arquitetural. A primeira ocorre quando computadores de mesma arquitetura apresentam: diferentes velocidades de processamento, diferentes quantidades de memória disponível, entre outras. A heterogeneidade arquitetural, por sua vez, ocorre quando as diferenças estão na própria arquitetura dos elementos de processamento utilizados, como por exemplo processadores CISC, RISC e paralelos [Mehra; Wah (1993) e Schnor et al. (1996)].

A presença de vários usuários simultâneos em uma plataforma também altera o desempenho dos algoritmos de escalonamento. Quando as plataformas distribuídas são "monopolizadas" por apenas um usuário é possível antever com mais facilidade como será o comportamento. Isso acontece porque as cargas de trabalho estão relacionadas exclusivamente aos processos em execução dessa aplicação. Essas cargas são conhecidas como "internas" [Krone et al. (1998)].

As plataformas multiusuárias agregam ao escalonamento a preocupação de considerar as variações na carga computacional que são externas à aplicação. Krone et al. (1998) designa esse tipo de carga como "externa". Assim, um escalonamento correto, realizado no início da execução de um processo pode se tornar inadequado em tempo de execução. Uma solução para esse problema é o emprego da migração de processos [Martin (1996) e Ferstl (1996)].

O comportamento de um algoritmo de escalonamento também varia em função da carga computacional que a plataforma está executando. Com altas cargas de trabalho (pior caso), alguns algoritmos de escalonamento tornam a plataforma computacional instável, contribuindo para uma degradação ainda maior do desempenho. Em outras situações, com cargas de trabalho moderada a baixa, essa situação pode não se repetir e, ao contrário, apresentar ganhos reais de desempenho [Shivaratri et al. (1992)].

A **classe da aplicação** ou **classe de software** utilizada é outro fator decisivo para determinar o desempenho de um algoritmo de escalonamento. O mesmo algoritmo, quando utilizado por um determinado tipo de aplicação, apresenta um "ganho" de desempenho que não se repete necessariamente com outro tipo de aplicação. O impacto do escalonamento em diferentes classes de aplicação é facilmente percebido considerando-se, por exemplo, que aplicações interativas necessitam de um tempo de resposta bem rápido (tempo de resposta de um mouse, por exemplo), enquanto que, normalmente, aplicações *batch* necessitam de um *turnaround time* menor. A caracterização da carga de trabalho é, portanto, uma atividade difícil, mas necessária para uma distribuição dos processos entre os processadores com maior

qualidade [Ferstl (1996), Feitelson et al. (1997)]. O fato de diferentes usuários submeterem de maneira concorrente seus processos, com diferentes necessidades, dificulta essa situação. Essas aplicações compartilham os mesmos recursos de processamento e afetam o desempenho delas mesmas e de outras aplicações em execução.

Para determinar a plataforma computacional usada e a natureza da aplicação que será executada, o algoritmo de escalonamento necessita de informações adicionais, além daquelas obtidas durante a execução (no caso de algoritmos de escalonamento dinâmicos). A quantidade e qualidade das informações (considerando sempre os custos para obtê-las), afeta proporcionalmente a qualidade do escalonamento realizado [Feitelson et al. (1997)].

2.10. Escalonamento: a Teoria e a Prática

Conforme está sendo demonstrado neste trabalho, a área de escalonamento de processos é muito pesquisada. Facilmente são encontrados vários trabalhos descrevendo alguma pesquisa relacionada à área. Entretanto, o conhecimento gerado através dessas pesquisas não é empregado com a mesma facilidade nos algoritmos de escalonamento utilizados na prática [Milojicic et al. (1993), Squillante (1995), Feitelson; Rudolph (1996), Islam et al. (1996), Paindaveine; Milojicic (1996), Schnor et al. (1996), Feitelson et al. (1997), Gehring (1997), Gibbons (1997)]. Essa lacuna existente entre a teoria e a prática na área de escalonamento de processos é conseqüência de alguns fatores, os quais são discutidos a seguir.

2.10.1. Fatores que Afastam a Teoria da Prática

A **complexidade da área** de escalonamento de processos, onde a escolha pelo melhor escalonamento depende de inúmeras questões já discutidas. Não é fácil para um usuário ter que se preocupar com todas as questões pertinentes à área. A falta de transparência, não só para o escalonamento em si, mas também para a escolha do escalonamento, dificulta a utilização das melhores tecnologias.

As **divergências na nomenclatura** dificultam o entendimento dos diferentes tipos de escalonamento existentes. A literatura da área é vasta, mas imprecisa e não fornece uma visão mais abrangente. O estudo da área de escalonamento, com a intenção de escolher a política mais apropriada para uma situação, não é uma tarefa fácil, desencorajando a aplicação na prática do que está sendo descoberto nas pesquisas.

A **visão algorítmica** das propostas de escalonamento fixam no algoritmo a política de escalonamento, determinando assim a maneira como o escalonamento será realizado. Essa inclusão não permite flexibilizar as decisões tomadas, restringindo a atuação do mesmo. Considerando a dificuldade de se desenvolver um algoritmo apto a resolver da melhor maneira possível todos os casos, os usuários ficam limitados a escalonamentos que até podem apresentar ganhos de desempenho, porém, podem não ser a melhor solução para o problema. Questões administrativas e/ou novas situações são fatos que podem surgir no decorrer do tempo e alteram os objetivos do escalonamento. Embora os algoritmos adaptáveis minimizem essa questão, inserindo certa flexibilidade ao escalonamento, esta continua limitada ao algoritmo.

A **complexidade dos algoritmos** de escalonamento "melhor elaborados", como por exemplo, com migração de processos ou que flexibilizam as políticas de escalonamento, muitas vezes inviabiliza a utilização dos mesmos na prática. Muitos produtos comerciais optam pela simplicidade e pelo menor custo, implementando algoritmos que apresentam algum ganho de desempenho para casos genéricos, mas não se preocupam com a qualidade dessa solução.

As **métricas de desempenho** devem ser tão flexíveis quanto o algoritmo de escalonamento que as mesmas avaliam. Métricas como a redução do tempo de execução, embora sejam muito utilizadas, não são as únicas. As métricas devem ser determinadas em conjunto com os objetivos do algoritmo de escalonamento e toda vez que os mesmos variarem, as mesmas devem ser reavaliadas.

2.10.2. Fatores que Convergem a Teoria e a Prática

Alguns autores destacam as características desejáveis de um algoritmo de escalonamento. Essas características, de certa forma, vêm ao encontro da necessidade de convergência entre a teoria e a prática na área de escalonamento.

Feitelson; Rudolph (1996) citam que a convergência é dependente dos seguintes pontos: (1) os trabalhos devem ser explícitos sobre as suas características, usando para tanto uma terminologia consistente e assim diminuindo a dificuldade em se relacionar trabalhos já desenvolvidos; (2) determinar e minimizar as deficiências de cada proposta, antes de construir novas; (3) combinar diferentes propostas, flexibilizando os algoritmos existentes; (4) usar

métricas (funções de custo) que reflitam as características e objetivos da organização, permitindo um monitoramento correto sobre os ganhos obtidos com o escalonamento.

Além dos pontos destacados acima, a lacuna em questão pode ser diminuída fornecendo aos usuários transparência completa em relação à atividade de escalonamento. Por transparência completa entende-se não só a transparência em se realizar um escalonamento, mas também em se escolher o melhor escalonamento a ser utilizado. Através do auxílio de uma ferramenta de software é possível, transparentemente, automatizar o processo de escolha do melhor algoritmo de escalonamento para uma plataforma computacional e também para situações específicas com a mesma plataforma. Automatizando-se a escolha correta do algoritmo de escalonamento retira-se a visão algorítmica da área e não se sobrecarrega um único algoritmo. Dessa maneira, vários algoritmos específicos podem ser utilizados no lugar de um genérico que apresenta maiores custos de execução.

3.Conclusões

Estudar a área de escalonamento de processos não é uma tarefa trivial. Não encontrou-se na literatura um trabalho que fornecesse uma visão abrangente da área, comparando as diferentes linhas de pesquisa descritas aqui, através das suas similaridades e diferenças. Este trabalho, além de fornecer uma revisão bibliográfica, teve por objetivo amenizar esse problema da literatura, fornecendo uma leitura abrangente e de mais alto nível, não descendo aos pormenores das implementações.

Além de fornecer um contexto mais abrangente para a área, este trabalho também teve por objetivo destacar a importância de se organizar de maneira modular e coerente o código de um algoritmo de escalonamento, através dos seus componentes. Não foram descritos os inúmeros algoritmos de escalonamento desenvolvidos e disponíveis na literatura. Essas implementações podem ser facilmente encontradas em vários trabalhos espalhados na literatura. O Apêndice 1 relaciona as referências bibliográficas encontradas de alguns desses trabalhos, os quais descrevem novas propostas de algoritmos de escalonamento.

O desempenho de um algoritmo de escalonamento é altamente influenciável por inúmeros fatores como: objetivos da plataforma usada (computação paralela ou sistema distribuído), a própria plataforma (topologia, heterogeneidade, multiusuários, etc), as classes de aplicação executadas, os horários de utilização, entre outros. Considerando-se todas essas possíveis variações, torna-se complexo afirmar que exista um algoritmo de escalonamento com desempenho favorável e constante para todos os casos.

A pouca utilização das pesquisas desenvolvidas nas propostas de escalonamento usadas comercialmente pode ser considerada como uma conseqüência de quatro fatores: (1) os problemas existentes na literatura; (2) todas as possíveis variações dos algoritmos de escalonamento; (3) a visão algorítmica das propostas de escalonamento; e (4) a falta de transparência para determinar qual política de escalonamento usar.

O escalonamento é tratado segundo a política e o mecanismo inserido no algoritmo de escalonamento, fato que limita uma maior flexibilidade no escalonamento dos processos,

mesmo que os algoritmos sejam adaptáveis. Para fornecer flexibilidade aos algoritmos de escalonamento, continuar usando algoritmos específicos e com menores custos de processamento e ainda permitir transparência total para os usuários de aplicações concorrentes e/ou distribuídas, deve-se utilizar uma ferramenta de software que encapsule todas as questões discutidas aqui e, assim, viabilizar a redução da lacuna existente entre a teoria e a prática na área.

Com o intuito de minimizar os problemas da área destacados aqui, está sendo definido e implementado o AMIGO (*DynAMical FlexIble SchedulinG EnvirOnment*), uma ferramenta de software aberta, responsável pela gerência do escalonamento de processos, de maneira flexível e transparente ao usuário [Souza et al. (1999a, 1999b, 1999c)].

Através do AMIGO diversos algoritmos de escalonamento podem ser reunidos sendo que cada um é específico a um determinado conjunto de fatores. Isso permite ao usuário utilizar os algoritmos de escalonamento mais apropriados às suas necessidades, quer sejam técnicas ou administrativas.

Referências Bibliográficas

AGUILAR, J.; JIMENEZ, T. A Processors Management System for PVM. *Euro-Par' 97 Parallel Processing - Third International Euro-Par Conference, Lecture Notes in Computer Science v.1300.* Passau, Germany, agosto, 1997.

ALANYALI, M.; HAJEK, B. On Simple Algorithms for Dynamic Load Balancing. In: *INFOCOM'95*, Boston, Massachusetts, abril, 1995.

AL-SAQABI, K.; PROUTY, R.M.; MCNAMEE, D.; OTTO, S.W.; WALPOLE, J. Dynamic Load Distribution in MIST. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDTA 97)*, 1997.

ANDERSON, T.E.; CULLER, D.E.; PATTERSON, D.A. A Case for Now (Networks of Workstations). *IEEE Micro*, p.54-64, fevereiro, 1995.

ANTONIS, K.; GAROFALAKIS, J.; SPIRAKIS, P. A competitive symetrical transfer policy for load sharing. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.352-355, setembro, 1998.

ARABE, J.; BEGUELIN, A.; LOWEKAMP, B.; SELIGMAN, E.; STARKEY, M.; STEPHAN, P. Dome: Parallel programming in a heterogeneous multi-user environment. *Technical Report CMU-CS-95-137*, School of Computer Science, Carnegie Mellon University, abril, 1995.

ARTSY, Y.; FINKEL, R. Designing a Process Migration Facility: The Charlotte Experience. *Computer*, v.22 n 9, p.47-56, setembro, 1989.

BARAK, A.; BRAVERMAN, A.; GILDERMAN, I.; LA'ADAN, O. Performance of PVM with the MOSIX Preemptive Process Migration. In: *Proceedings of the 7th Israeli Conf. on Computer Systems and Software Engineering*, Herzliya, p.38-45, junho, 1996.

BAUMGARTNER, K.M.; WAH, B. W. Computer Scheduling Algorithms: Past, Present and Future. *Information Sciences, Elsevier Science Pub. Co., Inc.*, New York, NY, v.57 & 58, p.319-345, setembro/dezembro, 1991.

BECKER, W. Dynamic balancing complex workload in workstation networks - challenge,concepts and experience. In: *High-Performance Computing and Networking, Lecture Notes in Computer Science v.919*, Milan, Italy, p.407-412, maio, 1995.

BEGUELIN, A.; SELIGMAN, E.; STEPHAN, P. Application Level Fault Tolerance in Heterogeneous Networks of Workstations. *Technical Report CMU-CS-96-157*, School of Computer Science, Carnegie Mellon University, agosto, 1996.

BEMMERL, T. The TOPSYS Architecture. *In: Proceedings of CONPAR'90 VAp.IV, Lecture Notes in Computer Science v.457*, 1990.

BERMAN, P.; CHARIKAR, M.; KARPINSKI, M. On-line load balancing for related machines. In: *Algorithms and data structures, Lecture Notes in Computer Science v.1272*, p.116-125, 1997.

BERMAN, F.; STRAMM, B. Mapping Function-Parallel Programs with the Prep.-P Automatic Mapping Preprocessed. *Technical Report Number CS 94-397*, p.1-28, dezembro, 1994.

BESTAVROS, A. Load Profiling in distributed real-time systems. *Information Sciences*, v.101, Iss 1-2, p.1-27, 1997.

BILLARD, E.A.; PASQUALE, J.C. Load Balancing to adjust for proximity in some network topologies. *Parallel Computing*, 22, p.2007-2023, 1997.

BLUMOFE, R.D.; JOERG, C.F.; KUSZMAUL, B.C.; LEISERSON, C.E.; RANDALL, K.H.; ZHOU, Y.L. Cilk: na efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, v.37, Iss 1, p.55-69, 1996.

BOERES, C.; REBELLO, V.E.F.; SKILLICORN, D.B. Static scheduling using task replication for LogP and BSP models. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.337-346, setembro, 1998.

BOUKERCHE, A.; DAS, S.K. Experimental studies in load balancing. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.318-321, setembro, 1998.

BREST, J.; ZUMER, V.; OJSTERSEK, M. Dynamic Scheduling on a Network Heterogeneous Computer System (Extend Abstract). In: *4th International ACPC Conference, Lecture Notes in Computer Science v.1557*, Salzburg, Austria, p.584-585, fevereiro, 1999.

BUBAK, M.; MOSCINSKI, J.; POGODA, M.; SLOTA, R. Load balancing for lattice gas and molecular dynamics simulations on networked workstations. In: *High-Performance Computing and Networking*, Lecture Notes in Computer Science v.919, Milan, Italy, p.329-334, maio, 1995.

CANDEA, G.M.; JONES, M.B. Vassal: Loadable Scheduler Support for Multi-Policy Scheduling. *Technical Report MSR-TR-98-30*. Microsoft Corporation, Advanced Technology Division. Redmond, WA, USA, agosto, 1998.

CASAS(a), J.; CLARK, D.; KONURU, R.; OTTO, S.; PROUTY, R.; WALPOLE, J. MPVM: A Migration Transparent Version of PVM. *Technical Report CSE-95-002*, fevereiro, 1995.

CASAS(b), J.; CLARK, D.; GALBIATI, P.; KONURU, R. ; OTTO, S.; PROUTY, R.; WALPOLE, J. MIST: PVM with Transparent Migration and Checkpointing. In: *3rd Annual PVM Users' Group Meeting*, Pittsburgh, PA, maio, 1995.

CASAVANT; T.L.; KUHL, J.G. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, p.141-154, fevereiro, 1988.

CHANDRA, R.; GUPTA, A.; HENNESSY, J.L. Data locality and load balancing in cool. *Sigplan notices*, v.28, Iss 7, p.249-259, 1993.

CHEN, H-L.; KING, C-T. Eager Scheduling with Lazy Retry for Dynamic Task Scheduling. In: *Second International Euro-Par Conference - Euro-Par'96, Lecture Notes in Computer Science v.1124*, Lyon, France, p.615-620, agosto, 1996.

CHERKASOVA, L. Scheduling strategy to improve response time for web applications. In: *Proceedings of High-Performance Computing and Networking, Lecture Notes in Computer Science v.1401*, Amsterdan, The Netherlands, p.305-314, abril, 1998.

CHICLAYO, P.W.C.; HENRIQUES, M.A.A. Alocação de Recursos Computacionais em Modelo Econômico para Sistemas de Processamento Massivamente Paralelo Virtuais. In: *XXVI Seminário Integrado de Software e Hardware-SEMISH99, Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação-SBC99*, Rio de Janeiro, p.211-220, julho, 1999.

CHIEN, Y.P.; ECER, A.; AKAY, H.U.; CARPENTER, F.; BLECH, R.A. Dynamic load balancing on a network of workstations for solving computational fluid-dynamics problems. *Computer Methods in Applied Mechanics and Engineering*, v.119, Iss 1-2, pp17-33, 1994.

CHRISTEN, P. Dynamic Load Balancing within a Parallel Iterative Linear System Solver. In: *Proceedings of High-Performance Computing and Networking, Lecture Notes in Computer Science v.1401*, Amsterdan, The Netherlands, p.857-859, abril, 1998.

CLARK, D.L.; CASAS, J.; OTTO, S.W.; PROUTY, R.M.; WALPOLE, J. Scheduling of Parallel Jobs on Dynamic, Heterogenous Networks. janeiro, 1995. *Disponível em: http://www.cse.ogi.edu/DISC/projects/cpe/*

COLAJANNI, M.; YU, P.S.; DIAS, D.M. Analysis of task assignment policies in scalable distributed web-server systems. *IEEE Transactions on Parallel and Distributed Systems*, v.9, Iss 6, p.585-600, 1998.

CORRADI, A.; LEONARDI, L.; ZAMBONELLI, F. Performance Comparison of Load Balancing Policies based on a Diffusion Scheme. In: *Euro-Par' 97 Parallel Processing -*

Third International Euro-Par Conference, Lecture Notes in Computer Science v.1300, Passau, Germany, agosto, 1997.

CORRADI, A.; LEONARDI, L.; ZAMBONELLI, F. On the effectiveness of different diffuse load balancing policies in dynamic applications. In: *Proceedings of High-Performance Computing and Networking, Lecture Notes in Computer Science v.1401*, Amsterdan, The Netherlands, p.274-283, abril, 1998.

CZAJKOWSKI, K.; FOSTER, I.; KARONIS, N.; KESSELMAN, C.; MARTIN, S.; SMITH, W.; TUECKE, S. A Resource Management Architecture for Metacomputing Systems. In: *IPPS/SPDP' 98 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1459*, Orlando, Florida, USA, p.62-82, março, 1998.

DANTAS, M.A.R. Efficient Scheduling of Parallel Applications on Workstation Clusters. *Tese de Doutorado apresentada à Faculdade de Engenharia da Universidade de Southampton*, Reino Unido, setembro, 1996.

DANTAS, M.A.R.; ZALUSKA, E.J. Efficient Scheduling of MPI applications on Networks of Workstations. *Future Generation Computer Systems - FGCS*, v.13, p.489-499, 1998

DAVOLI, R.; BABAOGLU, Ö. Parallel Computing in Networks of Workstations with Paralex. *IEEE Transactions on Parallel and Distributed Systems*, v.7, n.4, abril, 1996.

DECKER, T. Virtual data space - A universal load balancing scheme. In: *Solving irregularly structured problems in parallel, Lecture Notes in Computer Science v.1253*, p.159-166, 1997.

DEVARAKONDA, M.V.; IYER, R.K. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Transactions on Software Engineering*, v.15, n.12, p.1579-1586, dezembro, 1989.

DIKENELLI, O.; OZKASAP, O.; OZKARAHAN, E. Scheduling Parallel Programs Involving Parallel Database Interactions. *In: 4th International Conference PaCT'97 Yaroslaul, Parallel Computing Tecnologies, Lecture Notes in Computer Science v.1277*, Rússia, setembro, 1997.

DONGARRA, J.J.; GEIST, A.; KOHL, J.A.; PAPADOPOULOS, P.M.; SUNDERAM, V. HARNESS: Heterogeneous Adaptable Reconfigurable Netorked SystemS. Relatório técnico *disponível em http://www.epm.ornl.gov/harness/*, março, 1998.

DOUGLIS, F. Experience with Process Migration in Sprite. In: *Distributed and Multiprocessor Systems Workshop Proceedings*, Fort Lauderdale, FL, p.59-72, outubro, 1989.

DOUGLIS, F.; OUSTERHOUT, J. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software - Practice and Experience*, v.21, n.8, p.757-785, agosto, 1991.

DOWNEY, A.B. Lachesis: A Job Scheduler for the Cray T3E. In: *IPPS/SPDP' 98 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1459*, Orlando, Florida, USA, p.47-61, março, 1998.

DOWNEY, A.; HARCHOL-BALTER, M. A note on 'The Limited Performance Benefits of Migrating Active Processes for Load Sharing. *Technical Report UCB/CSD-95-888*, University of California at Berkeley, 1995.

DURAND, D.; MONTAUT, T.; KERVELLA, L.; JALBY, W. Impact of memory contention on dynamic scheduling on NUMA multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, v.7, Iss 11, p.1201-1214, 1996.

EPEMA, D.H.J.; LIVNY, M.; DANTZIG, R.; EVERS, X.; PRUYNE J. A worldwide flock of Condors:Load sharing among workstation clusters. *Future Generation Computer Systems - FGCS*, 12, p.53-65, 1996.

ESKICIOGLU, M.R. Design Issues of Process Migration Facilities in Distributed Systems. *IEEE Technical Committee on Operating System Newsletter*, p.3-13, 1989.

FAGG, G.E.; LONDON, K.S.; DONGARRA, J.J. Taskers and General Resource Managers: PVM Supporting DCE Process Management. In: *Third European PVM Conference - EuroPVM'96; Lecture Notes in Computer Science v.1156*, Munich, Alemanha, outubro, 1996.

FEITELSON, D.G.; RUDOLPH, L.; SCHWIEGELSHOHN, U.; SEVCIK, K.C.; WONG, P. Theory and Practice in Parallel Job Scheduling. In: *IPPS' 97 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1291*, Santa Barbara, Geneva, Switzerland, abril, 1997.

FEITELSON, D.G.; RUDOLPH, L. Parallel Job Scheduling: Issues and Approaches. In: *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.949*, Santa Barbara, CA, USA, abril, 1995.

FEITELSON, D.G.; RUDOLPH, L. Toward Convergence in Job Schedulers for Parallel Supercomputers. In: *IPPS' 96 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1162*, Santa Barbara, Honolulu, Hawaii, abril, 1996.

FERRARI, D.; ZHOU, S. An Empirical Investigation of Load Indices for Load Balancing Applications. In: *Proceedings of Performance'87, the 12th Int'l Symposium on Computer Performance Modeling, Measurement, and Evaluation*, p.515-528, 1987.

FERREIRA, A.B.H. *Novo Dicionário Aurélio da Língua Portuguesa*. 2.ed., Rio de Janeiro, Nova Fronteira S.A., 1986.

FERSTL, F. Job-and resource-management systems in heterogeneous clusters. *Future Generation Computer Systems - FGCS*, 12, p.39-51, 1996.

FLYNN, M.J. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, v.C-21, p.948-960, 1972.

FONLUPT, C.; MARQUET, P.; DEKEYSER, J.L. Data -parallel load balancing strategies. *Parallel Computing*, v.24, p.1665-1684, 1998.

FRANKLIN, M.A.; GOVINDAN, V. A general matrix iterative model for dinamic load balancing. *Parallel Computing*, v.22, p.969-989, 1996.

GEHRING, J. Dynamic Program Description as a Bassis for Runtime Optimization. Euro-Par' 97 Parallel Processing – In: *Third International Euro-Par Conference, Lecture Notes in Computer Science v.1300*, Passau, Germany, agosto, 1997.

GEIST, A.; BEGUELIN, A.; DONGARRA J.J.; JIANG, W; MANCHEK, R.; SUNDERAM, R. *PVM 3 Users Guide and Reference Manual.* Oak National Laboratory, setembro, 1994.

GENCO, A.; LORE, G. The egoistic approach to parallel process migration into heterogeneous workstation network. *Journal of Systems Architecture*, v.42, Iss 4, p.267-278, 1996.

GIBBONS, R. A Historical Application Profiler for Use by Parallel Schedulers. In: *IPPS' 97 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1291*, Santa Barbara, Geneva, Switzerland, abril, 1997.

GIL, J.; MATIAS, Y. An effective load balancing policy for geometric-decaying algorithms. *Journal of Parallel and Distributed Computing*, v.36, Iss 2, p.185-188, 1996.

GLAZEK, W. How To Share a Divisible Load in a Hypercube (Extend Abstract). *In: 4th International ACPC Conference, Lecture Notes in Computer Science v.1557*, Salzburg, Austria, p.588-589, fevereiro, 1999.

GRAHAM, P.C.J.; BARKER, K.E. Service Migration to Improve Load Balancing in Distributed Systems. *Technical Report TR94-04*, Dept. of Computer Science, University of Manitoba , abril 1994.

HA, S.H.; LEE, E.A. Compile-time scheduling and assignment of data-flow program graphs with data-dependent iteration. *IEEE Transactions on Computers*, v.40, Iss 11, p.1225-1238, 1991.

HALANG, W.A.; JUNG, S.K.; KRÄMER, B. Enforcing Management Policies in Distributed Systems. In: *Proceedings of the Fith IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Cheju Island, Korea, p.474-480, agosto, 1995.

HARCHOL-BALTER, M.; CROVELLA, M.; MURTA, C. To Queue or Not to Queue: When Queueing is Better Than Timesharing in a Distributed System. *Technical Report BUCS-TR-1997-017*, outubro, 1997.

HARCHOL-BALTER, M.; DOWNEY, A. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems* v.15, n.3, agosto, 1997.

HENDERSON, R.L. Job Scheduling Under the Portable Batch System. In: *IPPS' 95 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.949*, Santa Barbara, CA, USA, abril, 1995.

HILL, J.M.D.; DONALDSON, S. R.; LANFEAR, T. Process migration and fault tolerance of BSPlib programs running on networks of workstations. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.80-91, setembro, 1998.

HU, Y.F.; BLAKE, R.J.; EMERSON, D.R. An Optimal Migration algorithm for Dynamic load balancing. *Concurrency: Practice and Experience*, v.10(6), p.467-483, 1998.

HUANG, G.; ONGSAKUL, W. An efficient load-balancing processor scheduling algorithm for parallelization of Gauss-Seidel Type Algorithms. *Journal of Parallel and Distributed Computing*, v.22, Iss 2, p.350-358, 1994.

HUI, C.C.; CHANSON, S.T. Flexible and extensible load balancing. *Software-practice & experience*, v.27, Iss 11, p.1283-1306, 1997.

HWANG, K.; XU, Z. *Scalable Parallel Computing*: Technology, Architecture, Programming. New York, McGraw-Hill Inc., janeiro, 1998.

ISLAM, N.; PRODROMIDIS, A.; SQUILLANTE, M.S. Dynamic Partitioning in Diferent Distributed-Memory Environments. In: *IPPS' 96 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1162*, Santa Barbara, Honolulu, Hawaii, abril, 1996.

JACKSON, D.J.; HUMPHRES, C.W. A simple yet effective load balancing extension to the PVM software system. *Parallel Computing*, v.22, p.1647-1660, 1997.

JONES, M.B.; ROI U, D.; ROI U, M. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. *Technical Report MSR-TR-97-19*. Microsoft Corporation, Advanced Technology Division. Redmond, WA, USA, outubro, 1997.

JOOSEN, W.; BERBERS, Y.; VERBAETEN, P. Dynamic Load Balancing in Transputer Applications with Geometric Parallelism. *Microprocessing and Microprogramming*, v.30(1-5), p.77-84, agosto, 1990.

JOOSEN, W.; BIJNENS, S.; VERBAETEN, P. A Reusable Load Balancer for Parallel Search Problems. *Microprocessing and Micropragramming*, v.38(1-5), p.205-212, setembro, 1993.

JOOSEN, W.; BIJNENS, S.; ROBBEN, B.; OEYEN, J.V.; VERBAETEN, P. Flexible load balancing software for parallel applications in a time-sharing environments. In: *High-Performance Computing and Networking, Lecture Notes in Computer Science v.919*, Milan, Italy, p.398-406, maio, 1995.

KANAMORI, A.; WEISE, D. Worklist Management Strategies for Dataflow Analysis. *Technical Report MSR-TR-94-12.* Microsoft Corporation, Advanced Technology Division. Redmond, WA, USA, maio, 1994.

KAPLAN, J.A.; NELSON, M.L. A comparison of Queueing, Cluster and Distributed Computing Systems. *Nasa Langley Research Center*, junho, 1994.

KEMELMAKHER, M.; KREMIEN, O. Scalable and Adaptive Resource Sharing in PVM. In: *5th European PVM/MPI User's Group Meeting, Lecture Notes in Computer Science v.1497*, Liverpool, UK, p.196-205, setembro, 1998.

KONURU, R.; CASAS, J.; OTTO, S.; PROUTY, R.; WALPOLE, J. A User-Level Process Package for Concurrent Computing. *Technical Report CSE-93-016*, agosto, 1993.

KRONE, O.; RAAB, M.; HIRSBRUNNER, B. Load Balancing for Network Based Multi-threaded Applications. In: *5th European PVM/MPI User's Group Meeting, Lecture Notes in Computer Science v.1497*, Liverpool, UK, p.206-214, setembro, 1998.

KRUEGER, P.; LAI, T.; DIXIT-RADIYA, V.A. Job Scheduling is More Important than Processor Allocation for Hypercube Computers. *IEEE Transactions on Parallel and Distributed Systems*, v.5, n.5, maio, 1994.

KRUGER, P.; SHIVARATRI, N.G. Adaptive location policies for global scheduling. *IEEE Transactions on Parallel and Distributed System*, v.20, Iss 6, p.432-444, 1994.

KUNZ, T. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. *IEEE Transactions on Software Engineering*, v.17, n.7, p.725-730, julho, 1991.

KWOK, Y-K.; AHMAD, I. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, v.7, n.5, maio, 1996.

LEWIS, T.; EL-REWINI, H. Parallax: A Tool for Parallel Program Scheduling. *IEEE Parallel & Distributed Technology*, pp-62-72, maio, 1993.

LIM, K.; KIM, C. Dynamic Load Balancing in Distributed Computer Systems with Star Topology. In: *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Cheju Island, Korea, p.514-519, agosto, 1995.

LIN, F.C.H.; KELLER, R.M. The Gradient Model Load Balancing Method. *IEEE Transactions on Software Engineering*, v.SE-13, n.1, janeiro, 1987.

LITZKOW, M.; TANNENBAUM, T.; BASNEY, J.; LIVNY, M. Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System. *Computer Sciences Technical Report #1346 of the University of Wisconsin-Madison*, abril, 1997.

LO, V.; RAJOPADHYE, S.; GUPTA, S.; KELDSEN, D.; MOHAMED, M.A.; NITZBERG, B.; TELLE, J.A.; ZHONG, X. OREGAMI: Tools for Mapping Parallel Computations to Parallel Architectures. *International Journal of Parallel Programming*, p.237-270, 1991.

LOH, P.K.K.; HSU, W.J.; WENTONG, C.; SRISKANTHAN, N. How Network Topology Affects Dynamic Load Balancing. *IEEE Parallel & Distributed Technology*, v.04 n.03, p.25-35, 1996.

LÜLING, R.; MONIEN, B.; RAMME, F. Load Balancing in Large Networks: A Comparative Study. *Relatório Técnico, Departamento de Matemática e Ciência de Computação*, Universidade de Paderborn, Alemanha, 1993.

LÜLING, R.; MONIEN, B. A Dynamic Distributed Load Balancing Algorithm with Provable Good Performance. In: *Proceedings of ACM Symposium on Parallel Algorithms and Architectures (SPAA-93)*, 1993.

MACHADO, F.B., MAIA, L.P. *Introdução à Arquitetura de Sistemas Operacionais*. Rio de Janeiro, LTC-Livros Técnicos e Científicos, 1992.

MARTIN, P. Research plan: a new taxonomy for load distribution systems. Public Research Page - Paul Martin Home Page. *Disponível no endereço:http://www.mcs.vuw.ac.nz/~pmar/Old-research/obj1.html*, 1996.

MASCARENHAS, E.; REGO, V. Migrant threads on process farms: parallel programming with Ariadne. *Concurrency-practice and experience*, v.10, Iss 9, p.673-698, 1998.

MEHRA, P.; WAH, B.W. Automated Learning of Load-Balancing Strategies for a Distributed Computer System. University of Illinois at Urbana-Champaign, 1993. *Disponível em ftp.ibr.cs.tu-bs.de*

MILOJICIC, D.S.; GIESE, P.; ZINT, W. Load Distribution on Microkernels. In: *IEEE Workshop "Future Trends in Distributed Computing Systems"*, setembro, 1993.

MULLENDER, S. *Distributed Systems*. 2a. ed., New York, Addison-Wesley Publishing Company, 1993.

MUNIZ, F.J. Parallel Load-Balancing on Message Passing Architectures. *Tese de Doutorado apresentada à Faculdade de Engenharia da Universidade de Southampton*, março, 1994.

NASTEA, S.G.; EL-GHAZAWI, T.; FRIEDER, O. Performance optimization of combined variable-cost computations and I/O. In: *Solving irregularly structured problems in parallel, Lecture Notes in Computer Science v.1253*, p.198-206, 1997.

NEUMAN, B.C.; RAO, S. Resource Management for Distributed Parallel Systems. In: *Proceedings of the 2nd Internationsl Syposium on High Performance Distributed Computing*, Spokane, p.316-323, julho, 1993.

NEUMAN, B.C.; RAO, S. The Prospero Resource Manager: A Scalable Framework for Processor Allocations in Distributed Systems. *Concurrency: Practice and Experience*, v.6(4), p.339-355, junho, 1994.

NIKOLOPOULOS D.S.; POLYCHRONOPOULOS, E.D.; PAPATHEODOROU, T.S. Enhancing the performance of autoscheduling in distributed shared memory multiprocessors. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.491-501, setembro, 1998.

NOSSAL, R. An evolutionary approach to multiprocessor scheduling of dependent tasks. In: *Proceedings of Parallel and Distributed Processing, Lecture Notes in Computer Science v.1388*, Orlando, Florida, USA, p.279-287, março/abril, 1998.

OLIKER, L.; BISWAS, R.; GABOW, H.N. Performance analysis and portability of the PLUM load balancing system. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.307-317, setembro, 1998.

ORLANDO, S.; PEREGO, R. Scheduling data-parallel computations on heterogeneous and time-shared environments. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.356-366, setembro, 1998.

OUSTERHOUT, J.K. Scheduling Techiniques for Concurrent Systems. In: *Proceedings Third International Conference on Distributed Computing Systems*, IEEE, p22-30, 1982.

OVEREINDER, B.J.; SLOOT, P.M.A.; HEEDERIK, R.N.; HERTZBERGER, L.O. A dynamic load balancing system for parallel cluster computing. *Future Generation Computer Systems - FGCS*, 12, p.101-115, 1996.

PAINDAVEINE, Y.; MILOJICIC, D.S. Process vs. Task Migration. In: *29th Annual Hawaii International Conference on System Sciences*, janeiro, 1996.

POLYCHRONOPOULOS, C.D.; GIRKAR, M.; HAGHIGHAT, M.R.; LEE, C.L.; LEUNG, B.; SCHOUTEN, D. Parafrase-2: An Environment for Parallelizing, Partitioning, Synchronizing and Scheduling Programs on Multiprocessors. In: *Proceedings of International Conference on Parallel Processing*, v.II, p.II-39-II48, 1989.

PROUTY, R.; OTTO, S.; WALPOLE, J. Adaptive Execution of Data Parallel Computations on Networks of Heterogeneous Workstations. *Technical Report CSE-94-012 Departament of Computer Science and Engineering*, Oregon Graduate Institute of Science & Technology, março, 1994.

PRUYNE, J.; LIVNY, M. Managing Checkpoints for Parallel Programs. In: *IPPS' 96 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1162*, Santa Barbara, Honolulu, Hawaii, abril, 1996.

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

RAPINE, C.; SCHERSON, I.D.; TRYSTRAM, D. On-line sheduling of parallelizable jobs. In: *Euro-Par' 98 Parallel Processing - Fourth International Euro-Par Conference, Lecture Notes in Computer Science v.1470*, Southampton, UK, p.322-327, setembro, 1998.

RENESSE, R.V.; STAVEREN, H.V.; TANENBAUM, A.S. The Performance of the Amoeba Distributed Operating System. *Software - Practice and Experience*, v.19(3), p.223-234, março, 1989.

ROBINSON, J.; RUSS, S.H.; FLACHS, B.; HECKEL, B. A Task Migration Implementation of the Message-Passing Interface. In: *IEEE conference proceedings of the 5 th High Performance Distributed Computing Conference (HPDC-5)*, OnCenter, Syracuse, New York, agosto, 1996.

RONDE, J.F.; SCHONEVELD, A.; SLOOT, P.M.A. Load Balancing by redundant decomposition and mapping. *Future Generation Computer Systems - FGCS*, 12, p.391-406, 1997.

RUSS, S.H.; FLACHS, B.; ROBINSON, J.; HECKEL, B. Hector: Automated Task Allocation for MPI. In: *10th International Parallel Processing Symposium in Honolulu, (IEEE conference proceedings)*, Hawaii, abril, 1996.

RUSS, S.H.; MEYERS, B.; ROBINSON, J.; GLEESON, M.; RAJAGOPALAN, L.; TAN, C-H.; HECKEL, B. User-Transparent Run-Time Performance Optimization. In: *EHPC'97 - the 2nd International Workshop on Embedded HPC Systems and Applications at the 11th IEEE International Parallel Processing Symposium*, 1997.

RUSS, S.H.; BANICESCU, I.; GHAFOOR, S.; JANAPAREDDI, B.; ROBINSON, J.; LU, R. Hectiling: An Integration of Fine and Coarse-Grained Load-Balancing Strategies. In: *7th IEEE International Symposium on High Performance Distributed Computing*, 1998.

SAPHIR, W.; TANNER, L.A.; TRAVERSAT, B. Job Management Requirements for NAS PArallel Systems and Clusters. In: *IPPS' 95 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.949*, Santa Barbara, CA, USA, abril, 1995.

SCHNOR, B. Dynamic Scheduling of Parallel Applications. In: *Third International conference PaCT-95, Lecture Notes in Computer Science v.964, St.Petersburg, Rússia,* p.109-116, setembro, 1995.

SCHNOR, B.; PETRI, S.; LANGENDÖRFER, H. Load Management for Load Balancing on Heterogeneous Plataforms: A Comparison of Traditional and Neural Network Based Approaches. In: *Second International Euro-Par Conference - Euro-Par'96, Lecture Notes in Computer Science v.1124*, Lyon, France, p.611-614, agosto, 1996.

SCHWIEGELSHOHN, U. Preemptive Weighted Completion Time Scheduling of Parallel Jobs. In: *Fourth Annual European Symposium - Algorithms - ESA' 96; Lecture Notes in Computer Science v.1136*, Barcelona, Espanha, setembro, 1996.

SEKHARAN, C.N.; GOEL, V.; SRIDHAR, R. Load balancing methods for ray tracing and binary tree computing using PVM. *Parallel Computing*, v.21, Iss 12, p.1963-1978, 1995.

SHIRAZI(a), B.A.; HURSON, A.R.; KAVI, K.M. Introduction to Scheduling and Load Balancing. *Introdução do Primeiro Capítulo do Livro Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, CA, p.2-6, USA, 1995.

SHIRAZI(b), B.A.; HURSON, A.R.; KAVI, K. M. Mechanisms for Process Migration. *Introdução do Sexto Capítulo do Livro Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, CA, USA, p.411-413, 1995.

SHIRAZI, B.; HURSON, A.R. Special Issue on Scheduling and Load Balancing: guest editor's introduction. *Journal of Parallel and Distributed Computing*, v.16, Iss 4, p.271-275, 1992.

SHIVARATRI, N.G.; KRUEGER, P.; SINGHAL, M. Load Distributing for Locally Distributed Systems. *IEEE Computer*, dezembro, 1992.

SHMOYS, D.B.; WEIN, J.; WILLIAMSON, D.P. Scheduling Parallel Machines On-Line. *SIAM Journal on Computing*, v.24, n.6, p.1313-1331, dezembro, 1995.

SHU, W.; WU, M-Y. Runtime Incremental Parallel Scheduling (RIPS) on Distributed Memory Computers. *IEEE Transactions on Parallel and Distributed Systems*, v.7, n.6, junho, 1996.

SIEGELL, B.S.; STEENKISTE, P.A. Automatic selection of load balancing parameters using compile-time and run-time information. *Concurrency-practice and experience*, v.9, Iss 4, p.275-317, 1997.

SMITH, P.; HUTCHINSON, N.C. Heterogenous Process Migration: The tui System. *Software - Practice and Experience*, v.28, n.6, p.611-639, 1998

SNIR, M.; OTTO, S.; STEVEN, H.; WALKER, D.; DONGARRA, J.J. *MPI: The Complete Reference*. The MIT Press, Massachusetts, 1996.

SOBALVARRO, P.G.; PAKIN, S.; WEIHL, W.E.; CHIEN, A.A. Dynamic Coscheduling on Workstation Clusters. In: *IPPS/SPDP' 98 Workshop on Job Scheduling Strategies for*

Parallel Processing, Lecture Notes in Computer Science v.1459, Orlando, Florida, USA, p.231-256, março, 1998.

SONG, J.; CHOO, H.K.; LEE, K.M. Application-level load migration and its implementation on top of PVM. *Concurrency: Practice and Experience*, v.9(1), p.1-19, janeiro, 1997.

SOUZA(a), P.S.L.; SANTANA, M.J.; SANTANA, R.H.C. A New Scheduling Environment for Near-Optimal Performance. In: *International Conference on Parallel and Distributed Processing Techniques and Applications – PDPTA'99*, Las Vegas, Nevada, U.S.A., junho, 1999.

SOUZA(b), P.S.L.; SANTANA, M.J.; SANTANA, R.H.C. AMIGO – A Dynamical Flexible Scheduling Environment. In: *5th International Conference on Information Systems Analysis and Synthesis – ISAS'99*, Orlando, U.S.A., julho, 1999.

SOUZA(c), P.S.L.; SANTANA, M.J.; SANTANA, R.H.C.; ARAÚJO, A.P.F. PVM and a Viable and Flexible Scheduling. In: *Eleventh IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS'99*, Cambridge, MA, USA, novembro, 1999.

SQUILLANTE, M.S. On the Benefits and Limitations of Dynamic Partitioning in Parallel Computer Systems. In: *IPPS' 95 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.949*, Santa Barbara, CA, USA, abril, 1995.

STATHOPOULOS, A.; YNNERMAN, A. Dynamic load balancing of atomic structure programs on a PVM cluster. *In: High-Performance Computing and Networking*, Milan, Italy, Lecture Notes in Computer Science v.919, p.384-391, maio, 1995.

STEENSGAARD, B.; JUL, E. Object and Native Code Process Mobility Among Heterogenous Computers. *Technical Report MSR-TR-95-17*. Microsoft Corporation, Advanced Technology Division. Redmond, WA, USA, março, 1995.

STELLNER, G.; TRINITIS, J. Load Balancing Based on Process Migration for MPI. *Euro-Par' 97 Parallel Processing - Third International Euro-Par Conference, Lecture Notes in Computer Science v.1300*, Passau, Germany, agosto, 1997.

TANENBAUM, A.S. *Modern Operating Systems*. New Jersey, Prentice Hall International, Inc., 1992.

TANENBAUM, A.S. *Distributed Operating Systems*. New Jersey, Prentice Hall International, Inc., 1995.

TANENBAUM, A.S. *Computer Networks*. 3. ed. New Jersey, Prentice-Hall Inc., 1996.

THEIMER, M.M.; HAYES, B. Heterogeneous Process Migration by Recompilation. In: *Proceedings IEEE 11th Int'l Conf. on Distributed Computing System, IEEE CS Press*, p.18-25, Los Alamitos, California, 1991.

TRIPATHI, A.R.; KARNIK, N.M. Trends in multiprocessor and distributed operating-systems designs. *Journal of supercomputing*, v.9, Iss 1-2, p.23-49, 1995.

VESSEUR, J.J.J.; HEEDERIK, R.N.; OVEREINDER, B.J.; SLOOT, P.M.A. Experiments in Dynamic Load Balancing for Parallel Cluster Computing. In: *Proceedings of the Workshop on Parallel Programming and Computation*, Amsterdam, p.189-194, junho , 1995.

WANG, J.; LI, J.; KAMEDA, H. Scheduling Algorithms for Parallel Transaction Processing Systems. In: 4th International Conference, PaCT'97 Yaroslaul, Parallel Computing Tecnologies, Lecture Notes in Computer Science v.1277, Rússia, setembro, 1997.

WANG, Y.T.; MORRIS, R.J.T. Load Sharing in Distributed Systems. *IEEE Transactions on Computers*, p.204-217, março, 1985.

WU, M.Y. On runtime parallel scheduling for processor load balancing. *IEEE Transactions on Parallel and Distributed Systems*, v.8, Iss 2, p.173-186, 1997.

XIROGIANNIS, G.; TAYLOR, H. A dynamic task distribution and engine allocation strategy for distributed execution of logic programs. In: *Proceedings of High-Performance Computing and Networking, Lecture Notes in Computer Science v.1401*, Amsterdan, The Netherlands, p.294-304, abril, 1998.

XU, C.; LAU, F.C.M. Load Balancing in Parallel Computers: Theory and Practice. *Kluwer Academic Publishers*, Boston, USA, 1997.

YAN, Y.; JIN, C.M.; ZHANG, X.D. Adaptively scheduling parallel loops in distributed shared-memory systems. *IEEE Transactions on Parallel and Distributed Systems*, v.8, Iss.1, p.70-81, 1997.

ZHOU, S.; ZHENG, X.; WANG, J.; DELISLE, P. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software: Practice and Experience*, v.23(12), p.1305-1336, dezembro, 1993.

ZHOU, B.B.; BRENT, R.P.; WALSH, D.; SUZAKI, K. Job Scheduling Strategies for Networks of Workstations. In: *IPPS/SPDP' 98 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science v.1459*, Orlando, Florida, USA, p.143-157, março, 1998.

Apêndice 1 – Relação de Novas Propostas de Algoritmos de Escalonamento

Este apêndice relaciona trabalhos que descrevem novas propostas de escalonamento. Mesmo não sendo uma listagem completa, devido à dificuldade em se encontrar "todas" as propostas feitas, a relação oferece um vasto material à pesquisa e demonstra a grande quantidade de trabalhos que descrevem novas maneiras de se escalonar processos.

Os trabalhos abaixo estão organizados em ordem alfabética do nome do algoritmo de escalonamento proposto. Quando o trabalho não cita um nome específico, é apresentada apenas a referência que o descreve.

- ADM - Prouty et al. (1994), Song et al. (1997);
- AMOEBA - Renesse et al. (1989), Tanenbaum (1995);
- ARIADNE SYSTEM - Mascarenhas; Rego (1998);
- AUTOESCHEDULING - Nikolopoulos et al. (1998);
- BALANCE SYSTEM - Hui; Chanson (1997);
- CAMAS - Ronde et al. (1997);
- CHARLOTTE - Artsy; Finkel (1989), Song et al. (1997);
- CILK - Blumofe et al. (1996);
- COCHECK - Pruyne; Livny (1996);
- CODINE (Computing in DIstributed Networked Environments) - Kaplan;Nelson (1994), Ferstl (1996);
- CONDOR - Kaplan; Nelson (1994), Epema et al. (1996), Litzkow et al. (1997);
- CONNECT - Kaplan; Nelson (1994);
- COOL (Concurrent Object-Oriented Language) - Chandra et al. (1993);
- DCP (Dynamic Critical-Path) - Kwok; Ahmad (1996);
- DCS (Dynamic CoScheduling) - Sobalvarro et al. (1998);

- DI (Difference-Initiated) - Antonis et al. (1998);
- DOME - Beguelin et al. (1996), Arabe et al. (1995);
- DJM (Distributed Job Manager) - Kaplan; Nelson (1994);
- DQS (Distributed Queueing System) - Kaplan; Nelson (1994);
- DYNAMICPVM - Overeinder et al. (1996), Vesseur et al. (1995);
- ESLR (Eager Scheduling with Lazy Retry) - Chen; King (1996);
- FCFSP (FCFS with Priority) - Wang et al. (1997);
- GA (Genetic Algorithms) - Nossal (1998);
- GRADIENTE - Lin; Keller (1987);
- GRM (General Resource Manager) - Fagg et al. (1996);
- HARNNES (Heterogeneous Adaptable RecoNfigurable NEtwork Systems) - Dongarra et al. (1998);
- HECTOR - Russ et al. (1996), Robinson et al. (1996), Russ et al. (1997),
- HiCon - Becker (1995);
- LACHESIS - Downey (1998);
- LBS - Krone et al. (1998);
- LOAD BALANCER - Kaplan; Nelson (1994);
- LOAD LEVELER - Kaplan; Nelson (1994);
- LRF (Least Recently Fired) - Kanamori; Weise (1994);
- LSF (Load Sharing Facility) - Kaplan; Nelson (1994);
- MACH - Milojicic et al. (1993), Paindaveine; Milojicic (1996), Tanenbaum (1995);
- MIST (Migration And Integrated Scheduling Tools) - Al-Saqabi et al. (1997), Casas et al. (1995b);
- MOSIX - Barak et al. (1996);
- MPVM - Song et al. (1997);
- MODELO ECONÔMICO - Chiclayo; Henriques (1999);
- MPVM (Migratable PVM) - Casas et al. (1995a), Clark et al. (1995);
- MSA (Multi-stage Sheduling Approach) - Boeres et al. (1998);
- NEAREST LAYER FIRST - Glazek (1999);
- NC TOOLSET - Kaplan; Nelson (1994);
- NQE (Network Queueing Environment) - Kaplan; Nelson (1994);

- OREGAMI - Lo et al. (1991);
- PAISS (PArallel Iterative linear System Solver) - Christen (1998);
- PAN (Prolog Area Network) - Xirogiannis; Taylor (1998);
- PARAFRASE-2 - Polychronopoulos et al. (1989);
- PARALLAX - Lewis; El-Rewini (1993);
- PARALEX - Davoli; Babaoglu (1996);
- PBS (Portable Batch System) - Henderson (1995);
- PIRANHA - Song et al. (1997);
- PLUM - Oliker et al. (1998);
- PMS (Processors Management System) - Aguilar; Jimenez (1997);
- PREP-P - Berman; Stramm (1994);
- PRM (Prospero Resource Manager) - Neuman; Rao (1993), Neuman; Rao (1994);
- RIALTO - Jones et al. (1997);
- RIPS (Runtime Incremental Parallel Scheduling) - Shu; Wu (1996);
- SCAN - Krueger et al. (1994);
- S-MPI (Selective MPI) - Dantas; Zaluska (1998);
- SITA-E - Harchol-Balter et al. (1997), Harchol-Balter; Downey (1997);
- SPAWN - Schnor (1995);
- SPRITE - Douglis (1989), Douglis; Ousterhout (1991);
- SUPLE - Orlando; Perego (1998);
- TASK BROKER - Kaplan; Nelson (1994);
- TOPSYS (TOols for Parallel SYStems) - Bemmerl (1990);
- TPM (Task-to-Processor Mapping) - Dikenelli et al. (1997);
- TUMPI - Stellner; Trinitis (1997);
- TUI SYSTEM - Smith; Hutchinson (1998);
- UPVM - Konuru et al. (1993);
- UTOPIA - Zhou et al. (1993);
- VASSAL - Candea; Jones (1998);
- V-SYSTEM - Tanenbaum (1995);
- VDS (Virtual Data Space) - Decker (1997);
- XENOOPS - Joosen et al. (1995);

- Berman et al. (1997);
- Boukerche; Das (1998);
- Brest et al. (1999);
- Bubak et al. (1995);
- Cherkasova (1998);
- Chiclayo; Henriques (1999);
- Chien et al. (1994);
- Colajanni et al. (1998);
- Corradi et al. (1998);
- Czajkowski et al. (1998);
- Durand et al. (1996);
- Fonlupt et al. (1998);
- Genco; Lore (1996);
- Gil; Matias (1996);
- Ha; Lee (1991);
- Halang et al. (1995);
- Hill et al. (1998);
- Hu et al. (1998);
- Huang; Ongsakul (1994);
- Jackson; Humphres (1997);
- Joosen et al. (1990);
- Joosen et al. (1993);
- Kemelmakher; Kremien (1998);
- Krueger; Shivaratri (1994);
- Lim; Kim (1995);
- Lüling; Monien (1993);
- Nastea et al. (1997);
- Sekharan et al. (1995);
- Shivaratri et al. (1992);
- Shmoys et al. (1995);

- Stathopoulos; Ynnerman (1995);
- Steensgaard; Jul (1995);
- Theimer; Hayes (1991);
- Wu (1997);
- Yan et al. (1997);
- Zhou et al. (1998);

NOTAS DO ICMC

SÉRIE COMPUTAÇÃO

045/2000 MARTINS, R.T.; RINO, L.H.M.; NUNES, M.G.V.; MONTILHA, G.; OLIVEIRA JR., O. N. – An interlingua diming at communication on the Web: how language-independent can it be?

044/99 FELTRIM, V.D.; FORTES,R.P.M. – Evaluation of a reverse engineering method through the application in a hypermedia system.

043/99 PANSANATO, L.T.E.; NUNES, M.G.V. – EHDM: Método para projeto de hiperdocumentos para ensino.

042/99 MORABITO, R.; ARENALES, M. – Optimizing the cutting of stock plates in a furniture company.

041/98 SANTOS-MEZA, E.; SANTOS, M.O.; ARENALES, M.N. – Lot sizing and scheduling in na automated foundry.

040/98 FELTRIM, V.D.; FORTES, R.P.M. – Uma modelagem do domínio de engenharia reversa de software utilizando o método OOHDM.

039/98 FERREIRA, V.G.; MELLO, O .D.; OLIVEIRA, J.N.; FORTUNA, A . O . - Tópicos teóricos e computacionais em escoamentos de fluidos.

038/98 CAVICHIA, M.C.; ARENALES, M.N. - Piecewise linear programming via interior points.

037/98 REZENDE, S. O. ; PUGLIESI, J.B. - Aquisição de conhecimento explícito ou manual - versão 1.0

036/97 SOSSOLOTE, C.R.C; RINO, L.H.M; ZAVAGLIA, C.; NUNES, M.G.V. - As manifestações morfossintáticas da linguagem UNL no português do Brasil.