SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 19.02.2004

Assinatura: *Ana Paula Jampai J. J. J.*

Uma abordagem para a avaliação do
escalonamento de processos em sistemas
distribuídos baseada em monitoração

Márcio Augusto de Souza

Orientadora: Profa. Dra. Regina Helena Carlucci Santana

Tese apresentada ao Instituto de Ciências Matemáticas e de
Computação - ICMC-USP, como parte dos requisitos para
obtenção do título de Doutor em Ciências – Ciências de
Computação e Matemática Computacional.

USP – São Carlos
Fevereiro de 2004

A Comissão Julgadora:

Profa. Dra. Regina Helena Carlucci Santana

Regina HC Santana

Prof. Dr. Paulo Cesar Masiero

Paulo Cesar Masiero

Prof. Dr. Mario Antonio Ribeiro Dantas

Mario Antonio Ribeiro Dantas

Prof. Dr. Alfredo Goldman vel Lejbman

Alfredo Goldman vel Lejbman

Prof. Dr. Jan Frans Willen Slaets

Jan Frans Willen Slaets

Dedicatória

*À Nadja e aos meus pais,
José Augusto e Marly*

Agradecimentos

À Prof. Dra. Regina Helena Carlucci Santana, pela orientação, amizade e paciência.

Aos Profs. Drs. Marcos José Santana, Paulo Sérgio Lopes de Souza e Sarita Mazzini Bruschi, pela amizade e colaboração a este trabalho.

À Nadja, por ter sido meu porto seguro nos momentos difíceis.

Ao Arion e ao Omar, pela amizade e companheirismo nos dois anos em que dividimos o mesmo teto. À Elaine e a Cidinha, por serem grandes amigas.

À família Pena (Sr. Duílio, D. Claudete, Duílio e Nadja) por me hospedarem e me tratarem muito bem durante o último ano de doutorado.

À Maria Alice e às minhas avós Ermenegilda e Esther, pelo apoio e confiança.

À Luiza, por ser uma excelente pessoa e amiga.

A todos os amigos que fiz no ICMC, os quais omitirei os nomes para não cometer a injustiça de esquecer alguém, o que é bastante provável que aconteça. Sem o apoio que todos vocês me deram, o meu caminho teria sido muito mais difícil.

À Universidade Estadual de Ponta Grossa, pela liberação concedida a mim.

À CAPES, pelo apoio financeiro.

Resumo

Um bom escalonamento de processos é de fundamental importância para o bom desempenho de um sistema computacional. Porém, a grande diversidade de objetivos de escalonamento, cargas de trabalho e tipos de sistema computacional tornam difícil a existência de um algoritmo de escalonamento que seja bom em todas as situações ou durante todo o tempo. Torna-se importante, portanto, que o software de escalonamento possa avaliar o desempenho do sistema computacional em resposta ao algoritmo de escalonamento utilizado.

O **PSMS** (*Process Scheduling Monitoring System* -- Sistema de Monitoração do Escalonamento de Processos), monitor que implementa a abordagem de monitoração proposta por esta tese, foi desenvolvido com o intuito de oferecer uma infra-estrutura de monitoração que pode ser usada por qualquer software de escalonamento e que permita que o escalonamento de processos possa ser avaliado de maneira sistemática, independente de seu objetivo.

O PSMS baseia sua avaliação em métricas de desempenho, as quais são escolhidas de acordo com os objetivos do escalonamento e são de fundamental importância dentro da abordagem de monitoração proposta. Também são definidas regras de decisão, que são aplicadas sobre as métricas com o intuito de retornar um valor que quantifique o desempenho do sistema de acordo com o seu objetivo de escalonamento.

Esta tese apresenta uma descrição detalhada da abordagem de monitoração proposta e do protótipo desenvolvido; faz uma análise crítica sobre a utilização de métricas de desempenho na avaliação do escalonamento de processo; e propõe algumas regras de decisão. São apresentados alguns estudos de caso relacionando regras de decisão e métricas, que demonstram a viabilidade da abordagem de monitoração aqui proposta.

Abstract

An efficient process scheduling is of vital importance to the performance of a computer system. However, the existence of so many scheduling goals, workload types and computer system architectures makes it very difficult for a scheduling algorithm to sustain a good performance. Therefore, it is very important for the scheduling software to be capable of evaluating the computer system performance in order to guarantee the accomplishment of its scheduling goal.

This thesis presents PSMS (Process Scheduling Monitoring System), an on-line monitoring system for evaluating the process scheduling. PSMS was designed to be flexible, and it is capable of communicating with any scheduler and of evaluating computer systems with different scheduling goals.

The evaluation performed by PSMS is based on the analysis of performance metrics, which are chosen based on scheduling goals. Decision rules are applied on these metrics, generating a performance value that quantifies the performance of the computer system. The scheduling software has access to this performance measurement and, in case of poor performance, it can change the scheduling algorithm or perform a process migration in order to fix the performance problem.

This thesis presents a detailed description of the proposed monitoring approach and of the prototype tool developed. It also presents an analysis of performance metrics commonly used in process scheduling evaluation and a description of some decision rules. Some case studies of the use of decision rules on metrics are also discussed, demonstrating the viability of the monitoring approach proposed.

Lista de Ilustrações

FIGURA 2.1 - TAXONOMIA HIERÁRQUICA DE CASAVANT E KUHL (1988) 14

FIGURA 3.1 – SOLUÇÕES DE UM MODELO 30

FIGURA 3.2 – RELAÇÃO ENTRE PROPOSTAS DE AVALIAÇÃO E TÉCNICAS DE AFERIÇÃO 33

FIGURA 3.3 - VISÃO EM CAMADAS DE UM MONITOR PARA UM SISTEMA DISTRIBUÍDO 38

FIGURA 3.4 - COMPONENTES DE UM MONITOR PARA UM SISTEMA DISTRIUÍDO..... 39

FIGURA 5.1 – VISÃO GERAL DA ARQUITETURA DO PSMS 56

FIGURA 5.2 – DIAGRAMA UML DE CASOS DE USO DO PSMS..... 58

FIGURA 5.3 – LOCALIZAÇÃO DOS COMPONENTES DO PSMS..... 59

FIGURA 5.4 – CAMADA DE OBSERVAÇÃO DO PSMS..... 60

FIGURA 5.5 – CAMADAS DE COLETA E DE ANÁLISE DO PSMS..... 68

FIGURA 5.6 – CAMADA DE INTERPRETAÇÃO DO PSMS 71

FIGURA 5.7 – DIAGRAMA DE ATIVIDADES DA FUNÇÃO DE ATUALIZAÇÃO DO PSMS 83

FIGURA 6.1 – RELAÇÃO ENTRE TIPOS DE SISTEMAS E MÉTRICAS DE DESEMPENHO 89

FIGURA 7.1 – ESTRUTURA DE DIRETÓRIOS DO PROTÓTIPO DO PSMS 105

GRÁFICO 7.1 – COEFICIENTES DE VARIAÇÃO -- EXPERIMENTO Nº 1..... 117

GRÁFICO 7.2 – TEMPO DE EXECUÇÃO DA APLICAÇÃO BT – EXPERIMENTO Nº 1 118

GRÁFICO 7.3 - COEFICIENTES DE VARIAÇÃO -- EXPERIMENTO Nº 2..... 121

GRÁFICO 7.4 – TEMPO DE EXECUÇÃO DAS APLICAÇÕES BT E LU – EXPERIMENTO Nº 2 121

GRÁFICO 7.5 – UTILIZAÇÃO DA MEMÓRIA – ESTUDO DE CASO Nº 2..... 125

Lista de Tabelas

TABELA 2.1 – RELAÇÃO DE SOFTWARES DE ESCALONAMENTO 24

TABELA 6.1. OBJETIVOS E MÉTRICAS RELACIONADOS À CPU 96

TABELA 6.2. OBJETIVOS E MÉTRICAS RELACIONADOS À MEMÓRIA 99

TABELA 7.1 – IMPLEMENTAÇÃO DOS COMPONENTES DO PSMS 106

TABELA 7.2 - MÉTRICAS OBTIDAS PELO PSMS 107

TABELA 7.3 – CONFIGURAÇÃO DO SISTEMA COMPUTACIONAL UTILIZADO NOS ESTUDOS DE CASO 110

TABELA 7.4 – PESOS DEFINIDOS PARA MÉDIA MÓVEL EXPONENCIAL DE ACORDO COM O TAMANHO DA JANELA 113

TABELA 7.5 – VALORES DE DESEMPENHO EM FUNÇÃO DO COEFICIENTE DE VARIAÇÃO DOS *SLOWDOWNS* . 115

TABELA 7.6 – DISTRIBUIÇÃO DE PROCESSOS - EXPERIMENTO Nº 1..... 116

TABELA 7.7 – DISTRIBUIÇÃO DE PROCESSOS - EXPERIMENTO Nº 2..... 120

TABELA 7.8. INTERPRETAÇÃO DOS VALORES DE FREQUÊNCIA DE CLASSES..... 124

TABELA 7.9 - CONFIGURAÇÃO DO SISTEMA COMPUTACIONAL 125

Sumário

1	INTRODUÇÃO.....	1
1.1	OBJETIVOS DESTA TESE DE DOUTORADO.....	3
1.2	ORGANIZAÇÃO DOS CAPÍTULOS.....	5
2	ESCALONAMENTO DE PROCESSOS EM SISTEMAS DISTRIBUÍDOS	7
2.1	CONSIDERAÇÕES INICIAIS	7
2.2	SISTEMAS DISTRIBUÍDOS	8
2.3	ESCALONAMENTO DE PROCESSOS.....	10
2.4	TAXONOMIA PARA A ÁREA DE ESCALONAMENTO DE PROCESSOS	12
2.4.1	<i>Classificação hierárquica</i>	<i>13</i>
2.4.2	<i>Classificação Plana.....</i>	<i>15</i>
2.5	ORGANIZAÇÃO DE UM ALGORITMO DE ESCALONAMENTO.....	16
2.6	OBJETIVOS E DESEMPENHO DE UM ESCALONAMENTO DE PROCESSOS.	19
2.6.1	<i>Métricas de desempenho</i>	<i>20</i>
2.7	SOFTWARES DE ESCALONAMENTO	22
2.7.1	<i>Estrutura dos Softwares de Escalonamento.</i>	<i>25</i>
2.8	CONSIDERAÇÕES FINAIS	26
3	AValiação DE DESEMPENHO PARA O ESCALONAMENTO DE PROCESSOS 28	
3.1	CONSIDERAÇÕES INICIAIS	28
3.2	TÉCNICAS DE MODELAGEM	29
3.3	TÉCNICAS DE AFERIÇÃO	31
3.4	MONITORAÇÃO EM SISTEMAS COMPUTACIONAIS	34
3.4.1	<i>Monitoração em Sistemas Distribuídos.....</i>	<i>37</i>
3.5	CONSIDERAÇÕES FINAIS	40
4	MONITORAÇÃO APLICADA AO ESCALONAMENTO DE PROCESSOS	42
4.1	CONSIDERAÇÕES INICIAIS	42
4.2	APLICAÇÕES DE MONITORAÇÃO AO ESCALONAMENTO – ESTADO DA ARTE.....	43
4.3	AVALIANDO O ESCALONAMENTO DE PROCESSOS ATRAVÉS DE MONITORAÇÃO	45
4.4	SOFTWARES DE MONITORAÇÃO PARA O ESCALONAMENTO DE PROCESSOS.....	47
4.4.1	<i>NWS – Network Weather Service</i>	<i>48</i>
4.4.2	<i>AutoPilot.....</i>	<i>49</i>
4.5	CONSIDERAÇÕES FINAIS	50
5	PSMS – UM SISTEMA DE MONITORAÇÃO PARA O ESCALONAMENTO DE PROCESSOS	51
5.1	CONSIDERAÇÕES INICIAIS	51
5.2	OBJETIVOS DE PROJETO	52
5.2.1	<i>Sistema Computacional Alvo.....</i>	<i>53</i>
5.3	ARQUITETURA DO PSMS	55
5.4	CAMADA DE OBSERVAÇÃO	59
5.4.1	<i>Funções de Obtenção de Métricas</i>	<i>63</i>
5.4.2	<i>Descrição dos Casos de Uso Desempenhados pela Camada de Observação.....</i>	<i>65</i>
5.4.2.1	<i>Caso de Uso: Coletar Métrica</i>	<i>65</i>
5.4.2.2	<i>Caso de Uso: Executar Benchmark.....</i>	<i>66</i>

5.5	CAMADAS DE COLETA E DE ANÁLISE	66
5.6	CAMADA DE INTERPRETAÇÃO.....	70
5.6.1	<i>Descrição do Caso de Uso Desempenhado pelas Camadas de Coleta, de Análise e de Interpretação</i>	<i>74</i>
5.6.1.1	Caso de Uso: Utilizar Regras de Decisão.....	75
5.7	CAMADA DE CONSOLE	76
5.7.1	<i>Descrição dos Casos de Uso Desempenhados pela Camada de Console.....</i>	<i>78</i>
5.7.1.1	Caso de Uso: Atualizar PSMS	78
5.7.1.2	Caso de Uso: Inserir Métrica.....	78
5.7.1.3	Caso de Uso: Inserir Regra de Decisão.....	79
5.7.1.4	Caso de Uso: Atualizar Tabela de Métricas/Regras.....	79
5.8	COMUNICAÇÃO ENTRE O PSMS E O SOFTWARE DE ESCALONAMENTO	80
5.8.1	<i>Descrição dos Casos de Uso Envolvidos na Comunicação entre o Software de Escalonamento e o PSMS.....</i>	<i>81</i>
5.8.1.1	Caso de Uso: Requisitar Monitoração.....	81
5.8.1.2	Caso de Uso: Buscar Valor de Desempenho.....	81
5.9	DIAGRAMAS DE ATIVIDADES DO PSMS.....	82
5.10	CONSIDERAÇÕES FINAIS	85
6	ANÁLISE DE MÉTRICAS DE DESEMPENHO APLICADAS AO ESCALONAMENTO DE PROCESSOS.....	86
6.1	CONSIDERAÇÕES INICIAIS	86
6.2	ANÁLISE CRÍTICA DAS MÉTRICAS DE DESEMPENHO	87
6.2.1	<i>Métricas de Desempenho e Tipos de Sistema Computacional</i>	<i>87</i>
6.2.2	<i>Métricas de Desempenho e Objetivos de um Escalonamento de Processos</i>	<i>89</i>
6.2.3	<i>Tempo de Resposta e Slowdown.....</i>	<i>91</i>
6.2.4	<i>Métricas Relacionadas à CPU.....</i>	<i>95</i>
6.2.5	<i>Métricas Relacionadas à Memória</i>	<i>97</i>
6.2.6	<i>Outras Métricas.....</i>	<i>99</i>
6.2.7	<i>Métricas e Sistemas Distribuídos Heterogêneos</i>	<i>100</i>
6.3	CONSIDERAÇÕES FINAIS	101
7	PROTÓTIPO E ESTUDOS DE CASO DO PSMS.....	103
7.1	CONSIDERAÇÕES INICIAIS	103
7.2	PROTÓTIPO DO PSMS.....	104
7.2.1	<i>Descrição da Implementação.....</i>	<i>105</i>
7.2.2	<i>Sobrecarga do PSMS Sobre o Sistema Computacional</i>	<i>107</i>
7.3	ESTUDOS DE CASO	109
7.3.1	<i>Estudo de Caso nº 1 -- Otimizando a Utilização da CPU.....</i>	<i>114</i>
7.3.2	<i>Estudo de Caso nº 2 -- Otimizando a Utilização da Memória.....</i>	<i>122</i>
7.4	CONSIDERAÇÕES FINAIS	127
8	CONCLUSÕES.....	130
8.1	PONDERAÇÕES FINAIS DESTA TESE.....	130
8.2	CONTRIBUIÇÕES DESTA TESE	133
8.3	PROPOSTAS DE TRABALHOS FUTUROS.....	134
9	REFERÊNCIAS BIBLIOGRÁFICAS.....	136
	APÊNDICE A – GLOSSÁRIO.....	147
	APÊNDICE B – NOÇÕES DE ESTATÍSTICA	149
9.1	MEDIDAS DE LOCAÇÃO	149
9.1.1	<i>Média.....</i>	<i>150</i>

9.2	MEDIDAS DE DISPERSÃO OU VARIABILIDADE	152
9.2.1	<i>Variância e Desvio Padrão</i>	152
9.2.2	<i>Coeficiente de Variação</i>	153
9.3	REFERÊNCIAS BIBLIOGRÁFICAS	153

1 Introdução

O progresso tecnológico atual tem permitido à computação passar por profundas transformações, resultando em uma grande evolução em um tempo relativamente pequeno. CPUs tornam-se mais rápidas, a capacidade da memória aumenta, redes transmitem maior volume de dados com maior velocidade, e todo esse progresso é acompanhado por uma gradual diminuição de custo, tornando a tecnologia computacional cada vez mais acessível. Dentro desse contexto, uma área da computação que sofreu uma grande evolução nos últimos anos foi a computação distribuída.

Os sistemas computacionais distribuídos foram desenvolvidos com o intuito de serem uma alternativa ao modelo centralizado de computação (baseado em um computador de grande porte compartilhado entre diversos usuários) e de início eram basicamente utilizados para compartilhar recursos.

Nas últimas décadas, porém, o uso de sistemas distribuídos se difundiu de tal maneira que é muito difícil quantificar a influência que a computação distribuída exerce sobre a ciência da computação como um todo. São diversas as funções que podem ser desempenhadas por um sistema distribuído, e diversos os tipos de sistemas computacionais que se baseiam no modelo de computação distribuída, como por exemplo, os *clusters* de computadores e o *grid*.

Sistemas distribuídos também podem ser utilizados para executar aplicações paralelas através, por exemplo, de bibliotecas de programação que contenham rotinas de comunicação através de troca de mensagens, como o MPI (*Message Passing Interface*) e o PVM (*Parallel Virtual Machine*). Sistemas distribuídos são uma alternativa que possui uma relação custo/benefício atrativa em relação a computadores paralelos, tornando a computação paralela acessível para um número muito maior de usuários.

Utilizar um sistema distribuído com eficiência implica em compartilhar, de maneira eficiente, os processadores disponíveis entre todos os usuários. Para que essa atividade seja feita de maneira organizada, deve haver uma ferramenta¹ (que pode ser implementada por

¹ Software ou hardware que desempenha uma função específica dentro de um sistema computacional

software, hardware ou de uma maneira híbrida) que controle o acesso a esses processadores. Ou seja, deve haver um componente do sistema computacional que seja responsável pelo escalonamento dos processos.

Diversos tipos de aplicações existem (interativas, *batch*, paralelas, tempo real, etc.) com diferentes necessidades em termos de recursos (processadores, memória, discos rígidos, etc.) necessários para sua execução, e o escalonamento de processos deve idealmente satisfazer a todos os usuários, o que não é uma tarefa simples de ser feita. O grande desafio para o software de escalonamento (que, neste trabalho, será definido como o conjunto de software responsável pelo escalonamento, e é sinônimo de escalonador de processos) é garantir um sistema com bom desempenho diante de um conjunto tão heterogêneo de aplicações

Um escalonamento de processos é feito de acordo com um determinado objetivo, e como se define neste trabalho, é necessário que o escalonamento alcance os seus objetivos para que um sistema computacional atinja um bom desempenho. Ou seja, como podem ser definidos diferentes objetivos, existem, portanto, diversas definições de bom desempenho.

A diversidade de objetivos de um escalonamento, os diferentes tipos de aplicações, e a heterogeneidade dos processadores componentes de um sistema distribuído são características que, combinadas, geram uma grande variedade de possíveis perfis para um sistema computacional, o que impossibilita que exista um algoritmo de escalonamento que seja eficiente para toda e qualquer situação.

Além disso, é comum que ocorram variações na carga de trabalho de um sistema distribuído, de maneira que um determinado escalonamento pode deixar de ser eficiente no decorrer do tempo. Portanto, é importante que exista uma forma de avaliar o desempenho do escalonamento enquanto ele é efetuado, para que se possa concluir se os objetivos aos quais o software de escalonamento se propõe estão sendo alcançados.

Diversas técnicas de avaliação de desempenho podem ser utilizadas para avaliar o desempenho de um escalonamento de processos. A literatura consultada discute diversas técnicas de avaliação de desempenho aplicadas ao escalonamento de processos, como por exemplo: modelagem (utilizando simulação ou técnicas analíticas para a solução dos modelos) (Buyya & Murseh, 2002; Hlavacs & Ueberhuber, 1999; Schnor et al., 1996;

Souza, 2000(b); Xiao et al., 2003; Zhang et al., 2000) e *benchmarks* (Feitelson & Rudolph, 1998; Chapin et al., 1999). Ambas técnicas, porém, não são adequadas ao objetivo de avaliar o algoritmo de escalonamento durante a sua execução. Avaliação de desempenho feita com dados coletados no próprio sistema computacional avaliado é uma tarefa para técnicas de monitoração.

Diversos trabalhos discutem a utilização de monitoração para avaliar o escalonamento de processos (Arpaci-Dusseau et al., 1998; Frachtenberg et al., 2003; Schnor & Petri, 1999; Silva & Scherson, 2000; Devarakonda & Iyer, 1989; Smith et al., 1998). Alguns trabalhos (Berman et al., 2003(a); Ribler et al., 2001; Wolski et al., 1999) propõem que os dados sejam interpretados no momento em que são coletados, caracterizando uma **monitoração on-line**, abordagem seguida por esta tese de doutorado.

Dentro desse contexto, esta tese propõe uma abordagem de monitoração *on-line*, flexível e de propósito geral, desenvolvida com o objetivo de permitir que um sistema distribuído seja monitorado de maneira que se colem dados que forneçam subsídios para que se possa avaliar de maneira segura o quanto o escalonamento está sendo bem feito em relação ao objetivo estipulado.

Através dessa avaliação, o software de escalonamento pode manter-se constantemente informado a respeito do desempenho do sistema computacional. Dessa maneira, pode-se tomar medidas corretivas que permitam corrigir o escalonamento em caso de mau desempenho através, por exemplo, de migrações de processo ou da troca do algoritmo de escalonamento. Procura-se assim garantir que o sistema computacional seja utilizado com bom desempenho na maior parte do tempo.

1.1 Objetivos Desta Tese de Doutorado.

Esta tese propõe uma abordagem de monitoração cujo objetivo é avaliar o escalonamento de processos. Essa abordagem, a qual foi implementada em um protótipo cujo nome é **PSMS** (*Process Scheduling Monitoring System* – Sistema de Monitoração do Escalonamento de Processos), foi desenvolvida com o intuito de ser genérica, ou seja, pode ser utilizada por qualquer software de escalonamento, independente de seu objetivo de escalonamento. Por isso, o PSMS foi projetado para ser altamente flexível, e pode ser

adaptado de acordo com as necessidades de um determinado sistema computacional distribuído.

O PSMS baseia sua avaliação em **métricas de desempenho**, as quais são escolhidas de acordo com os objetivos do escalonamento, permitindo que se possa quantificar o desempenho do sistema distribuído. A escolha de métricas corretas garante que os resultados obtidos pelo uso do PSMS sejam corretos. Em virtude de sua importância dentro da abordagem de monitoração proposta, esta tese apresenta uma análise crítica de métricas de desempenho comumente encontradas em trabalhos científicos de computação, discutindo a sua adequação ao tipo de monitoração efetuada pelo PSMS.

Além de coletar as métricas, o PSMS também define **regras de decisão**, que são aplicadas sobre as métricas com o intuito de retornar um valor que represente o quanto o desempenho do sistema está bom, ou seja, o quanto os objetivos do escalonamento de processos estão sendo alcançados. Algumas regras de decisão são propostas nesta tese, e a sua utilização prática é exemplificada através de alguns estudos de caso.

De maneira sucinta, os objetivos que guiaram o desenvolvimento do trabalho de doutorado descrito nesta tese são:

- Proposta de uma abordagem de monitoração de propósito geral para avaliar o escalonamento de processos;
- Análise crítica das métricas de desempenho, analisando a sua adequação à abordagem de monitoração proposta, e também discutindo a relação entre métricas de desempenho e objetivos de um escalonamento;
- Proposta de um esquema de interpretação das métricas de desempenho, baseado na utilização de regras de decisão, que retorne um valor que quantifique o desempenho do sistema computacional de acordo com o objetivo do escalonamento de processos;
- Implementação de um protótipo para a validação das idéias propostas nesta tese, nomeado PSMS.

O trabalho de doutorado descrito nesta tese está organizado em 8 capítulos e 4 apêndices, conforme é detalhado na próxima seção.

1.2 Organização dos Capítulos

Esta tese é dividida em 8 capítulos, sendo que o primeiro é esta introdução, que apresenta a área da computação em que o trabalho está inserido e as principais contribuições da tese de doutorado desenvolvida.

No segundo capítulo é apresentada uma introdução ao escalonamento de processos em sistemas distribuídos, incluindo conceitos básicos sobre sistemas distribuídos e escalonamento de processos. A revisão bibliográfica apresentada nesse capítulo procura dar um maior enfoque aos aspectos relacionados a este trabalho. Também, nesse capítulo, são apresentados alguns exemplos de softwares de escalonamento.

No capítulo 3 são discutidas as técnicas de avaliação de desempenho mais comuns encontradas na literatura computacional, discutindo sua aplicação à atividade de escalonamento de processos. Uma seção é destinada ao estudo da monitoração, mostrando alguns aspectos relacionados à utilização de monitores em sistemas distribuídos.

O capítulo 4 discute as diversas aplicações da monitoração à atividade de escalonamento de processos, mostrando alguns trabalhos científicos correlatos, de maneira a contextualizar o monitor descrito nesta tese de doutorado no cenário de pesquisa em escalonamento de processos.

A abordagem de monitoração e as principais características do monitor proposto são apresentadas no capítulo 5 em detalhes, incluindo a especificação do monitor através de alguns diagramas da linguagem UML. Também são apresentados alguns exemplos de regras de decisão.

O capítulo 6 apresenta uma análise crítica das métricas de desempenho, discutindo a sua adequação à abordagem de monitoração proposta por esta tese de doutorado, e mostrando alguns exemplos da combinação entre objetivos de escalonamento e métricas de desempenho.

No capítulo 7 discute-se o protótipo desenvolvido baseado na abordagem de monitoração proposta, e também são apresentados alguns estudos de caso de seu uso, utilizando métricas de desempenho relacionadas à CPU e também à memória de um computador.

Finalmente, no capítulo 8 são apresentadas as ponderações finais desta tese de doutorado, juntamente com as principais contribuições e também as propostas de trabalhos futuros.

Esta tese traz também três apêndices, com os seguintes conteúdos: um glossário dos termos utilizados nesta tese; uma descrição das rotinas estatísticas utilizadas pelo PSMS; uma descrição simplificada da linguagem de modelagem UML.

2 Escalonamento de Processos em Sistemas Distribuídos

Este capítulo aborda os principais tópicos sobre escalonamento de processos em sistemas distribuídos, e é organizado em duas partes. Inicialmente, discutem-se brevemente as principais características de um sistema distribuído; em seguida, são apresentados os conceitos relacionados ao escalonamento de processos, procurando enfocar a sua utilização em sistemas distribuídos.

As seções que discutem o escalonamento de processos dividem-se em: taxonomia para escalonamento de processos, componentes de um algoritmo de escalonamento, desempenho do escalonamento e exemplos de softwares de escalonamento.

2.1 *Considerações Iniciais*

O ato de escalonar é uma tarefa bastante comum na vida cotidiana, e em muitas situações corriqueiras do dia-a-dia enfrentam-se problemas que requerem decisões de escalonamento. Por exemplo, na limpeza de uma casa, geralmente dispõe-se de apetrechos de limpeza (baldes, vassouras, etc.) e pessoas que os utilizarão. Para que o trabalho seja feito de maneira organizada, deve-se distribuir os apetrechos entre as diversas pessoas, e cada uma deve ser responsável por um determinado cômodo, ou uma determinada tarefa. Dessa maneira, o processo de dividir as tarefas entre as diversas pessoas pode ser visto como um “escalonamento de tarefas”.

Essa analogia pode ser facilmente estendida para os sistemas computacionais, substituindo-se pessoas por processadores e considerando que as diversas tarefas de limpeza sejam diferentes processos computacionais. Quando se possui um ou mais processadores e diversos processos, deve existir uma maneira de distribuir esses processos de maneira organizada e eficiente no(s) processador(es).

O escalonamento de processos tem sido muito estudado pela comunidade acadêmica nos últimos anos, visto que é necessário em diversas áreas e diversos níveis do processo computacional, mas adquiriu maior importância com o advento da utilização de sistemas distribuídos. Esses sistemas possuem um grande potencial de desempenho para diversos tipos de aplicações, mas trazem junto a esse potencial uma grande dificuldade para

o escalonamento dos processos nos diversos (e possivelmente heterogêneos) processadores, de maneira que se consiga uma utilização eficiente do sistema computacional disponível.

Em outras palavras, não basta ter um sistema distribuído disponibilizando um grande poder computacional e utilizá-lo sem nenhum critério, pois acarretará um benefício bem inferior ao que o sistema computacional poderia oferecer.

Na próxima seção, são apresentados alguns conceitos relacionados a sistemas distribuídos, fornecendo um contexto para apresentar uma visão geral sobre escalonamento de processos em sistemas distribuídos, apresentada na seção 2.3.

2.2 Sistemas Distribuídos

Um sistema distribuído caracteriza-se pela interconexão de vários processadores autônomos, possivelmente heterogêneos, que sob o controle de um software distribuído idealmente fornece ao usuário a imagem de um sistema único, ou seja, sob o ponto de vista do usuário, a distribuição dos componentes deve ser a mais transparente possível. Dentre as características desejáveis que um sistema distribuído deve possuir podem ser citadas: compartilhamento de recursos, capacidade de aumentar o número de recursos em escala (*scalability*), tolerância a falhas e heterogeneidade. (Coulouris et al., 2001; Mullender, 1993; Tanenbaum & Steen, 2001)

Sistemas distribuídos podem ser utilizados em uma grande variedade de aplicações, tanto em ambientes comerciais quanto acadêmicos, e podem ser implementados desde em redes locais (LAN¹) até redes de longa distância (WAN²). Uma abordagem bastante utilizada nos dias atuais é a utilização de sistemas distribuídos para a execução de aplicações paralelas, permitindo que instituições que não dispõem de um computador paralelo possam construir e executar aplicações paralelas. Uma abordagem de programação paralela em sistemas distribuídos bastante comum é a chamada **programação através de troca de mensagens** (Baker & Buyya, 1999; Souza, 1997; McBryan, 1994), e é oferecida por softwares como o PVM e o MPI (Beguelin et al, 1994; Snir et al, 1998).

¹ *Local Area Network* – Rede Local de Computadores

² *Wide Area Network* – Rede Metropolitana de Computadores

Sistemas distribuídos podem ser organizados seguindo basicamente três modelos (Tanenbaum & Steen, 2001). O primeiro consiste em organizar o sistema como um conjunto de estações de trabalho (*workstations*) e/ou computadores pessoais conectados através de uma rede de comunicação de alta capacidade. Esse enfoque é bastante utilizado atualmente (por exemplo, em uma rede de computadores pessoais executando um sistema operacional *UNIX-like*), e é bastante vantajoso por ser simples e de custo/benefício bastante atrativo. Algumas variações desse modelo citadas em trabalhos científicos são: a NOW (*Network of workstations* – Redes de estações de trabalho) e o *Cluster* de Computadores (*Computer Clusters*) (Anderson et al., 1995; Baker & Buyya, 1999).

Uma importante característica desse modelo é que durante a maior parte do tempo, enquanto algumas estações de trabalho estão sobrecarregadas, outras estão livres ou pouco carregadas, gerando um aproveitamento não eficiente dos recursos computacionais disponíveis (Basney & Livny, 1999; Livney & Melman, 1982). Portanto, existe um grande potencial de aumento de desempenho pela utilização dessas estações livres, e o estudo do escalonamento de processos tem sido direcionado para essa área. Alguns autores definem um sistema onde se procura aproveitar o máximo de recursos disponíveis como um sistema de computação de alto *throughput* (*High throughput computing*) (Basney & Livny, 1999; Thain et al., 2003).

O segundo modelo é definido como *processor pool* (banco de processadores), que é um conjunto de processadores, disponíveis em um compartimento separado (por exemplo, um computador paralelo), que são atribuídos aos usuários de acordo com a demanda. O *Beowulf*, que consiste em um grupo de computadores pessoais interligados através de uma rede de alta velocidade, executando um software livre (*Linux*, por exemplo) e destinado apenas a aplicações de alta performance, pode ser citado com um exemplo de um banco de processadores (Baker & Buyya, 1999).

O terceiro modelo caracteriza-se pela utilização de um modelo híbrido entre os dois modelos citados. Esse modelo é utilizado por grande parte dos sistemas distribuídos acadêmicos.

Uma abordagem diferente de organização para sistemas distribuídos é utilizar a computação em *Grid* (*Grid Computing*), que se caracteriza pela execução de aplicações

paralelas de grande porte em processadores distribuídos por diferentes “domínios administrativos”, ou seja, os processadores pertencem a diferentes organizações e instituições de pesquisa, com diferentes políticas de utilização e interconectadas por redes de grande escala (WAN) ou pela Internet (Berman et al., 2003(b); Buyya & Murshed, 2002; Thain et al., 2003).

Uma característica importante de um sistema distribuído é a heterogeneidade, ou seja, a existência de processadores heterogêneos conectados no mesmo sistema. A heterogeneidade pode ser dividida em dois níveis: processadores de igual arquitetura, mas com diferentes capacidades em termos de recursos (quantidade de memória, velocidade da CPU³, etc.) (heterogeneidade configuracional); processadores de diferentes arquiteturas (heterogeneidade arquitetural) (Zhou et al., 1993).

Na última década, com o desenvolvimento de tecnologias de interconexão de processadores bastante avançadas e com a existência de protocolos de comunicação padronizados - como o TCP/IP⁴ (Comer, 1998), por exemplo - é comum que sistemas distribuídos sejam heterogêneos nos dois níveis citados acima (configuracional e arquitetural). Essa heterogeneidade é um fator complicador para a gerência do sistema distribuído (incluindo aí o escalonamento de processos), mas a flexibilidade que um sistema computacional heterogêneo pode oferecer é uma característica que torna atrativa a sua utilização (Beguelin et al., 1994; Eshaghian, 1996).

Na próxima seção, são apresentados os principais conceitos relacionados ao escalonamento de processos, procurando dar maior ênfase aos conceitos que sejam diretamente relacionados à utilização do escalonamento em sistemas distribuídos.

2.3 Escalonamento de Processos

Como foi exposto anteriormente, o escalonamento de processos se aplica a um ou mais processadores. Porém, de acordo com o número de processadores disponíveis, pode-se definir dois tipos de escalonamento possíveis: local e global (Casavant & Kuhl, 1988).

³ *Central Processing Unit* ou UCP (Unidade Central de Processamento)

⁴ *Transport Control Protocol/Internet Protocol*

O escalonamento local refere-se à atribuição de processos concorrentes a apenas um processador, fato que é comum em sistemas operacionais multitarefa executando em computadores pessoais. Geralmente, esse tipo de escalonamento é feito pelo sistema operacional, e baseia-se no compartilhamento do processador através de técnicas de *time-sharing*, ou seja, o tempo de uso do processador é dividido em pequenas fatias de tempo (*time-slice*) que são utilizadas pelos diversos processos ordenadamente. Dessa maneira, permite-se uma utilização eficiente e justa do processador disponível. Para garantir a imparcialidade, o sistema operacional cuida da suspensão e ativação dos processos através de uma técnica chamada de preempção. Técnicas de escalonamento local são abordadas detalhadamente em Machado e Maia (2003) e Tanenbaum (2003).

Quando se possui mais de um processador, o escalonamento é chamado de global e implica em fazer a distribuição dos processos entre os processadores disponíveis. Casavant e Kuhl (1988) definem formalmente o problema do escalonamento global enxergando-o como um “recurso para o gerenciamento de recursos”. Dessa forma, pode-se dividir o problema do escalonamento de processos em três principais componentes:

- Consumidores (processos dos usuários e do sistema)
- Recursos (CPU, memória, rede de comunicação, etc.)
- Algoritmo de escalonamento

Os consumidores utilizam recursos computacionais e o algoritmo de escalonamento é responsável por distribuir os processos entre os processadores de acordo com as necessidades e a disponibilidade de recursos. Complementando essa definição, Shirazi e Hurson (1992), Baumgartner e Wah (1991) e Krallman e Schwigelshohn (1999) afirmam que essa distribuição de processos é atrelada a um objetivo, baseado no qual todas as decisões de escalonamento são tomadas. Exemplos de possíveis objetivos são: diminuir o tempo médio de resposta dos processos e balancear a carga do sistema. Os diferentes objetivos que podem estar atrelados a um escalonamento são discutidos na seção 2.6.

Casavant e Kuhl (1988) definem escalonamento de processos como sinônimo de gerenciamento de recursos, o que muitos autores consideram ser atividades diferentes. Esse desacordo em relação à terminologia é um problema comum nos trabalhos encontrados

sobre escalonamento de processos. Diferentes autores utilizam diferentes nomes para designar muitas vezes atividades computacionais bastante semelhantes. Tanenbaum (1995), por exemplo, utiliza o termo alocação de processadores com um sentido semelhante ao de escalonamento de processos definido neste trabalho. Existem autores que utilizam o termo balanceamento de cargas como sinônimo para a atividade de escalonamento, como por exemplo, Lüling et al. (1993). A distinção entre escalonamento de processos e balanceamento de carga é muito importante uma vez que cada uma dessas atividades possui objetivos distintos. O problema da terminologia na área de escalonamento de processos é discutido em detalhes por Souza et al. (2000).

O escalonamento de processos é implementado através de um algoritmo de escalonamento, e este é organizado como um conjunto de componentes, de maneira que se tenha uma visão modular do algoritmo. Não existe consenso entre os autores a respeito dos componentes de um algoritmo de escalonamento, mas uma abordagem comum é dividi-lo em políticas (ver seção 2.5), que implementam diferentes partes do processo de escalonamento, facilitando seu projeto e compreensão (Muniz, 1994; Shirazi et al., 1995(a); Shivaratri et al., 1992).

Existe um conjunto de atividades comuns para o escalonamento em geral, que são chamados mecanismos, e que são usados de diferentes maneiras e em diferentes combinações para implementar as políticas. Um exemplo de um mecanismo é a migração de processos, que corresponde ao transporte de um processo (ou partes de um processo) em execução de um processador origem para um processador destino (Tripathi & Karnik, 1995; Wibisono, 2002). Por isso, pode se dizer que um algoritmo de escalonamento implementa políticas, que por sua vez utilizam diversos mecanismos (Song et al., 1997).

A fim de permitir uma definição concisa e coerente para os termos relacionados ao escalonamento de processos, este trabalho utiliza a taxonomia de Casavant e Kuhl (1988) como ponto de referência, e essa taxonomia é apresentada na seção seguinte.

2.4 Taxonomia para a Área de Escalonamento de Processos

Diversas taxonomias têm sido propostas com o passar dos anos, a fim de permitir a organização do conhecimento adquirido sobre todos os aspectos envolvidos na atividade de

escalonamento de processos, e também permitindo comparar os trabalhos desenvolvidos na área (Souza et al., 2000). Algumas taxonomias são relativamente genéricas (Casavant & Kuhl, 1988; Baumgartner & Wah, 1991); outras foram definidas para classificar algoritmos de escalonamento que têm como objetivo balancear a carga do sistema (Lüling et al., 1993; Xu & Lau, 1997); outras são aplicadas apenas a computadores paralelos (Quinn, 1994; Feitelson & Rudolph, 1995).

Em virtude de ser bastante referenciada e ser apropriada para os objetivos deste trabalho, foi escolhida a classificação de Casavant e Kuhl (1988) para definir os termos e conceitos relacionados ao escalonamento de processos. Essa classificação é dividida em duas partes: uma classificação hierárquica (figura 2.1), onde os termos se relacionam entre si; e uma classificação plana, onde não existe nenhuma relação hierárquica entre os termos. As duas classificações são discutidas a seguir.

2.4.1 Classificação hierárquica

Como foi exposto anteriormente, Casavant e Kuhl (1988) dividem o escalonamento em duas grandes classes: **local** (apenas um processador) e **global** (vários processadores).

O escalonamento global pode ser **estático** ou **dinâmico**. No estático, o escalonamento é feito antes da execução do processo e é definido explicitamente no código fonte ou durante a compilação do programa a ser escalonado, não sendo modificado até o final de sua execução. O escalonamento **dinâmico** é efetuado em tempo de execução, e pode ser modificado a qualquer momento enquanto o processo estiver sendo executado.

Os algoritmos de escalonamento estático requerem que um grande número de informações sobre o processo a ser executado esteja disponível antes de sua execução (por exemplo, tempo de execução do processo, localização dos dados a serem manipulados, etc.), o que raramente acontece, visto que a maioria dos problemas resolvidos computacionalmente não são determinísticos (Baumgartner & Wah, 1991). Se a estimativa da necessidade de recursos computacionais não corresponder à realidade, um escalonamento estático fatalmente ocasionará a degradação do desempenho (Shirazi & Hurson, 1992). Silva (1997) apresenta, em seu trabalho, uma discussão sobre alguns aspectos relacionados ao escalonamento estático.

Figura 2.1 -- Taxonomia Hierárquica de Casavant e Kuhl (1988)

O escalonamento dinâmico possibilita uma maior flexibilidade através da utilização de informações sobre o estado do sistema computacional (índices de carga, por exemplo) para auxiliar nas suas decisões de escalonamento, possibilitando um melhor desempenho. Também é possível modificar a distribuição dos processos (através de migração de processos), permitindo a adequação do sistema em caso de uma mudança brusca das cargas nos processadores. Uma desvantagem dessa abordagem é a sobrecarga imposta no sistema pelo próprio algoritmo de escalonamento e pela necessidade de possíveis migrações de processos (Shirazi et al., 1995(a)).

A responsabilidade pelo escalonamento pode ser centralizada em um processador (**não distribuído**) ou dividida entre vários processadores (**distribuído**). Se distribuído, o escalonamento pode ser organizado de maneira **cooperativa**, quando as decisões de escalonamento em cada processador são tomadas levando em consideração uma visão global do sistema, ou de maneira **não cooperativa**, quando cada processador decide baseado apenas em informações locais.

Alguns termos, que não possuem uma relação hierárquica entre si, mas podem ser usados para classificar um algoritmo de escalonamento, são definidos por Casavant e Kuhl (1988) em uma classificação plana, a qual é discutida a seguir.

2.4.2 Classificação Plana

Um algoritmo de escalonamento pode ser **adaptativo** ou não, no sentido de que é possível adaptar o seu próprio comportamento de acordo com a situação atual do sistema (Shirazi & Hurson, 1992; Shao, 2001). Enquanto que, em um escalonamento dinâmico, são utilizadas informações sobre a carga do sistema sempre de acordo com parâmetros e regras fixas, um algoritmo de escalonamento adaptativo pode modificar inclusive esses parâmetros e regras, de acordo com o desempenho obtido de sua utilização (Shivaratri et al., 1992).

Um algoritmo de escalonamento de processos dinâmico pode tomar uma decisão de escalonamento, e não modificá-la mais até o final da execução do processo. Tem-se então um caso que Casavant e Kuhl (1988) definem como **atribuição inicial**, e esse enfoque se contrapõe à **reatribuição dinâmica**, onde um processo pode ser transportado de um processador a outro através de uma migração. Naturalmente, uma migração de processos, principalmente em uma rede de computadores, pode ser uma atividade bastante onerosa (pelo tempo de salvamento de contexto e transporte do processo pela rede). Migração de processos é um tema muito estudado por trabalhos acadêmicos relacionados ao escalonamento dinâmico (Harchol-Balter & Downey, 1997; Douglass & Ousterhout, 1991; Milojicic et al., 1993; Stellner & Trinitis, 1997; Song et al., 1997; Wibisono, 2002).

Um uso muito comum da reatribuição dinâmica, e que em muitos trabalhos é confundido com a própria atividade de escalonamento, é o **balanceamento de carga**.

Shivaratri et al. (1992) discute, de maneira mais genérica, a atividade de distribuição de carga (*load distributing*), que possui duas possíveis variantes: compartilhamento de carga (*load sharing*) e balanceamento de carga (*load balancing*). Compartilhar carga significa evitar estados onde processadores estejam sobrecarregados enquanto outros estejam disponíveis. Indo mais além, se esse processo de compartilhamento se preocupa em manter as cargas homogêneas no sistema computacional, então se tem o balanceamento de carga.

A classificação de Casavant e Kuhl é bastante completa, mas não inclui alguns termos que são encontrados em trabalhos de escalonamento de processos. Alguns exemplos de termos encontrados em artigos científicos relacionados ao escalonamento de processos em computadores paralelos e não contemplados por Casavant e Kuhl, são: compartilhamento de espaço (*space sharing*), compartilhamento de tempo (*time sharing*) e escalonamento em grupo (*gang scheduling*).

No primeiro, os processadores disponíveis são organizados em grupos, e cada aplicação paralela tem acesso exclusivo a um grupo de processadores. No segundo, todos os processadores são compartilhados entre todos os processos, em enfoque semelhante ao escalonamento local utilizando *time sharing* (Feitelson & Rudolph, 1995; Setia et al., 1998; Squillante, 1995).

Escalonamento em grupo se refere ao ato de escalonar em conjunto (no mesmo instante de tempo) processos que se comunicam entre si, evitando que processos em execução se comuniquem com processos que não tenham sido escalonados, garantindo uma melhor sincronização entre eles (Feitelson & Rudolph, 1995; Setia et al., 1999).

Nesta subseção, foram discutidos alguns termos e definições relacionadas ao escalonamento de processos utilizando como ponto de referência algumas taxonomias encontradas na literatura científica de computação. As próximas seções discutem questões mais específicas relacionadas ao escalonamento de processos, iniciando pela discussão da organização de um algoritmo de escalonamento.

2.5 Organização de um Algoritmo de Escalonamento

Como foi exposto na seção 2.3, um algoritmo de escalonamento pode ser organizado como um conjunto de políticas, permitindo uma visão mais modular das

diferentes atividades desempenhadas por ele. O número e função dessas políticas variam de acordo com a visão e o enfoque que se dá à área de escalonamento de processos (Souza et al., 2000). Por exemplo, algoritmos de escalonamento específicos para a computação paralela tendem a ser organizados de maneira diferente do que algoritmos para sistemas distribuídos.

Um enfoque bastante interessante é apresentado por Shivaratri et al. (1992), e é aplicado a algoritmos de escalonamento dinâmico, que são baseados na distribuição da carga entre os diversos processadores utilizando migração de processos. Segundo Shivaratri et al. (1992), um algoritmo de escalonamento é composto por quatro políticas, que são:

- Política de transferência: determina a disponibilidade de um processador de participar de uma transferência de processos, seja como transmissor ou como receptor;
- Política de seleção: definido que um processador será um transmissor, a política de seleção será a responsável por decidir qual processo será transferido;
- Política de localização: responsável por definir um processador parceiro para participar da transferência;
- Política de informação: a política de informação é responsável por definir em que momento informações sobre o estado do sistema são necessárias, de que processadores elas devem ser coletadas, e que tipo de informação deve ser utilizada.

Uma informação muito importante, e que deve ser medida de maneira bastante criteriosa, é a carga de um processador, que permite avaliar o quanto um processador está carregado ou não. Geralmente, a carga é medida utilizando **índices de carga**, que são valores inteiros, iniciados em zero (quando o sistema está desocupado) e que aumentam de acordo com o aumento da carga computacional de um processador (Branco, 2000; Ferrari & Zhou, 1987; Kunz, 1991; Merah & Wah, 1993; Shirazi et al. 1995(b)).

Exemplos de índices de carga são: índices baseados no tamanho da fila de acesso a um recurso (por exemplo, fila de processos da CPU e fila de acesso à memória); índices baseados no percentual de utilização de um recurso (por exemplo, percentual de ocupação

da CPU); índices baseados no tempo de execução/resposta (por exemplo, tempo de resposta dos processos e tempo de CPU ociosa) (Ferrari & Zhou, 1987; Souza et al., 2000).

Embora a definição de índice de carga pareça simples, a obtenção e, principalmente, a interpretação de um índice de carga não é uma tarefa trivial. Por exemplo, considere um processador que esteja com uma grande utilização da CPU, mas que esteja utilizando muito pouco o disco rígido. Para processos *CPU-bound*, o processador poderá ser considerado muito carregado e, portanto, inviável, enquanto que, para processos *disk-bound*, o processador poderá ser considerado uma boa opção de escalonamento.

Dessa forma, índices de carga simples muitas vezes não conseguem definir a carga de um processador de maneira satisfatória, sendo necessária a utilização de índices combinados para refletir com consistência a situação atual do sistema. A obtenção de tais índices combinados incorre em maior sobrecarga no sistema, e pode, se os índices não forem bem escolhidos, fornecer resultados imprecisos para aplicações que possuam requisitos bem definidos em termos de recursos (por exemplo, o mais importante para uma aplicação *CPU-bound* é saber o quanto a CPU está ocupada).

Em sistemas distribuídos, que geralmente possuem grande heterogeneidade, índices de carga absolutos podem levar a contradições, pois um processador de maior poder computacional às vezes pode ser uma opção mais viável, mesmo que em termos absolutos possua uma carga maior. Por isso, costuma-se normalizar os índices de carga, levando em consideração a potência computacional dos processadores envolvidos.

Discussões sobre índices de carga são encontradas em diversos trabalhos relacionados ao escalonamento de processos (Araújo et al., 1999; Clark et al., 1995; Dantas & Zaluska, 1998; Santana & Zaluska, 1988; Schnor et al., 1996; Zhou et al., 1993).

A escolha de um ou mais índices de carga, e também a função de cada uma das políticas de um algoritmo de escalonamento são definidas levando em consideração o objetivo ao qual o escalonamento se propõe. A próxima seção apresenta uma discussão sobre os possíveis objetivos relacionados a um escalonamento de processos, e o desempenho que se pode obter baseado nesses objetivos.

2.6 Objetivos e Desempenho de um Escalonamento de Processos.

Uma maneira de avaliar o desempenho de um algoritmo de escalonamento é avaliar o quanto esse algoritmo é adequado ao objetivo ao qual se propõe (Krallmann & Schwigelshohn, 1999). Existem diversos objetivos que podem estar atrelados a um escalonamento, que podem ser coletivos (quando são definidos baseados em uma visão global do sistema, aumentando a produtividade do sistema como um todo) ou individuais (quando são definidos focando-se a satisfação dos usuários). Exemplos de objetivos, segundo Feitelson e Rudolph (1996) são:

Otimizar a utilização de um recurso (Balanceamento de carga): Caracteriza-se pela utilização ótima dos recursos computacionais disponíveis (CPU, memória, etc.), ou seja, deve-se evitar que alguns recursos fiquem sobrecarregados enquanto outros estão livres.

Maximizar a utilização de um recurso: Procura-se manter a maior parte dos recursos ocupados. É um objetivo comum em computadores paralelos com vários processadores, onde se procura escalonar os processos de maneira que todos os processadores sejam utilizados.

Maximizar o *throughput*: Caracteriza-se por aumentar o número de aplicações completadas por unidade de tempo. Aumentando o *throughput*, na teoria, aumenta-se o número de usuários satisfeitos.

Reduzir o tempo médio de resposta: Por tempo de resposta, entenda-se o tempo decorrido entre o momento em que se submete o processo ao sistema computacional até o término dele.

Outros possíveis objetivos que podem ser citados são: execução de processos em um tempo pré-determinado (tempo real), tolerância a falhas, etc. (Souza et al., 2000). Alguns objetivos são ainda relacionados a questões administrativas, como por exemplo, priorizar determinados processos em detrimento de outros, priorizar a utilização de um

determinado processador para um projeto específico, entre outros (Feitelson & Rudolph, 1996; Ferstl, 1996).

Muitos dos objetivos citados são contrastantes. Por exemplo, geralmente, quando o objetivo do escalonamento é maximizar a utilização da CPU, existe uma tendência a se escalonar processos de maior duração, aumentando o tempo de resposta. (Feitelson & Rudolph, 1995, 1998).

A escolha de um objetivo adequado depende de diversos fatores, como por exemplo: o tipo de carga de trabalho dos processadores, as características do hardware disponível no sistema computacional, os interesses administrativos, entre outras. Por exemplo, um sistema computacional que execute predominantemente aplicações que usem bastante memória provavelmente terá objetivos que sejam relacionados à memória. Também, determinados objetivos são comuns em sistemas distribuídos (por exemplo, balanceamento de carga) e outros são comuns em computadores paralelos (por exemplo, maximização do uso dos processadores).

A definição de desempenho do escalonamento considerada neste trabalho é formalizada de acordo com o objetivo ao qual o algoritmo de escalonamento se propõe. Por isso, define-se que o escalonamento está sendo feito de forma eficiente (com desempenho) quando o escalonamento está alcançando os objetivos estipulados.

Através da definição de objetivos, podem ser definidas métricas de desempenho que permitem avaliar, em termos numéricos, a qualidade do escalonamento. Essas métricas têm como função formalizar os objetivos, de maneira que se possa avaliar o desempenho do escalonamento utilizando técnicas de avaliação de desempenho clássicas como simulação, *benchmarks* e monitoração.

Na próxima subseção, são discutidos exemplos de algumas métricas de desempenho encontradas comumente na literatura científica de computação.

2.6.1 Métricas de desempenho

Como foi exposto anteriormente, um escalonamento de processos apresenta bom desempenho quando alcança seu objetivo. Para medir o quanto um determinado objetivo foi alcançado, são utilizadas métricas de desempenho. Dentro desse contexto, é muito

importante que as métricas sejam consistentes e conclusivas, para que se possa fazer uma avaliação coerente.

Essa é uma área pouco explorada nos trabalhos relacionados ao escalonamento de processos (Feitelson & Rudolph, 1998), e a maioria dos trabalhos científicos envolvendo avaliação de algoritmos de escalonamento utiliza um conjunto padrão de métricas de desempenho sem fazer uma análise crítica sobre a sua adequação e validade.

Alguns exemplos de trabalhos que citam métricas de desempenho são: (Cheng & Dandamudi, 1992; Clark et al., 1995; Feitelson, 2001; Feitelson, 2003; Gibbons, 1997; Harchol-Balter & Downey, 1997; Krone et al., 1998; Milojicic et al., 1993; Naik et al., 1993; Schnor et al., 1996; Schnor & Petri, 1999; Shao et al., 1998; Silva & Scherson, 2000; Zhang et al., 2000).

Alguns exemplos de métricas de desempenho são:

Throughput: Mede o número de processos terminados por unidade de tempo;

Percentual de utilização de recursos de um sistema computacional: Mede a percentagem de utilização de um determinado recurso (CPU, memória, etc.);

Tempo médio de resposta: Tempo decorrido entre o início e o término de um processo;

Makespan: Mede o tempo total de execução de um conjunto de processos;

Slowdown: Tempo de execução do processo em um sistema carregado dividido pelo seu tempo de execução em um sistema livre. Aplicações com valor de *slowdown* igual a 1 têm bom desempenho, e valores maiores que 1 significam degradação do desempenho (Feitelson & Rudolph, 1998).

Um problema da métrica *slowdown* é o fato de que aplicações de duração mais curta tendem a ter valores de *slowdown* muito altos. Por exemplo, uma aplicação de 1 segundo de duração, que demore 20 minutos para se iniciar, terá um valor de *slowdown* de 1200. Para evitar essa discrepância, geralmente se define um limite mínimo de duração das aplicações, de maneira que toda a aplicação que durar menos que esse limite terá seu tempo de duração reajustado. Para um limite de 20 segundos, a mesma aplicação de 1 segundo

terá valor de *slowdown* igual a 60. Essa métrica é chamada de *slowdown limitado* (*bounded slowdown*).

O capítulo 6 apresenta uma discussão mais detalhada sobre métricas de desempenho e sua adequação para avaliar um escalonamento de processos.

Como foi exposto nesta subseção, um escalonamento de processos pode ter diversos objetivos e pode ser avaliado por diferentes métricas de desempenho. Considerando que existem diferentes tipos de sistemas computacionais, diferentes tipos de carga de trabalho e diferentes objetivos, percebe-se que é muito difícil implementar um algoritmo de escalonamento que seja eficiente em todas as situações possíveis. Por isso, há a necessidade de que os softwares de escalonamento de processos disponíveis sejam cada vez mais flexíveis e dinâmicos, permitindo que vários algoritmos de escalonamento possam ser utilizados.

Na próxima seção, são discutidos alguns estudos de casos de softwares que fazem o escalonamento de processos em sistemas distribuídos, de maneira que se tenha uma visão geral de como o escalonamento de processos é organizado e implementado na prática.

2.7 Softwares de Escalonamento

Segundo Feitelson e Rudolph (1995), existem diversas maneiras pelas quais o escalonamento de processos pode ser implementado em um determinado sistema computacional. Em um primeiro nível, está o escalonamento que é feito pelo próprio usuário, seja ele definido explicitamente no código fonte, através de um grafo direcionado (onde os nós são processos e os arcos que ligam esses nós são padrões de comunicação) (Silva, 1997), ou através de um compilador que gere um escalonamento automático.

Quando se quer fornecer um escalonamento mais genérico, que contemple vários usuários e não apenas as necessidades de um único usuário, o escalonamento é feito pelo próprio sistema operacional ou por algum software que execute sobre o sistema operacional e que desempenhe a função de escalonamento, funcionando como um *middleware*. Esse último tipo de escalonador é comumente chamado de **sistema de gerenciamento de jobs e recursos** (Ferstl, 1996; Baker & Buyya, 1999). Neste trabalho, é utilizado o termo software

de escalonamento para representar todo software que seja responsável por fazer o escalonamento de processos em um sistema distribuído.

Existem diversos exemplos de softwares de escalonamento de processos, e em muitos casos são projetados para se adequarem ao sistema computacional sobre o qual irão executar, fazendo parte do próprio sistema operacional. Essa abordagem é comum a muitos sistemas distribuídos e computadores paralelos (Tanenbaum, 1995), e tem a vantagem de ser implementada de maneira otimizada para o sistema computacional onde executará (dificultando a portabilidade) e também de poder obter índices de carga bastante precisos, pois pode-se acessar os recursos do sistema operacional diretamente.

Ferstl (1996) afirma que o escalonamento feito fora do sistema operacional, apresenta algumas vantagens em relação ao escalonamento feito pelo sistema operacional:

- Permite ao administrador o controle e planejamento da utilização dos recursos de uma maneira estruturada;
- São facilmente adaptáveis a sistemas computacionais heterogêneos;
- Possibilita, na teoria, uma interface independente do sistema operacional ou do fabricante, permitindo a implementação de softwares de escalonamento portáteis.

Existem diversas implementações de softwares de escalonamento, comerciais e de domínio público, que são relacionadas e descritas em trabalhos como os de Baker et al. (1996), Ferstl (1996), Jones (1996) e Kaplan e Nelson (1994). Segundo esses mesmos autores, pode-se citar um conjunto de características desejáveis que um software desse tipo deve oferecer, e que podem ser estendidas para software de escalonamento de processos para sistemas distribuídos em geral. Entre elas, destacam-se:

- Suporte à heterogeneidade, de maneira que possa controlar sistemas computacionais compostos por diferentes arquiteturas de computadores (heterogeneidade arquitetural);
- Suporte a aplicações *batch*, interativas e paralelas;

- Suporte a diferentes abordagens de escalonamento, que podem ser substituídas de acordo com as necessidades do usuário;
- Suporte a migração de processos;
- Suporte ao balanceamento de carga;
- Presença de ferramentas para a monitoração dos recursos do sistema e também dos processos sendo executados no sistema.

A tabela 2.1 apresenta um conjunto dos principais softwares de escalonamento encontrados na literatura (Baker et al., 1996; Ferstl, 1996), relacionando a eles as características citadas acima. Os softwares considerados são: *Sun Grid Engine* (Sun Microsystems Inc., 2002), *Condor* (Condor Team, 2004), *AMIGO* (Souza, 2000(a); Souza et al., 1999(a), 1999(b), 1999(c)), *LSF* (Platform Computing Corporation, 2000), *NQE* (Cray Research Inc., 2000) e *PBS* (Veridian Systems, 2000).

	<i>Sun Grid Engine</i>	<i>Condor</i>	<i>AMIGO</i>	<i>LSF</i>	<i>NQE</i>	<i>PBS</i>
Heterogeneidade	*	*	*	*	*	*
Batch	*	*	*	*	*	*
Paralelas	*	*	*	*	*	*
Interativas	*		*	*	*	*
Diversos alg. de escalonamento	*		*	*	*	*
Migração de Processos	*	*		*		
Balanceamento de Carga	*		*	*	*	*
Monitoração	*	*	*	*	*	*
Gratuito	*	*	*			*

Tabela 2.1 – Relação de Softwares de Escalonamento

A próxima subseção apresenta uma descrição da estrutura dos softwares de escalonamento citados, enfatizando as características apresentadas na tabela 2.1.

2.7.1 Estrutura dos Softwares de Escalonamento.

Uma abordagem comum para automatizar a tarefa de escalonar processos é a definição de *scripts* onde o usuário define a necessidade de recursos de sua aplicação. Os processadores disponíveis são acessados através de filas, e essas filas são definidas de acordo com os recursos que são oferecidos pelo processador, sendo que um mesmo processador pode ter diferentes perfis definidos por diferentes filas. O software de escalonamento então escolhe uma determinada fila de acordo com o *script* definido pelo usuário.

Além de verificar a necessidade de recursos dos processos a serem escalonados, os softwares de escalonamento também procuram manter a carga do sistema balanceada, utilizando variados índices de carga. Os algoritmos de escalonamento utilizados são flexíveis e podem ser modificados (ou trocados) para se adaptarem às diferentes necessidades de um sistema computacional. Seguem essa abordagem *Sun Grid Engine*, *LSF*, *NQE* e *PBS*. Adicionalmente, *Sun Grid Engine* e *LSF* oferecem suporte à migração de processos, permitindo que o balanceamento de carga seja reajustado através de reatribuição dinâmica.

No sistema Condor, um processo também é escalonado segundo a sua necessidade de recursos, mas através de um mecanismo definido como *ClassAd* (anúncios classificados). Através desse mecanismo, as estações de trabalho anunciam a quantidade de recursos que possuem (quantidade de memória, capacidade da CPU, etc.) e esses anúncios são combinados com a necessidade de recursos dos processos a serem escalonados, permitindo que os processos sejam executados nas estações que sejam mais apropriadas a eles.

Processos são então escalonados para estações de trabalho disponíveis, priorizando sempre o usuário que seja proprietário da estação. Dessa maneira, se o proprietário inicia a utilização de sua estação de trabalho, os processos que lá estão executando são migrados para outra estação que esteja disponível. Nesse caso não existe balanceamento de carga, visto que o objetivo do software de escalonamento é evitar que existam estações de trabalho livres mantendo o respeito ao usuário que comumente usa essas estações.

O *AMIGO*, desenvolvido no grupo de Sistemas Distribuídos e Programação Concorrente do ICMC/USP, possui uma filosofia relativamente diferente dos softwares de escalonamento citados acima. Ele foi desenvolvido com o intuito de fornecer uma maneira flexível e dinâmica de agrupar algoritmos de escalonamento e relacioná-los com as aplicações dos usuários.

Através da interface gráfica do *AMIGO*, é possível fazer o cadastro de classes de aplicação (algo como um perfil que caracteriza um tipo de aplicação), e também de algoritmos de escalonamento. O *AMIGO*, no caso, faz o elo de ligação entre as classes de aplicações e os algoritmos de escalonamento.

O *AMIGO* e todos os outros softwares de escalonamento possuem suporte à heterogeneidade, sendo que todos são projetados para executar em diferentes arquiteturas computacionais. Aplicações *batch* e paralelas, que representam grande parte das aplicações científicas e que são os alvos primários de um software de escalonamento, também são suportadas pelos softwares de escalonamento.

Condor suporta apenas aplicações *batch* e paralelas, visto que considera que aplicações interativas geralmente são executadas no próprio processador em que foram iniciadas. Esse não é o enfoque dos demais softwares de escalonamento, que incluem suporte explícito a aplicações interativas.

Concluindo, todos os softwares de escalonamento oferecem algum tipo de monitoração, com diversos objetivos. A relação entre a monitoração e os softwares de escalonamento é discutida em detalhes no capítulo 4.

2.8 Considerações Finais

Nos últimos anos, a computação sofreu grandes transformações devido ao progresso tecnológico, permitindo que sistemas distribuídos e computadores paralelos se transformassem em excelentes alternativas para a organização do processo computacional. Atualmente tem havido uma convergência entre as duas áreas de pesquisa, em virtude da existência de características comuns em ambos tipos de sistemas computacionais.

Sistemas distribuídos e computadores paralelos caracterizam-se pela existência de diversos processadores conectados através de uma rede de comunicação, e o acesso a esses

processadores tem de ser feito de maneira organizada, para que se possa obter resultados satisfatórios de sua utilização.

O escalonamento de processos define as regras de distribuição dos processos aos processadores, e um escalonamento feito de maneira eficiente, ou seja, que alcança os objetivos a que se propõe, contribui para que o sistema computacional seja utilizado de maneira eficiente.

Existem diversos fatores que podem influenciar o desempenho de um escalonamento de processos e diversos objetivos possíveis, tornando muito difícil a existência de um algoritmo de escalonamento que seja bom para toda e qualquer situação. Por isso, é muito importante que sejam utilizadas técnicas diversas de avaliação de desempenho aplicadas ao escalonamento de processos, garantindo melhores algoritmos de escalonamento e, por conseguinte, sistemas computacionais mais eficientes.

Para que isso seja alcançado, é importante que algoritmos de escalonamento sejam avaliados durante o seu desenvolvimento e também durante a sua utilização em sistemas computacionais. Dentro desse contexto, este trabalho discute um exemplo de avaliação de desempenho para o escalonamento de processos utilizando monitoração.

O próximo capítulo apresenta uma descrição sobre técnicas de avaliação de desempenho, com ênfase na monitoração, apresentando exemplos onde essas técnicas podem ser empregadas na avaliação do escalonamento de processos.

3 Avaliação de Desempenho Para o Escalonamento de Processos

Este capítulo descreve algumas técnicas de avaliação de desempenho bastante difundidas (de modelagem e de aferição), mostrando exemplos de aplicação dessas técnicas no escalonamento de processos. A técnica de monitoração possui uma seção própria, mostrando seus principais conceitos e sua aplicação em sistemas distribuídos.

3.1 Considerações Iniciais

Toda e qualquer atividade envolvida no processo computacional pode e deve ser medida e avaliada, a fim de que se possa certificar se ela é adequada à aplicação para a qual foi projetada e se corresponde às expectativas em termos de eficiência e confiabilidade (Kant, 1992).

Avaliar o desempenho não é uma tarefa trivial. Jain (1991), por exemplo, inicia seu livro sobre o assunto apresentando os erros mais comuns (vinte e dois no total) que são cometidos quando se tenta fazer uma avaliação de desempenho. Uma avaliação feita sem critérios vai ter o efeito contrário ao esperado, ou seja, vai levar a resultados errados que podem inclusive ocasionar modificações desnecessárias em um sistema que esteja funcionando bem.

Avaliações de desempenho são baseadas na análise de **métricas de desempenho**, que são responsáveis por formalizar e quantificar os objetivos da avaliação. Ou seja, a métrica de desempenho vai fornecer um valor que permitirá afirmar se o desempenho está bom ou ruim, de acordo com os objetivos preestabelecidos.

O escalonamento de processos, como qualquer outra atividade computacional, pode e deve ser avaliado para que seu comportamento possa ser aprimorado. Técnicas de avaliação de desempenho podem ser aplicadas em diversas fases do desenvolvimento e no uso cotidiano de um algoritmo de escalonamento.

Existem basicamente dois grandes grupos de técnicas de avaliação de desempenho, com diferentes aplicações e características, que são: **técnicas de aferição** e **técnicas de**

modelagem (Orlandi, 1995; Francês, 1998). Nas próximas seções, cada uma das técnicas é brevemente discutida, mostrando como elas podem ser utilizadas para avaliar a atividade de escalonamento de processos.

3.2 Técnicas de Modelagem

Em muitos casos, há a necessidade de fazer determinadas experimentações em um sistema computacional sem influir no seu comportamento, ou seja, não é viável nem desejável experimentar no próprio sistema. Uma solução interessante nesse contexto é construir um **modelo do sistema**, ou seja, representá-lo através de alguma técnica de representação formal, de maneira que se possa fazer experimentações no modelo, e extrapolar os resultados para o sistema real.

Um modelo não precisa representar o sistema completo, mas sim apenas as características relevantes ao que se quer avaliar. A modelagem também pode ser aplicada para avaliar um sistema não existente, de maneira que se possa experimentá-lo e aperfeiçoá-lo em uma fase anterior à implementação, o que é bastante interessante em relação ao custo de se desenvolver esse sistema.

Diversas técnicas para a modelagem de sistemas computacionais são descritas na literatura, com diferentes características e diferentes aplicações (Orlandi, 1995; Francês, 1998). Redes de filas, Redes de Petri e *Statecharts* são as mais utilizadas. Descrições detalhadas dessas técnicas são encontradas em (Soares, 1992; MacDougall, 1987; Harel, 1987).

A partir do momento que se modelou o sistema, há a necessidade de resolvê-lo, para que se obtenham resultados que possam demonstrar o desempenho do sistema modelado. Dessa forma, através da solução do modelo, será possível obter as métricas de desempenho.

Um modelo pode ser resolvido por **simulação** ou **analiticamente**. Simular um modelo é construir um programa computacional que represente esse modelo, e através de sua execução pode-se obter as métricas de desempenho. Técnicas analíticas fornecem uma solução para o modelo através da utilização de equações matemáticas que o representem. Resolvendo essas equações, obtêm-se as métricas de desempenho. Cada um dos dois

enfoques tem suas vantagens e desvantagens, sendo que a indicação do mais adequado depende do sistema a ser modelado, do objetivo da avaliação, da quantidade de parâmetros disponíveis, etc. A figura 3.1 (Orlandi, 1995) apresenta um esquema simplificado de quando deve ser utilizado cada um dos enfoques.

Figura 3.1 - Soluções de Um Modelo

Um modelo possui diversos parâmetros (componentes que podem possuir diferentes valores com o decorrer do tempo), que representam os diferentes estados do modelo. Esses parâmetros têm que ser instanciados com valores obtidos no (ou deduzidos a partir do) sistema real. Essa instanciação é chamada de **parametrização**, e é de vital importância para a correta solução do modelo (Orlandi, 1995; Francês, 1998).

Técnicas de modelagem podem ser aplicadas na fase de desenvolvimento de algoritmos de escalonamento, para que os mesmos possam ser avaliados antes mesmo de serem implementados (Feitelson & Rudolph, 1995, 1998). Também, modificações e aprimoramentos podem ser feitos nos algoritmos já implementados sem influenciar no comportamento do sistema, sendo feitos apenas após vários testes. Um problema a ser considerado é o da parametrização do modelo, que geralmente é feita através da caracterização da carga de trabalho, que não é uma tarefa trivial, principalmente em

ambientes com grande variação da carga computacional, como por exemplo, redes de computadores utilizadas por vários usuários.

O trabalho de Souza (2000(b)) demonstra que é possível e viável aplicar técnicas de modelagem para a avaliação e aprimoramento de algoritmos de escalonamento. Outros trabalhos que também empregam técnicas de modelagem para avaliar o escalonamento são: (Buyya & Murshed, 2002; Hlavacs & Ueberhuber, 1999; Schnor et al., 1996; Xiao et al., 2003; Zhang et al., 2000).

3.3 Técnicas de Aferição

Em algumas situações existe a necessidade de coletar dados diretamente em um sistema computacional pronto, ou que se encontra em fase de desenvolvimento. Nesse caso, a avaliação de desempenho pode ser feita através dos dados obtidos diretamente no sistema, e a partir desses dados são calculadas as métricas de desempenho.

Técnicas de aferição apresentam o potencial de obter dados mais precisos do que os obtidos pela solução de modelos. No entanto, têm a desvantagem de atuarem diretamente no sistema, muitas vezes disputando recursos com as aplicações sendo avaliadas, podendo influenciar no desempenho das mesmas, afetando os próprios resultados a serem aferidos.

Dentre as técnicas de aferição, três se destacam (Orlandi, 1995; Francês, 1998): protótipos, *benchmarks* e monitoração. A terceira técnica será estudada em uma seção à parte, em virtude de sua importância neste trabalho.

Protótipos são implementações simplificadas de um determinado software computacional, procurando manter as funcionalidades do mesmo, que implementam apenas as características essenciais do software, ou seja, aquelas que se quer avaliar. Protótipos podem ser utilizados na implementação de algoritmos de escalonamento, mas o custo de implementação desses protótipos geralmente acaba tornando essa uma técnica não muito utilizada (Feitelson & Rudolph, 1998).

Protótipos não são utilizados para avaliação de sistemas já existentes, e por isso não são adequados para a avaliação proposta neste trabalho de doutorado.

Outra técnica de aferição muito usada, os *benchmarks* (Hockney, 1996) são programas utilizados como padrões para avaliação de desempenho, ou seja, através da execução desses programas é possível obter métricas de desempenho que permitem avaliar diferentes componentes de um sistema computacional. *Benchmarks* são bastante usados para fins de comparação, pois através de sua execução em diferentes sistemas computacionais, é possível obter métricas que possam ser utilizadas para compará-los. Não existem métricas genéricas, e geralmente *benchmarks* avaliam características particulares de um sistema computacional.

Um *benchmark*, através de sua execução, impõe uma **carga de trabalho** a um sistema computacional, e a maneira que esse sistema reage a essa carga é que caracteriza se o seu desempenho é bom ou não. Um escalonamento de processos, dentro desse contexto, nada mais é que um gerenciador de cargas de trabalho, que deve garantir (idealmente) que o desempenho do sistema seja bom quando submetido a essas cargas de trabalho. E nesse aspecto, é passível de avaliação e comparação por *benchmarks*.

Para comparar e avaliar algoritmos de escalonamento, o que se faz geralmente é criar cargas de trabalho sintéticas, que reflitam o comportamento do sistema no dia-a-dia, e aplicá-las aos algoritmos de escalonamento. Através disso, é possível comparar o comportamento de diferentes algoritmos e concluir o que melhor desempenho oferece (Feitelson & Rudolph, 1998).

Enquanto um *benchmark* padrão executa “aplicações representativas” geralmente isoladas, um *benchmark* para algoritmos de escalonamento consiste de um modelo de chegada de aplicações ao sistema e uma caracterização da estrutura interna dessas aplicações. Através da modificação das aplicações, é possível testar diferentes funcionalidades de um algoritmo de escalonamento, como por exemplo, a resposta a uma carga predominantemente *memory-bound*.

O problema não está só na especificação dessa carga de trabalho, mas também na definição de métricas de desempenho que sejam significativas e que forneçam um resultado objetivo que permita comparar os diferentes algoritmos de escalonamento. Feitelson e Rudolph (1998) e Chapin et al. (1999) apresentam uma discussão bastante aprofundada sobre a utilização de *benchmarks* para avaliar o escalonamento de processos.

Finalmente, a terceira técnica de aferição é a monitoração (ou coleta de dados). A partir dela é possível avaliar o desempenho de todo e qualquer componente de um sistema computacional, coletando dados diretamente no próprio sistema em execução, o que permite potencialmente uma precisão maior do que todas as técnicas citadas acima. Além disso, a monitoração é um método bastante genérico, podendo ser utilizado nas mais diversas áreas de pesquisa da computação e nos mais variados tipos de problemas (Orlandi, 1995; Francês, 1998).

Essa generalidade permitida pela possibilidade de coletar dados no próprio sistema que se quer avaliar pode trazer consequências inesperadas, pois a própria monitoração pode influenciar no sistema em si, distorcendo os próprios dados coletados. Dessa forma, a monitoração deve ser organizada de maneira que a sobrecarga causada seja mínima, o que não é uma tarefa trivial.

Figura 3.2 – Relação entre Propostas de Avaliação e Técnicas de Aferição

A figura 3.2 traz uma relação entre as três técnicas de aferição discutidas nesta seção, relacionando-as às suas possíveis aplicações na avaliação de sistemas computacionais. É importante observar que essas técnicas podem ser utilizadas em conjunto, dependendo do objetivo e da forma de avaliação pretendida. A utilização dos

benchmarks, por exemplo, pode requerer um processo de monitoração que permita a obtenção dos resultados esperados.

Na próxima seção, é apresentada uma discussão mais detalhada sobre a monitoração em sistemas computacionais em geral, e no próximo capítulo são mostradas as várias aplicações possíveis dessa técnica de aferição dentro do escalonamento de processos, uma das quais é utilizada no trabalho apresentado nesta tese.

3.4 Monitoração em Sistemas Computacionais

A ferramenta utilizada para fazer a monitoração de um sistema computacional é comumente chamada de monitor. Suas possíveis funções são: observar o desempenho de um sistema, coletar estatísticas de desempenho, analisar os dados e mostrar os resultados. Alguns monitores também identificam problemas e sugerem possíveis soluções (Jain, 1991).

Existe um conjunto de termos que são utilizados com certa frequência em livros e artigos sobre monitoração. Os termos que são usados neste trabalho, de acordo com a definição de Jain (1991), são:

- Evento: uma mudança no estado do sistema, como por exemplo, uma operação de E/S ou a chegada de um pacote pela rede;
- *Trace*: histórico (*log*) de eventos, geralmente incluindo o tempo do evento e alguns parâmetros relacionados ao evento;
- Domínio: o conjunto de atividades observáveis por um monitor é definido como o domínio do monitor;
- Taxa de entrada: a frequência (observações por período de tempo) de eventos que um monitor pode observar corretamente;
- Resolução: período de tempo possível entre duas observações. Geralmente, a resolução do monitor depende da resolução do relógio (*clock*) do sistema onde executa. Por exemplo, se o relógio tem a resolução de milisegundos, o monitor também a terá.

Monitores têm sido utilizados para uma grande gama de objetivos, entre eles (Jain, 1991; Nutt, 1972):

- Aperfeiçoar um sistema, através do ajuste de alguns de seus parâmetros, de maneira a aumentar o desempenho;
- Medir a utilização dos recursos, permitindo descobrir gargalos de desempenho no sistema;
- Determinar as características de uma carga de trabalho (ou de um programa específico);
- Fornecer dados para modelar um sistema, possibilitando também fazer a sua parametrização e validação.

Existem diversas questões relacionadas à monitoração de um sistema computacional, que influenciam na escolha e na organização de um monitor. Por exemplo:

- A quantidade de dados necessária para se obter métricas de desempenho conclusivas, que varia desde a observação de todos os eventos relacionados ao problema que se quer analisar, até uma monitoração por amostragem, ou seja, coletando dados em intervalos de tempo fixos;
- Alguns problemas requerem que se monitore o software do sistema computacional, outros necessitam de monitoração do hardware, e alguns requerem que se monitore ambos;
- A taxa de entrada do monitor pode variar desde uma observação a cada milissegundo ou nanosegundo (nível microscópico) até uma observação a cada segundo, minuto ou hora (nível macroscópico).

Os tipos de monitores podem ser classificados de acordo com algumas características que apresentam. Uma primeira classificação é em relação ao mecanismo que ativa a monitoração (ou seja, o momento em que o dado é coletado), que pode ser por evento ou por tempo. Um monitor **baseado em eventos** é ativado quando acontece um determinado evento (um acesso a disco, por exemplo), e um monitor **baseado no tempo** (ou baseado em amostragem) é ativado em intervalos de tempo regulares. Monitores

baseados em eventos são úteis para observar eventos de frequência baixa, pois eventos de alta frequência podem gerar uma sobrecarga muito grande, sendo então preferível utilizar monitores baseados no tempo.

Os dados coletados por um monitor podem ser analisados e apresentados ao usuário no mesmo momento em que são observados, o que caracteriza um monitor *on-line*. Por outro lado, a análise pode ser feita por um programa separado depois de ser feita a coleta de todos os dados, o que necessariamente implica na existência de espaço de armazenamento suficiente para mantê-los. Nesse caso, tem-se um monitor *batch*.

Monitores também são classificados de acordo com a maneira que são implementados. Jain (1991) e Nutt (1972) definem quatro possíveis maneiras de implementação de um monitor, que são: por software, hardware, *firmware* e híbridos.

Monitores por software são normalmente usados para avaliar sistemas operacionais ou softwares que executam sobre esses sistemas. Em geral, esses monitores disputam os recursos (CPU, memória, etc.) com o sistema que estão avaliando, e por isso têm baixa taxa de entrada e maior sobrecarga. Sua resolução geralmente depende do relógio do sistema sendo avaliado.

Monitores por hardware caracterizam-se pela existência de um hardware que é acoplado ao sistema a ser avaliado através de interfaces que capturam sinais eletrônicos. Como o monitor está separado tanto logicamente quanto fisicamente do sistema sendo monitorado, causará uma sobrecarga praticamente nula. Permitem altas taxas de entrada, mas têm a desvantagem de ter um custo de implementação geralmente elevado.

Comparando as duas abordagens, Kant (1992) diferencia os dois tipos de monitores em cinco tópicos. Inicialmente, o **tipo de informação** que ambos coletam estão em níveis diferentes, e em geral um monitor por hardware não tem acesso a informações relacionadas ao software executando no sistema, e vice-versa. Também, monitores por software possuem maior **flexibilidade**, sendo facilmente modificáveis e adaptáveis. Por outro lado, monitores por software possuem maior **sobrecarga**, podem influenciar nos dados sendo coletados (esse tipo de interferência é chamada de *artifact*), e possuem **taxa de entrada** maior. Jain (1991) acrescenta que um monitor por hardware possibilita fazer uma monitoração de

maior resolução do que a do relógio do sistema avaliado, pois pode possuir um relógio próprio.

Geralmente, baseado em suas diferenças bastante significativas, não há muita dificuldade de decidir onde e quando usar cada tipo de monitor. Uma abordagem bastante interessante, e que procura unir as vantagens de ambas abordagens de implementação, é o chamado **monitor híbrido**, que apresenta uma combinação entre software e hardware. Finalmente, um **monitor firmware** caracteriza-se por acrescentar instruções de monitoração no próprio micro-código de um processador.

Os conceitos expostos até aqui se referem a monitores para sistemas computacionais em geral. O trabalho de doutorado descrito nesta tese propõe um monitor que executa em um sistema distribuído. Por isso, na próxima subseção, são apresentados conceitos relacionados à monitoração em sistemas distribuídos.

3.4.1 Monitoração em Sistemas Distribuídos

Grande parte dos sistemas computacionais atuais é organizada como sistemas distribuídos geralmente heterogêneos (configuracionalmente e arquiteturalmente). Naturalmente, essa distribuição dos componentes do sistema torna a monitoração mais complexa do que em computadores isolados. Isso porque, além de todas as questões envolvidas na implementação de um monitor mostradas nesta seção anterior, existe a necessidade do próprio monitor também ser implementado de maneira distribuída.

Joyce et al. (1987) citam algumas razões que tornam a monitoração de sistemas distribuídos mais complexa. Por exemplo, técnicas de monitoração clássicas que são aplicadas a computadores isolados têm de ser ampliadas para se adequarem a sistemas distribuídos. Por exemplo, monitores baseados em tempo terão dificuldades para obter uma determinada informação em um instante de tempo qualquer predeterminado, pelo fato dos relógios dos processadores não estarem necessariamente sincronizados.

Sistemas distribuídos também têm um comportamento não determinístico, sendo que diversas execuções de uma aplicação distribuída podem produzir diferentes ordens de execução dos eventos avaliados, dificultando a sua análise. Adicionalmente, a sobrecarga

imposta pelo monitor pode ser diferenciada nos diferentes processadores, confundindo os resultados gerados.

Monitores por hardware, que fornecem uma solução para alguns dos problemas citados acima, têm de se adaptar à distribuição física dos processadores (às vezes distribuídos em diversas salas), e lidar com a heterogeneidade.

Segundo Hofmann et al. (1994), um monitor para um sistema distribuído deve idealmente se adaptar às características inerentes e desejáveis de um sistema distribuído, ou seja, devem ser flexíveis, capazes de aumentar de escala, transparentes, e preparados para lidar com a heterogeneidade. Naturalmente, por serem distribuídos, devem possuir um mecanismo eficiente de comunicação entre seus diversos componentes, e que não sobrecarregue a rede de comunicação.

Figura 3.3 - Visão em Camadas de um Monitor Para um Sistema Distribuído

Jain (1991) argumenta que um monitor para um sistema distribuído pode ser visto como um conjunto de camadas, semelhante ao modelo ISO/OSI¹ para redes de computadores, de maneira que as camadas inferiores forneçam serviços às camadas

¹ Modelo padrão para estruturar uma rede de computadores, desenvolvido pela ISO, que apresenta uma organização hierárquica em camadas.

superiores. A figura 3.3 apresenta a organização proposta por Jain (1991). A nomenclatura definida por Jain é utilizada no restante desta tese. A **camada de observação** é responsável pela coleta de dados propriamente dita (pode ser implementada por hardware, software ou hibridamente), sendo que cada componente monitorado geralmente tem um observador responsável.

O componente do monitor responsável pela coleta dos dados dos diversos observadores é chamado de coletor (**camada de coleta**), e um sistema de monitoração pode conter vários coletores trabalhando em conjunto.

A **camada de análise** é responsável pela análise dos dados obtidos dos coletores (como por exemplo, análises estatísticas sobre esses dados). Os dados analisados são apresentados de acordo com os padrões definidos na **camada de apresentação**, através de relatórios, gráficos, alarmes, etc. Compreender e tomar uma decisão baseada nos dados apresentados é tarefa da **camada de interpretação**, e pode ser feita por um ser humano, ou em monitores mais complexos, por algum software especialista.

Figura 3.4 - Componentes de um Monitor Para um Sistema Distribuído

Através da **camada de console**, pode-se configurar parâmetros relacionados ao monitor, e a **camada de gerenciamento** é responsável por modificar o sistema de acordo com os dados obtidos pela monitoração.

Um monitor pode possuir zero ou mais componentes em cada uma das camadas, todos eles relacionando-se entre si. Um exemplo dessa relação é apresentado na figura 3.4 (Jain, 1991).

Monitorar um escalonamento global de processos implica em monitorar um sistema distribuído ou um computador paralelo, e deve ser organizado de acordo com os preceitos citados nesta subseção. Exemplos da aplicação de monitores no escalonamento de processos são mostrados no próximo capítulo.

3.5 Considerações Finais

Existem diversas técnicas de avaliação de desempenho que podem ser utilizadas para avaliar o desempenho de um escalonamento de processos. Durante o desenvolvimento do algoritmo de escalonamento, podem ser usadas técnicas de modelagem (resolvendo os modelos por simulação ou técnicas analíticas), que permitem prever o comportamento do algoritmo. Pode-se ainda implementar um protótipo que represente as características essenciais de um algoritmo de escalonamento, obtendo-se dados preliminares de como será o comportamento do algoritmo implementado.

A modelagem pode ser usada também no aperfeiçoamento e correção de erros de algoritmos já prontos. Nesse caso, pode-se avaliar o comportamento do algoritmo de escalonamento quando diferentes fatores são modificados, tais como a carga de trabalho imposta ao sistema, plataformas em que o algoritmo é executado, etc.

Outra técnica de avaliação bastante difundida, os *benchmarks* podem ser utilizados para avaliar diferentes aspectos de um algoritmo de escalonamento, e em especial podem ser usados para comparar diferentes algoritmos.

A técnica de monitoração desempenha um papel fundamental em qualquer atividade de avaliação de desempenho e pode ser utilizada em conjunto com outras técnicas de avaliação. Por exemplo, na modelagem, uma das formas de se determinar os parâmetros do modelo é através da monitoração de um sistema igual ou similar ao modelado.

Este trabalho considera a monitoração como uma técnica de avaliação independente. No contexto de escalonamento de processos, a monitoração pode ser utilizada sob diversos enfoques. Por exemplo, é possível fornecer informações atualizadas sobre o estado de um processo submetido a um software de escalonamento monitorando a sua execução. Também, através de monitoração pode-se investigar a utilização de recursos por parte das aplicações escalonadas, a fim de possibilitar ajustes em algoritmos de escalonamento adaptativos. No trabalho descrito nesta tese, a monitoração é utilizada com o intuito de obter dados que permitam ao software de escalonamento verificar o desempenho do escalonamento de processos, ou seja, verificar se o algoritmo de escalonamento escolhido está conseguindo alcançar os objetivos definidos para o escalonamento.

No próximo capítulo são discutidas diversas aplicações da monitoração na atividade de escalonar processos, e são mostrados alguns exemplos de softwares de monitoração desenvolvidos com o intuito de auxiliar o escalonamento de processos. Também são discutidos aspectos iniciais relacionados à monitoração proposta por esta tese de doutorado.

4 Monitoração Aplicada ao Escalonamento de Processos

Este capítulo apresenta uma discussão das diversas aplicações da monitoração no escalonamento de processos. Inicialmente, discute-se o estado da arte da utilização de monitores no escalonamento de processos, enfocando a abordagem de monitoração proposta por este trabalho. Finalmente, são apresentados exemplos de softwares de monitoração criados exclusivamente para auxiliar o escalonamento de processos.

4.1 Considerações Iniciais

Monitores podem ser utilizados com diversos objetivos em uma grande gama de atividades computacionais. No escalonamento de processos, a monitoração também pode ser usada de diversas maneiras, de acordo com o que se pretende alcançar.

Os dados coletados por um monitor podem ser utilizados em diversas fases de um escalonamento de processos. Antes do escalonamento de uma aplicação, por exemplo, podem ser utilizados para prover o software de escalonamento com informações sobre a carga do sistema. Durante a execução da aplicação, monitores podem ser utilizados para avaliar a utilização de recursos por parte dessa aplicação, permitindo um escalonamento de melhor qualidade em uma próxima execução. Ao término da execução da aplicação, os dados coletados podem ser utilizados para avaliar o desempenho do algoritmo de escalonamento em si, possibilitando comparar com outros algoritmos de escalonamento utilizados para a execução dessa mesma aplicação.

O trabalho descrito nesta tese tem como objetivo utilizar a técnica de monitoração de uma maneira mais geral, avaliando a atividade de escalonamento como um todo, através da utilização de monitores que colem dados que permitam obter métricas de desempenho que sejam utilizadas para se concluir quanto à qualidade do escalonamento. As métricas utilizadas e a maneira de avaliá-las dependem do objetivo do escalonamento de processos.

Com o intuito de apresentar um panorama geral da relação entre monitoração e escalonamento de processos, nas próximas seções são apresentados alguns exemplos da utilização de monitores para auxiliar o escalonamento de processos, enfatizando a abordagem de monitoração proposta nesta tese de doutorado. Também são discutidos

exemplos de softwares de monitoração utilizados em conjunto com softwares de escalonamento.

4.2 Aplicações de Monitoração ao Escalonamento – Estado da Arte

Uma pesquisa na Internet ou em periódicos que contenham artigos que relacionem monitoração e escalonamento vai resultar em um número grande de documentos. Monitores podem ser utilizados de várias maneiras quando relacionados com o escalonamento de processos. A seguir são discutidas diferentes formas de utilizar monitores em conjunto com o escalonamento de processos, com o intuito de apresentar o estado da arte da área e definir qual o enfoque utilizado neste trabalho.

Basicamente, pode-se classificar a relação monitoração/escalonamento em três grandes classes, de acordo com o propósito do monitor:

I. Monitoração de processos: Grande parte dos softwares de escalonamento permitem que se monitore a execução de um processo, de maneira que o usuário possa acompanhar a execução do processo submetido por ele, e que o software de escalonamento possa reiniciar o processo caso ocorra algum problema, garantindo alguma tolerância a falhas. Esse tipo de funcionalidade é oferecido por quase todos os principais softwares de escalonamento, comerciais ou de domínio público, segundo atesta o trabalho de Baker (1996). Esse tipo de monitoração é bastante específico, relacionado aos processos, e não tem relação com a monitoração abordada por este trabalho de doutorado.

II. Monitoração para a geração de dados que auxiliem o escalonamento: Esta classe engloba os monitores utilizados para fornecer dados que possam auxiliar o escalonamento, e esses monitores podem ou não fazer parte do software de escalonamento. Na literatura consultada, foram encontrados quatro tipos de possíveis objetivos para esse tipo de monitoração.

Balanceamento de carga: Alguns softwares de escalonamento possuem monitores que são utilizados para auxiliar no balanceamento de carga. Por exemplo, o *NQE* (Cray Research, Inc, 2000) implementa um componente chamado *NLB* (*Network Load Balancer*), que é organizado da seguinte forma: um servidor, chamado de servidor *NLB*, funciona como um coletor que armazena em uma base de dados todas as informações enviadas por

observadores, chamados de clientes *NLB*, que devem estar executando em todos os processadores que possam ser utilizados para executar processos. Esses clientes são responsáveis por obter dados relacionados à carga dos diversos processadores.

O servidor *NQE*, que é responsável por escalonar os processos, pode ou não obter informações do *NLB* para decidir a que processador atribuir um processo. A estrutura fornecida pelo *NQE* é muito semelhante à de softwares de escalonamento como *Sun Grid Engine* (Sun Microsystems, Inc, 2002) e *LSF* (Platform Computing Corporation, 2000).

Caracterização da carga de trabalho: Monitores podem ser úteis para a atividade de caracterização da carga de trabalho, fornecendo subsídios para a escolha de um algoritmo de escalonamento. A tarefa de caracterizar uma carga de trabalho não é simples, principalmente em ambientes multi-usuário e heterogêneos, como sistemas distribuídos, visto que existe uma grande variação da carga de acordo com o período do dia e dos diferentes perfis dos usuários que estão utilizando o sistema. Exemplos de trabalhos que estudam a relação entre a carga de trabalho e o escalonamento são: (Feitelson 2003; Feitelson & Rudolph, 1995, 1998; Kunz, 1991; Shao et al., 1998; Zhang, 2000).

Monitorando processos para melhor escaloná-los: Alguns trabalhos citam casos específicos de monitoração aplicados a processos, de maneira que se possa aprimorar o escalonamento desses processos. Duas abordagens encontradas na literatura são: monitoração de características específicas de processos paralelos escalonados em grupos (*gang scheduling*) para melhor utilização de um computador paralelo (Arpaci-Dusseau et al., 1998; Frachtenberg et al., 2003; Nguyen et al., 1996; Schnor & Petri, 1999; Silva & Scherson, 2000); monitoração para prever a necessidade de recursos de um processo para melhor escaloná-lo (Devarakonda & Iyer, 1989; Smith et al., 1998).

Monitoração da utilização de recursos do sistema para auxiliar as decisões de escalonamento: Alguns trabalhos citam softwares de monitoração utilizados na medição da utilização de recursos do sistema com o intuito de obter dados que sejam utilizados para aprimorar as decisões de escalonamento (Berman et al., 2003(a); Wolski et al., 1999).

III. Monitoração para a avaliação do escalonamento: Monitores podem ser usados para avaliar a próprio escalonamento de processos como um todo (Ribler et al., 2001). Por exemplo, podem fornecer dados que permitam avaliar o quanto os resultados

obtidos pelo escalonamento estão condizendo com os objetivos propostos. Essa é a abordagem proposta por esta tese de doutorado, e ela será discutida em detalhes na próxima seção.

4.3 Avaliando o Escalonamento de Processos Através de Monitoração

Como foi exposto na seção anterior, existem vários motivos que justificam a utilização de monitores para auxiliar o escalonamento de processos. Os vários trabalhos científicos já publicados sobre o assunto demonstram a importância de se unir as áreas de pesquisa.

Todas as abordagens mostradas na seção anterior têm o mesmo objetivo, ou seja, melhorar o escalonamento e permitir uma melhor utilização do sistema computacional disponível. O que as diferencia é basicamente a maneira pela qual os dados coletados são utilizados.

Se por um lado a abordagem proposta nesta tese de doutorado poderia ter sido incluída na classe II, visto que a idéia principal dela é fornecer dados que auxiliem o escalonamento, a maneira pela qual os dados obtidos da monitoração são utilizados justificam a sua inclusão em uma classe diferente.

A classe II inclui abordagens onde os dados coletados são utilizados para auxiliar a escolha de algoritmos de escalonamento ou para auxiliar as decisões de escalonamento de um determinado algoritmo, ou seja, o dado monitorado é usado efetivamente na decisão de como será feito o escalonamento.

A abordagem seguida neste trabalho tem como objetivo utilizar os dados coletados para avaliar o escalonamento, com o intuito de determinar se o escalonamento está sendo bem feito. A informação que é retornada pelo monitor não é usada, por exemplo, para que o software de escalonamento faça uma determinada decisão de escalonamento ou escolha um algoritmo de escalonamento específico, e sim para fornecer subsídios para que o software de escalonamento possa verificar se o algoritmo de escalonamento atual está alcançando os objetivos pretendidos.

E por que é importante avaliar o escalonamento de processos? As seguintes características, que exercem influência no escalonamento de processos, justificam a

necessidade dessa avaliação: multiplicidade de objetivos, heterogeneidade de sistemas distribuídos e variação da carga de trabalho. Cada um delas é discutida a seguir.

Multiplicidade de objetivos: Como já foi discutido no decorrer desta tese, um escalonamento de processos é feito baseado em um objetivo que deve ser usado como um fator referencial para decidir se o escalonamento está atuando de maneira eficiente ou não.

Existem diversos objetivos possíveis, e não é possível implementar um algoritmo de escalonamento que retorne resultados eficientes para todos os objetivos, isto é, um escalonamento pode ser satisfatório para um objetivo e insuficiente para outro. Exemplos de objetivos são: maximizar a utilização de um processador, minimizar o tempo de resposta, balancear as cargas, evitar que existam processadores livres, maximizar a utilização de um determinado recurso, etc. (ver seção 2.6).

Um algoritmo de escalonamento genérico pode alcançar ou não um determinado objetivo. Mesmo um algoritmo de escalonamento que tenha sido desenvolvido especificamente para alcançar um objetivo pode falhar em suas atribuições por causa de uma carga de trabalho diferenciada, por exemplo. Ou um algoritmo de escalonamento adaptativo pode ter sido configurado de maneira que não obtenha o resultado pretendido.

Heterogeneidade do sistema: Muitos sistemas distribuídos são heterogêneos, e essa característica influi de maneira decisiva na qualidade de um escalonamento, visto que geralmente é mais simples escalonar processos em ambientes homogêneos. Por isso, é importante que periodicamente o software de escalonamento verifique o estado do sistema computacional e tome ações corretivas no sentido de melhorar o desempenho.

Variação da carga de trabalho: Outro fator que influencia o desempenho apresentado por um algoritmo de escalonamento é a variabilidade da carga de trabalho imposta ao sistema, fato bastante comum em sistemas distribuídos, que são compartilhados por diversos usuários. Além disso, a carga de trabalho varia de acordo com o dia (dias úteis ou finais de semana) e com o horário (dentro e fora do horário comercial). Uma variação da

carga de trabalho pode fazer com que um algoritmo de escalonamento possa deixar de ser eficiente.

Pelo que foi exposto acima, o ideal é que o algoritmo de escalonamento seja flexível a ponto de poder se modificar quando o desempenho do sistema computacional se tornar insuficiente. Algoritmos de escalonamento adaptativos conseguem, até certo ponto, lidar com essa instabilidade no comportamento de um escalonamento, mas essa não é a solução definitiva. O ideal é que se possa mudar o próprio algoritmo de escalonamento de acordo com a situação do sistema.

Alguns softwares de escalonamento oferecem essa possibilidade, como por exemplo, *Sun Grid Engine* (Sun Microsystems, Inc, 20002), *PBS* (Veridian Systems, 2000) e *AMIGO* (Souza, 2000(a)), mas embora ofereçam a infra-estrutura necessária para isso, ainda não possuem técnicas automatizadas que possam definir exatamente quando e por qual algoritmo trocar, sendo que geralmente essa tarefa ou é feita pelo administrador do sistema, ou não é feita.

Existindo algoritmos adaptativos ou mesmo a possibilidade de troca de algoritmos de escalonamento, é muito importante que o software de escalonamento possua ferramentas para que possa avaliar o seu próprio desempenho.

Essa é a abordagem seguida pelo monitor PSMS, que é proposto nesta tese de doutorado. O PSMS é um software de monitoração desenvolvido especialmente para avaliar o escalonamento de processos e é discutido em detalhes no próximo capítulo.

Existem outros projetos de softwares de monitoração, que também foram desenvolvidos para serem usados em conjunto com softwares de escalonamento e que apresentam abordagens diferentes à do PSMS. Exemplos desses softwares são apresentados na próxima seção.

4.4 Softwares de Monitoração Para o Escalonamento de Processos.

A literatura científica em computação cita diversos exemplos de ferramentas que fazem monitoração em sistemas distribuídos, com propósito de, por exemplo, analisar o

desempenho de aplicações paralelas e administrar cluster de computadores (Baker & Buyya, 1999).

Ferramentas de análise de desempenho oferecem rotinas que permitem monitorar o comportamento de uma aplicação paralela, permitindo ao programador um entendimento maior das características de desempenho dessa aplicação. Exemplos desse tipo de ferramenta são: *SvPablo*, que oferece rotinas que permitem analisar os dados coletados (DeRose et al., 1998); e *Vampir*, que permite visualizar graficamente os dados de desempenho coletados das aplicações (Pallas, 1999).

Ferramentas de administração de clusters geralmente permitem que se monitore um sistema computacional em diversos níveis, permitindo que se tenha uma visão geral da utilização de recursos do sistema. Os dados coletados são, geralmente, utilizados para auxiliar as decisões de controle e manutenção do cluster. *Berkeley NOW* (Anderson et al., 1995) e *PARMON* (Buyya et al., 1998) são exemplos de ferramentas que permitem, inclusive, que os dados monitorados sejam visualizados através de navegadores (*browsers*) pela Internet.

Ambos tipos de ferramentas podem gerar dados que podem ser utilizados para auxiliar o escalonamento de processos. Porém, dentro do contexto definido por esta tese de doutorado, é interessante analisar os softwares de monitoração que tenham sido desenvolvidos especialmente para auxiliar o escalonamento de processos, de maneira que se possua uma visão dos trabalhos correlatos ao PSMS. Os principais softwares de monitoração encontrados na literatura, *NWS* e *Autopilot*, são discutidos a seguir.

4.4.1 NWS – Network Weather Service

NWS foi criado a partir de um projeto desenvolvido na Universidade da Califórnia, campus de Santa Bárbara e é utilizado e citado em trabalhos científicos desenvolvidos em diversas instituições de pesquisa (Wolski, 1999).

NWS é um software de monitoração que periodicamente avalia a utilização dos recursos computacionais de uma rede de computadores e faz previsões sobre o desempenho que pode ser oferecido por esses recursos em um tempo futuro. O nome *Network Weather*

foi cunhado baseado nas semelhanças que o *NWS* tem com um sistema de previsão de tempo.

Os dados de desempenho coletados no sistema são: desempenho da rede de comunicação (latência e largura de banda disponíveis) e percentual de CPU e memória livres. Nesses dados são aplicados métodos estatísticos que permitem que se possa fazer previsões de como estará a carga do sistema computacional dentro de um período de tempo estipulado.

O sistema *NWS* foi desenvolvido para interagir com softwares de escalonamento dinâmicos. Por exemplo, as previsões por ele geradas são utilizadas através de uma metodologia de escalonamento chamada *Apples* (*Application Level Scheduling*), desenvolvida na Universidade da Califórnia, campus de San Diego (Berman et al., 2003(a)). Através da combinação dos requisitos de uma aplicação paralela com as previsões fornecidas pelo *NWS*, o sistema *Apples* simula diversos escalonamentos para essa aplicação, e para cada um dos escalonamentos é gerada uma estimativa do desempenho a ser alcançado. O escalonamento que conseguir o melhor resultado (o usuário escolhe qual métrica deve ser utilizada para avaliar o escalonamento) é selecionado e utilizado.

4.4.2 *AutoPilot*

O *AutoPilot* faz parte do projeto Pablo, um dos principais projetos relacionados à avaliação de desempenho do mundo, e é desenvolvido pela Universidade de Illinois, em Urbana-Champaign (Ribler et al., 2001).

O objetivo do *AutoPilot* é integrar em um mesmo software ferramentas de monitoração e algoritmos de escalonamento configuráveis que se adaptam aos diversos estados do sistema computacional.

Através de sensores distribuídos em todos os processadores disponíveis são coletadas métricas de desempenho relacionadas à utilização dos recursos por parte das aplicações. Sobre esses valores são aplicados processos de decisão, utilizando lógica nebulosa (lógica *fuzzy*), que fornecem diretivas a serem seguidas pelos algoritmos de escalonamento. Os resultados obtidos do módulo nebuloso são então utilizados para

reconfigurar algoritmos de escalonamento adaptativos projetados especificamente para esse fim.

O *Autopilot* é um software de monitoração bastante completo, pois não se limita apenas a coletar dados e analisá-los através de técnicas estatísticas, como faz o *NWS*. Os dados coletados são interpretados e os resultados dessa interpretação guiam as decisões de escalonamento.

4.5 Considerações Finais

Este capítulo apresentou uma visão abrangente sobre a relação entre monitoração e escalonamento de processos, mostrando diversos enfoques onde se pode utilizar monitores para auxiliar as decisões de um software de escalonamento. A grande quantidade de trabalhos existentes na literatura mostra a importância de utilizar monitoração para melhorar o desempenho de um escalonamento de processos.

A monitoração é, muitas vezes, usada apenas como uma ferramenta auxiliar para se obter dados que validem determinadas propostas relacionadas ao escalonamento de processos, mas também é o alvo principal da pesquisa em diversos trabalhos, como por exemplo, nos dois projetos de softwares de monitoração apresentados neste capítulo (*NWS* e *AutoPilot*) que foram desenvolvidos por renomados grupos de pesquisa internacionais e que têm como único objetivo auxiliar o escalonamento de processos.

O PSMS e a abordagem que ele utiliza, propostos nesta tese, também foram desenvolvidos com esse objetivo em mente, e são descritos em detalhes no próximo capítulo.

5 PSMS – Um Sistema de Monitoração Para o Escalonamento de Processos

Este capítulo apresenta uma visão geral do PSMS, discutindo os seus objetivos de projeto, o tipo de sistema computacional para o qual se destina e também a sua estrutura interna organizada em camadas. Cada camada que constitui o PSMS é discutida em detalhes, utilizando alguns diagramas da linguagem UML, permitindo uma especificação mais formal.

5.1 Considerações Iniciais

O escalonamento de processos é uma atividade de extrema importância para o bom desempenho de um sistema computacional, e é muito importante que os algoritmos de escalonamento sejam sempre precisos em cumprir os objetivos aos quais se propõem. Porém, como já foi discutido no capítulo anterior, existem diversos aspectos que influenciam e tornam difícil garantir que o escalonamento seja sempre eficiente.

Partindo desse pressuposto, é importante que o software de escalonamento, de tempos em tempos, avalie o quanto a sua abordagem de escalonamento está alcançando os objetivos desejados. Em caso de desempenho abaixo da expectativa, o software de escalonamento opcionalmente pode trocar e/ou reajustar o algoritmo de escalonamento, ou mesmo migrar processos com o intuito de melhorar o desempenho.

O PSMS (*Process Scheduling Monitoring System* – Sistema de Monitoração para o Escalonamento de Processos) foi proposto com a intenção de preencher essa lacuna existente em grande parte dos softwares de escalonamento, que apenas distribuem processos sem verificar a eficácia ou não de suas decisões no desempenho do sistema.

É importante destacar que os problemas de desempenho nem sempre acontecem nos instantes seguintes ao escalonamento, pois os algoritmos de escalonamento utilizados por softwares de escalonamento modernos são bastante precisos em suas decisões de escalonamento. Mas é muito comum ocorrer variações na carga de trabalho que podem piorar o desempenho do sistema mesmo após um escalonamento bem sucedido.

Esse problema já foi abordado em outros trabalhos, e algumas soluções foram propostas, como, por exemplo, na abordagem seguida pelo *Autopilot* (Ribler et al., 2001), software de monitoração discutido no capítulo anterior. A abordagem proposta nesta tese difere dos trabalhos correlatos em diversos pontos, a iniciar pelos seus objetivos, os quais são discutidos na próxima seção.

5.2 Objetivos de Projeto

A abordagem de monitoração proposta nesta tese de doutorado, a qual é utilizada pelo PSMS, foi desenvolvida seguindo alguns princípios básicos que nortearam todas as decisões de projeto, que são:

Direcionamento ao escalonamento de processos: A monitoração aqui proposta é feita com o objetivo de avaliar o sistema sob o ponto de vista do escalonamento de processos. Por isso, toda a estrutura do monitor e as métricas de desempenho foram propostas levando em consideração a necessidade de avaliar algoritmos de escalonamento.

Proposta de uma camada de interpretação: A abordagem de monitoração não deve se limitar apenas a coletar os dados, mas também deve interpretá-los de maneira a retornar alguma informação ao software de escalonamento.

Propósito geral: O monitor deve ser projetado de maneira que possa interagir com qualquer software de escalonamento.

Definição abrangente de métricas: O monitor deve permitir que se obtenham o maior número possível de métricas, principalmente as que forem diretamente relacionadas ao escalonamento de processos.

Definição de regras de decisão: O monitor deve implementar regras de decisão que permitam tirar conclusões a respeito do desempenho do sistema baseado nos valores das métricas. Baseado nessas conclusões, o monitor deve retornar um valor numérico que quantifique o desempenho.

Possibilidade de extensão do monitor: O monitor deve permitir que novas métricas e novas regras de decisão sejam facilmente acrescentadas às já existentes.

Portabilidade: A utilização do monitor não deve implicar em nenhuma modificação no sistema operacional ou nas aplicações dos usuários.

Baixa sobrecarga: A interferência imposta pelo monitor no sistema deve ser mínima, de maneira que o sistema computacional não seja sobrecarregado em virtude da execução do próprio monitor.

Além de seguir os oito princípios de projeto citados acima, o PSMS foi proposto para um tipo específico de sistema computacional, o qual é discutido na próxima subseção.

5.2.1 Sistema Computacional Alvo

O termo sistema distribuído é bastante genérico, sendo aplicado para representar diversos tipos de sistemas computacionais. Baseado no tipo de uso que se faz do sistema distribuído, a abordagem de monitoração do PSMS pode ou não oferecer resultados úteis. Por esse motivo, é importante que seja definido o perfil que um sistema distribuído deve possuir para que a abordagem definida nesta tese seja totalmente aplicável.

Sistemas distribuídos de alta interatividade, que geralmente são utilizados para tarefas administrativas (em empresas, indústrias, universidades, etc.) são de difícil monitoração, visto que existe uma grande variação nos valores de suas métricas. Algumas características comuns a esse tipo de sistema são:

- É constituído por computadores pessoais do tipo PC de médio ou pequeno porte;
- Executa um sistema operacional com interface gráfica (geralmente uma versão do *Windows*, da *Microsoft*);
- É utilizado para editar textos, criar planilhas, acessar a Internet, etc.;
- Os processadores são conectados em rede a um servidor de arquivos e de autenticação de usuários;
- Os processadores são geralmente utilizados por uma única pessoa;

- As aplicações executadas nesse tipo de sistema são altamente interativas, de curta duração e geralmente executadas localmente.

Quaisquer métricas medidas em um sistema com essas características são bastante instáveis, pois existe uma grande variação da carga de trabalho sobre esses sistemas. Por conseguinte, as interpretações que possam ser feitas sobre elas também são pouco conclusivas.

Em contrapartida ao tipo de sistema distribuído caracterizado (utilizado geralmente em ambientes administrativos), alguns sistemas distribuídos, encontrados principalmente em ambientes de pesquisa e desenvolvimento, são utilizados com diferentes objetivos. Esse tipo de sistema distribuído possui as seguintes características:

- É constituído por computadores pessoais do tipo PC de médio ou grande porte, estações de trabalho ou computadores paralelos;
- Executa um sistema operacional *Unix-like*, como o *Linux* ou o *Solaris*;
- Os processadores são conectados por uma rede de alto desempenho;
- Os processadores são compartilhados por vários usuários;
- Executa predominantemente aplicações *batch* e paralelas, geralmente com pouca interação do usuário;
- Executam aplicações de médio e grande porte.

Sistemas distribuídos com essas características, que são predominantemente utilizados para executar aplicações científicas com pouca interação do usuário e de média e longa duração, caracterizam o tipo de sistema computacional para o qual o PSMS se destina.

Nesse tipo de sistema, geralmente a atuação do escalonamento de processos tem bastante influência no desempenho. Por exemplo, aplicações paralelas geralmente utilizam vários processadores, e uma má distribuição dos processos pode causar problemas de desempenho no sistema como um todo. Sistemas distribuídos com essas características são utilizados na computação em *clusters* (*cluster computing*) e na computação em *grid* (*grid computing*), por exemplo (Baker e Buyya, 1999; Berman(b), 2003).

Também podem ser citados sistemas distribuídos híbridos, utilizados em ambas atividades (administrativas e científicas) em períodos diferentes do dia. Não existe um fator que inviabilize a utilização da monitoração do PSMS durante o período em que os computadores são utilizados para fins administrativos, porém os resultados retornados nesse tipo de situação tendem a ser de pouca utilidade.

A partir da próxima seção, é apresentada a estrutura geral do PSMS, discutindo-se cada um de seus componentes em detalhes. Com o intuito de acrescentar algum formalismo ao projeto do PSMS, foram utilizados alguns diagramas da UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada) para especificar e tornar mais clara a estrutura do PSMS.

A UML é uma linguagem gráfica padronizada destinada à especificação, à construção, à visualização e à documentação de sistemas de software. Tem como objetivo oferecer a qualquer pessoa envolvida na produção, instalação e manutenção de software uma notação padronizada para expressar o projeto de um sistema de software (Booch et al, 2000).

A especificação UML não é o objetivo principal deste trabalho, e por isso não foram utilizados todos os recursos da linguagem, e sim apenas os necessários para uma apresentação mais clara do PSMS, facilitando a sua compreensão e futura extensão. Uma descrição dos objetivos da UML e dos diagramas utilizados nesta tese é apresentada no apêndice C.

5.3 Arquitetura do PSMS

Uma visão macro da organização do PSMS é apresentada na figura 5.1. De maneira geral, o escalonamento de processos sob a observação do PSMS envolve a interação de quatro entidades computacionais, mostradas na figura 5.1 como retângulos com a borda escura. As flechas indicam a interação entre as quatro entidades computacionais, e o sentido no qual a comunicação entre as entidades ocorre. Por exemplo, a flecha 1, que parte do software de escalonamento e termina no módulo central do PSMS, indica que quem inicia a comunicação é o software de escalonamento.

Figura 5.1 – Visão Geral da Arquitetura do PSMS

Cada um dos n processadores do sistema executa um *daemon*⁷ que desempenha a função de observador, ou seja, é responsável por coletar todos os dados necessários ao cálculo das métricas. Na figura 5.1, eles são representados por elipses numeradas de 1 até n . O **módulo central** é responsável pelas demais funções do PSMS, incluindo a comunicação com o **software de escalonamento**. O **administrador do sistema** interage diretamente com o módulo central, configurando e anexando novos módulos ao sistema de monitoração.

O escalonamento de processos sob a monitoração do PSMS envolve os seguintes passos (figura 5.1):

- O software de escalonamento faz uma requisição de monitoração para o PSMS informando o seu objetivo de escalonamento (flecha 1);
- A requisição é analisada pelo módulo central do PSMS, que escolhe uma métrica apropriada a ser calculada no sistema computacional;
- O módulo central requisita aos *daemons* observadores os dados necessários ao cálculo das métricas (flecha 2);
- Os *daemons* observadores coletam os dados nos diversos processadores e calculam as métricas que são posteriormente agrupadas e organizadas pelo módulo central (flecha 3);

⁷ Processo que executa ininterruptamente em um processador, geralmente em segundo plano (*background*), normalmente responsável por tarefas de gerenciamento do sistema.

- e. O módulo central retorna um valor numérico que representa o desempenho do sistema de acordo com o objetivo do escalonamento (flecha 4).

Esse valor representativo do desempenho, que nesta tese será nomeado como **valor de desempenho**, é calculado a partir da aplicação de **regras de decisão** sobre os valores das métricas obtidas no sistema computacional. Uma regra de decisão é definida como uma função que retorna um valor de desempenho a partir de um conjunto de métricas coletadas no sistema computacional.

Novas métricas e novas regras de decisão podem ser acrescentadas ao PSMS pelo administrador do sistema (flecha 5), de acordo com as necessidades particulares de um determinado sistema computacional. Por esse motivo, são definidos dois **repositórios**² de dados, onde são catalogadas todas as métricas e regras que são acessíveis pelo PSMS. Também é definido um repositório onde serão armazenadas as métricas medidas no sistema, chamado de **histórico do sistema**.

A figura 5.2 apresenta o diagrama UML de casos de uso do PSMS, o qual é utilizado para a modelagem dos aspectos dinâmicos do software, e apresenta um conjunto de casos de uso, atores e seus relacionamentos. Um caso de uso representa uma determinada função (seqüência de ações) desempenhada pelo software, a qual produz um resultado observável por um ator (que pode ser um usuário ou um software que interaja com o software especificado). Em outras palavras, o diagrama de casos de uso representa uma visão de alto nível do que o software faz, e como ocorre a interação entre os atores (usuário e demais softwares) com o software projetado.

No PSMS, tem-se dois atores: o administrador do sistema e o software de escalonamento. É importante ressaltar que o diagrama de casos de uso enfatiza a relação entre os atores e o software projetado, e por esse motivo dá-se, na figura 5.2, maior destaque aos casos de uso que representam funções desempenhadas pelo PSMS onde existe interação com o usuário ou com o software de escalonamento. A descrição de cada um dos casos de uso descritos no diagrama é feita no final da respectiva seção que trata sobre a função representada por eles.

² Banco de dados bastante simples.

Figura 5.2 – Diagrama UML de Casos de Uso do PSMS

A figura 5.3 mostra a organização do PSMS em termos da localização de seus componentes em um sistema computacional com quatro processadores (representados por retângulos com borda escura). O módulo central (que é composto pelos módulos de controle e decisão), o repositório de regras de decisão e de métricas, e o histórico do sistema são localizados em um processador que é chamado aqui de **servidor PSMS**³. O módulo central acessa os três repositórios de dados, conforme assinalado pelas flechas rotuladas com o número 1 na figura 5.3.

Os demais processadores, definidos como **clientes PSMS**, executam apenas um *daemon* de observação que acessa uma cópia local do repositório de métricas (flechas 2). O próprio servidor PSMS também funciona como um cliente, e também executa um *daemon*. Como já foi exposto nesta seção, os *daemons* são controlados pelo módulo central

³ Na figura 5.3, o servidor PSMS é maior do que os clientes apenas para tornar a figura mais clara. Na prática, os 4 processadores podem ser idênticos.

(flecha 3) e transferem as métricas coletadas em cada processador para o histórico do sistema (flecha 4).

Figura 5.3 - Localização dos Componentes do PSMS

A organização do módulo central do PSMS e dos três repositórios de dados é discutida em detalhes nas próximas seções. Como foi exposto na seção 3.4.1, um monitor para um sistema distribuído pode ser dividido em 7 camadas, que são: observação, coleta, análise, apresentação, interpretação, console e gerenciamento. Essa estruturação é utilizada para descrever cada um dos componentes do PSMS.

5.4 Camada de Observação

A camada de observação é mostrada na figura 5.4, e os seguintes componentes (representados por um quadrado de borda mais escura) desempenham funções para essa camada: módulo central, histórico do sistema, *daemon* de observação e repositório de

métricas. O módulo central e o histórico do sistema são representados de maneira simplificada, e a estrutura deles é mais bem descrita nas camadas superiores.

Figura 5.4 – Camada de Observação do PSMS

Como já foi exposto, o PSMS utiliza diversos *daemons* de observação, que são responsáveis por coletar os dados necessários ao cálculo das métricas de desempenho. Os *daemons* acessam o repositório de métricas, e de lá retiram a informação que necessitam para desempenhar suas funções. Todos os processadores do sistema executam um *daemon* e possuem uma cópia do repositório de métricas.

Cada entrada do repositório de métricas contém as seguintes informações: um identificador da métrica e o nome da função responsável por coletá-la. Os *daemons* simplesmente executam essa função periodicamente, sendo que o tempo entre duas execuções consecutivas (que define a taxa de entrada do monitor) e o identificador da

métrica a ser coletada são definidos e informados pelo módulo central. Os valores das métricas que forem medidos nos diversos processadores são armazenados no histórico do sistema.

Na primeira medição é armazenado o identificador da métrica, e para cada uma das medições subseqüentes, são armazenadas as seguintes informações no histórico do sistema: o valor da métrica coletada na respectiva observação e um valor de média ponderada das últimas métricas medidas no sistema.

Quando se mede o desempenho de um sistema computacional em um determinado instante, a métrica mais importante a ser considerada é a métrica medida naquele instante. Porém, é interessante também considerar, com menor peso, o desempenho do sistema nas últimas medições, permitindo que se tenha uma visão mais consistente da real situação de desempenho do sistema e minimizando a influência de uma métrica que porventura tenha um valor muito diferente do normal em virtude de um problema de software ou de uma condição anormal do sistema e que possa levar a conclusões erradas. Por esse motivo, em vez de analisar os valores puros das métricas é geralmente preferível fazer uma análise baseada em uma média ponderada.

Uma maneira interessante e eficiente de se calcular uma média ponderada sobre valores que são medidos em instantes consecutivos de tempo é através do método de média móvel exponencial.

A média móvel exponencial calculada depois de x observações é:

$$M_x = (1 - k)M_{x-1} + k.D_x \quad (5.1)$$

onde: M_x = Média exponencial calculada na x -ésima observação

k = Constante exponencial, que equivale a $2/(1+J)$

M_{x-1} = Média exponencial calculada na observação anterior ($x-1$)

D_x = Dado coletado na x -ésima observação

J = Janela de observação

A média móvel exponencial define que os pesos de cada um dos valores são definidos de acordo com uma função exponencial, conferindo maior peso aos valores mais recentes. O valor J (definido como janela de observação), que é utilizado no cálculo da constante exponencial k , é um valor variável que pode receber valores inteiros a partir de 2. Quanto menor o valor de J , maior será o peso conferido aos valores mais recentes. O método de média móvel exponencial é discutido em detalhes no apêndice B.

A média móvel exponencial possui muitas características que a torna adequada à abordagem de monitoração definida pelo PSMS. Além de ser uma média ponderada e de ter uma distribuição de pesos exponencial, essa distribuição de pesos é variável e dependente dos valores da janela de observação. Ou seja, de acordo com a métrica que se está medindo, pode-se utilizar valores de janela diferentes que definirão maior ou menor peso para as métricas mais recentes.

Além disso, o cálculo de uma média em um instante específico depende apenas do valor da média anterior e da métrica calculada nesse instante. Ou seja, o método é rápido e dispensa o armazenamento de todos os dados a partir dos quais se quer calcular a média ponderada.

O PSMS trabalha com diversos tipos de métricas, pois tem o objetivo de ser um monitor de propósito geral. Algumas métricas podem ser calculadas diretamente através de consultas ao sistema operacional, outras não. Por exemplo, o percentual de utilização da CPU de um computador pessoal é obtido diretamente do sistema operacional. Por outro lado, uma métrica como o *slowdown* deve ser calculada diretamente nas aplicações dos usuários, requerendo algum tipo de instrumentação nessas aplicações.

Para que se possam obter métricas sem a necessidade de modificações nas aplicações do usuário o PSMS permite que sejam criadas aplicações sintéticas, que neste trabalho serão definidos como *benchmarks*, que são inseridas no sistema computacional e a partir delas são calculadas as métricas. Além disso, através da implementação de *benchmarks*, o administrador do sistema pode criar métricas projetadas para necessidades específicas de um determinado sistema computacional.

É importante ressaltar que para o *daemon* de observação ou para o módulo central do PSMS não existe diferença entre métricas medidas diretamente no sistema operacional ou não. A única diferença está na maneira de calcular a métrica, que é definida através de uma função de obtenção de métricas, como exposto na figura 5.4.

5.4.1 Funções de Obtenção de Métricas

As funções de obtenção de métricas, catalogadas no repositório de métricas, têm um formato padrão e são executadas em períodos pré-determinados (de acordo com a taxa de entrada do monitor) pelos *daemons* de observação. Por definição, essas funções apenas retornam o valor da métrica, sendo que as funções de organização, cálculo das médias exponenciais e o armazenamento no histórico do sistema são desempenhadas pelo *daemon* de observação.

As métricas calculadas diretamente no sistema operacional já possuem suas funções de obtenção definidas, visto que o protótipo atual do PSMS faz uma monitoração bastante abrangente das informações retornadas pelo sistema operacional. Uma análise crítica dessas métricas e de sua utilização para avaliação do escalonamento de processos é apresentada no próximo capítulo.

O PSMS define quatro *benchmarks* básicos, cada um deles representando uma classe comum de aplicações, que são: *CPU-bound*, *disk-bound*, *memory-bound* e *network-bound*. Esses *benchmarks* são construídos através da implementação de uma aplicação que simula as características de utilização de recursos de cada classe. Por exemplo, para implementar um *benchmark CPU-bound*, foi desenvolvida uma aplicação que faz intenso uso da CPU, e cujo comportamento retrata com bastante semelhança o comportamento de aplicações do tipo *CPU-bound* reais.

A métrica que é calculada pela execução desses quatro *benchmarks* é a métrica de *slowdown*, e através dela pode-se ter uma idéia do *slowdown* de qualquer aplicação da mesma classe que esteja executando no sistema computacional.

Esses quatro *benchmarks* básicos também são utilizados para calcular quatro valores que representam a capacidade computacional de cada um dos processadores disponíveis em um sistema distribuído. Esses valores são chamados de **fatores de**

desempenho do sistema e são de suma importância para o correto funcionamento do PSMS.

Quando o PSMS é iniciado pela primeira vez, os quatro *benchmarks* básicos são executados em todos os processadores do sistema computacional (sem a concorrência de outros processos), e baseado no tempo de execução desses *benchmarks* é calculado um grupo de quatro valores de desempenho (relacionados à CPU, memória, disco e rede), sendo um grupo para cada processador. O processador que executar mais rápido um determinado *benchmark* terá um fator de desempenho maior, significando um maior poder de processamento.

O fator de desempenho da CPU, por exemplo, que é calculado a partir do *benchmark CPU-bound*, demonstra de maneira aproximada a capacidade computacional em termos de CPU de um determinado processador. Através desses fatores pode-se comparar diferentes processadores, fornecendo uma visão geral da heterogeneidade do sistema computacional. Seguindo raciocínio similar, pode-se calcular a heterogeneidade da memória, disco e rede de comunicação. É importante ressaltar que o fator de desempenho mede velocidade, e não tamanho. A diferença entre as quantidades de memória dos processadores não seria detectada pelo fator de desempenho.

Os fatores de desempenho também são utilizados com fins de normalização das métricas, como é exposto na próxima seção.

Além dos quatro *benchmarks* básicos, o administrador do sistema pode incluir novos *benchmarks* no sistema, respeitando o modelo de uma função de obtenção de métricas, ou seja, deve implementar uma função que retorne apenas um valor. Um exemplo prático da utilização de *benchmarks* é apresentado no capítulo 7, onde são discutidos casos de estudo da utilização do PSMS.

Para concluir esta seção, de maneira sucinta, a camada de observação do PSMS é constituída de vários *daemons* de observação, controlados pelo módulo central, cujas funções são: coletar métricas (definidas em um repositório de métricas), calcular a média exponencial dessas métricas e armazená-las no histórico do sistema.

Como existem vários *daemons*, um para cada processador disponível, existirão vários conjuntos de métricas no histórico do sistema, cada um relacionado a um

processador. A função de organizar as diferentes métricas dos diferentes processadores é feita pelas camadas de coleta e de análise, as quais são discutidas na seção 5.5.

5.4.2 Descrição dos Casos de Uso Desempenhados pela Camada de Observação

A camada de observação é responsável por desempenhar os casos de uso **Coletar Métrica** e **Executar Benchmark** (descritos na figura 5.2), que são descritos a seguir.

5.4.2.1 Caso de Uso: Coletar Métrica

Descrição: O PSMS inicia a coleta de métricas após uma requisição do software de escalonamento

Pré-condição: O software de escalonamento deve ter feito uma requisição de monitoração

Fluxo Principal de Eventos

1: *Daemons* de observação (um em cada processador do sistema) consultam o repositório de métricas e obtêm função computacional para obtenção das métricas

2: *Daemons* executam essa função computacional e iniciam a coleta das métricas

3: *Daemons*, em um intervalo de tempo definido pela taxa de entrada, coletam um dado do sistema operacional e calculam uma métrica

4: Para cada métrica obtida, calcula-se a média móvel exponencial das últimas métricas obtidas no sistema

4: *Daemons* armazenam métricas e médias no histórico do sistema

Fluxo Alternativo de Eventos

1: Extends: PSMS executa um *benchmark*, sobre o qual são calculadas as métricas

2: *Daemons* armazenam métricas no histórico do sistema

Pós-condição: O PSMS pode calcular o desvio padrão das métricas e/ou calcular um valor de desempenho

5.4.2.2 Caso de Uso: Executar Benchmark

Descrição: Os *daemons* de observação obtêm as métricas através da execução de um *benchmark*

Pré-condição: O software de escalonamento deve ter feito uma requisição de monitoração, e a métrica a ser coletada não pode ser obtida diretamente do sistema operacional

Fluxo Principal de Eventos

1: *Daemons* de observação (um em cada processador do sistema) consultam o repositório de métricas e obtêm *benchmark* para obtenção das métricas

2: *Daemons* executam esse *benchmark* e iniciam a coleta das métricas

3: *Daemons*, em um intervalo de tempo definido pela taxa de entrada, executam *benchmark*, e sobre ele calculam as métricas

4: Para cada métrica obtida, calcula-se a média móvel exponencial das últimas métricas obtidas no sistema

Pós-condição: Os *daemons* de observação têm a capacidade de calcular métricas que não sejam oferecidas pelo sistema operacional

5.5 Camadas de Coleta e de Análise

O módulo central do PSMS contém um componente chamado **módulo de controle**, o qual é responsável pelas tarefas definidas nas camadas de coleta e de análise. Por coleta define-se a atividade de obter e organizar as métricas medidas nos diversos processadores em um só conjunto para que possam ser interpretadas. Por análise define-se a aplicação de cálculos estatísticos sobre essas métricas antes de interpretá-las.

Se existem n processadores disponíveis, o histórico do sistema conterá n conjuntos de métricas, um para cada processador. Todos os conjuntos conterão a mesma métrica, visto que os *daemons* de observação sempre coletam a mesma métrica em todos os processadores.

Quando o conjunto de métricas alcança um tamanho determinado, o módulo de controle calcula o desvio padrão das últimas métricas coletadas. O momento em que o desvio padrão deve ser calculado e a quantidade de valores sobre os quais será calculado pode ser configurado pelo administrador do sistema.

A análise dos valores de desvio padrão de um determinado conjunto de métricas permite que se conclua a respeito da variabilidade das métricas medidas em um determinado processador. Se o valor do desvio padrão é muito alto, significa que os valores das métricas variaram de tal maneira que não é possível chegar a uma conclusão segura.

Uma variação muito grande dos valores das métricas geralmente acontece em duas situações: quando o sistema possui uma carga de trabalho muito flutuante ou está executando aplicações que requerem muita interação com o usuário. Independente do motivo, quando há alta variabilidade dos valores das métricas, o PSMS não retornará ao software de escalonamento um valor de desempenho, e sim um alerta sobre a alta variabilidade das métricas coletadas no sistema.

Se os valores das métricas são confiáveis, o módulo de controle reúne um conjunto de métricas, cada uma obtida em um dos processadores disponíveis, e os envia ao **módulo de decisão**, que será responsável por retornar um valor de desempenho ao software de escalonamento.

A figura 5.5 apresenta um exemplo desse processo de coleta e análise para um sistema computacional com três processadores (na figura, considera-se que os valores das métricas são confiáveis). Para cada um dos três processadores, existe um conjunto de métricas com o mesmo identificador de métrica (com valor 1, no exemplo da figura 5.5), visto que a mesma métrica foi medida em todos os processadores.

Os *daemons* de observação iniciam a coleta de dados sincronizadamente, visto que todos atuam sob o controle do módulo central do PSMS. Ou seja, os dados são coletados mais ou menos no mesmo instante de tempo em todos os processadores. O módulo de controle forma um grupo de n métricas (ou n médias exponenciais), cada uma medida em um processador diferente, e as envia para o módulo de decisão. Por definição, e para possibilitar a aplicação de diferentes regras de decisão, as métricas sempre são analisadas em grupos do tamanho definido pelo número de processadores disponíveis. No exemplo,

são enviados ao módulo de decisão três valores da média exponencial ($m1$, $m2$ e $m3$). Avaliar médias exponenciais permite uma avaliação mais consistente, conforme foi exposto na seção anterior.

Figura 5.5 – Camadas de Coleta e de Análise do PSMS

Opcionalmente, antes de serem enviadas ao módulo de decisão, é possível normalizar os valores das métricas. Essa normalização é importante quando se monitora sistemas computacionais heterogêneos, de maneira que se possa comparar as métricas na mesma escala, como se tivessem sido coletadas em um sistema homogêneo.

Essa normalização geralmente é feita através da utilização dos fatores de desempenho dos processadores. Por exemplo, na figura 5.5, se os três processadores fossem

heterogêneos, cada um deles teria um valor diferente para os fatores de desempenho. Supondo que se queira analisar métricas relativas à utilização da CPU, pode-se usar os fatores de desempenho da CPU para normalizar essas métricas.

Por exemplo, em cada um dos três processadores (P_1 , P_2 e P_3) é calculado um fator de desempenho⁴ da CPU (F_1 , F_2 e F_3). Em um instante qualquer, medem-se três métricas, uma em cada processador (M_1 , M_2 e M_3). Escolhe-se então um dos três fatores de desempenho, o qual será usado como base para a normalização. No exemplo, é utilizado F_1 . O cálculo das novas métricas $M_2normal$ e $M_3normal$ normalizadas em função de F_1 é feito através de uma regra de três simples. O seguinte raciocínio é aplicado: se uma métrica tem valor M_2 quando medida em um processador com fator de desempenho F_2 , qual seria o valor dela se fosse medida em F_1 ? Tem-se a seguinte relação:

$$\frac{M_2normal}{M_2} = \frac{F_1}{F_2} \quad (5.2)$$

Dessa relação, tem-se:

$$M_2normal = \frac{F_1 \cdot M_2}{F_2} \quad (5.3)$$

Aplicando a mesma relação para M_3 , é possível então analisar as três métricas reduzidas para a mesma escala, sendo então possível compará-las sem se preocupar com as diferenças arquiteturais do sistema computacional.

Concluindo, o módulo de controle desempenha as funções da camada de coleta e análise, que são: agrupar as métricas calculadas nos diversos processadores, aplicar algumas medidas estatísticas (desvio padrão, por exemplo) sobre essas métricas e normalizá-las para a mesma escala. Os grupos de métricas normalizadas são então transferidas ao módulo de decisão, que é responsável por retornar ao software de escalonamento o valor de desempenho.

⁴ Calculado através da execução do *benchmark CPU-bound* na instalação do PSMS

5.6 Camada de Interpretação

O módulo de controle do PSMS, como foi exposto na seção anterior, fornecia grupos de métricas que foram obtidas nos diferentes processadores em instantes de tempo semelhantes. O módulo de controle é responsável por enviar cada grupo de métricas ao módulo de decisão, juntamente com o identificador que indica qual regra de decisão deverá ser utilizada para calcular o valor de desempenho que será retornado ao software de escalonamento.

O valor de desempenho é um número inteiro que pode variar entre 1 e 5. A interpretação desses valores é mostrada na tabela 5.1.

Valor de Desempenho	Interpretação
1	Escalonamento altamente eficiente
2	Escalonamento eficiente
3	Escalonamento no limite de eficiência
4	Escalonamento ineficiente
5	Escalonamento altamente ineficiente

Tabela 5.1 – Interpretação dos Valores de Desempenho

Uma regra de decisão é uma função computacional que recebe como argumentos um vetor de métricas, analisa essas métricas e retorna um valor de desempenho cuja interpretação deve respeitar o definido na tabela 5.1.

As regras de decisão disponíveis são catalogadas em um repositório semelhante ao repositório de métricas, discutido na seção 5.4. Para cada regra de decisão, define-se um identificador da regra e uma função que a calcule, como pode ser observado na figura 5.6.

O módulo de decisão busca, de acordo com o identificador recebido, a regra de decisão necessária no repositório de regras de decisão. Essa regra é aplicada sobre o grupo de métricas, calculando assim o valor de desempenho, o qual é enviado para o módulo de

controle, que por sua vez se comunica com o software de escalonamento. Um exemplo desse processo é mostrado na figura 5.6, para um conjunto de três métricas, medidas em três processadores, conforme mostrado na seção anterior.

Figura 5.6 – Camada de Interpretação do PSMS

Os valores de desempenho são enviados ao software de escalonamento quando um determinado número de medições é alcançado, sendo que esse número é definido pelo administrador do sistema. Não há necessidade de que para toda medição seja calculado um valor de desempenho, embora seja possível. Porém, quanto maior o intervalo entre o cálculo dos diversos valores de desempenho, maior será o tempo de descoberta de um possível problema de desempenho. Por outro lado, se a regra de decisão gerar uma sobrecarga muito grande no sistema, deve-se executá-la com menor frequência, evitando

maior interferência no desempenho do sistema como um todo. Portanto, a escolha do intervalo de cálculo dos valores de desempenho deve ser baseada em um compromisso entre a sobrecarga imposta no sistema e o tempo de espera para a detecção dos possíveis problemas de desempenho.

No desenvolvimento do PSMS, foram propostas algumas regras de decisão, que são discutidas a seguir. Nesta seção apenas se faz a descrição delas, e exemplos de sua utilização são apresentados no capítulo 7, que mostra alguns estudos de caso.

Regra do coeficiente de variação: Esta regra caracteriza-se pelo cálculo do coeficiente de variação das métricas calculadas nos diversos processadores. Em uma grande gama de situações, é importante manter a carga do sistema balanceada de maneira que todos os processadores do sistema estejam igualmente “carregados”. Esse é um objetivo de escalonamento bastante comum, e se aplica a diversos recursos de um processador. Por exemplo, pode-se balancear a utilização da CPU, da memória principal, entre outros.

Para garantir um balanceamento de carga, é necessário que as métricas medidas no sistema também estejam “balanceadas”, ou seja, o valor do coeficiente de variação entre elas não pode ser muito alto. O coeficiente de variação, como é exposto no apêndice A, calcula o quanto um conjunto de valores estão dispersos, e quanto maior a aproximação entre eles, menor será o coeficiente de variação. Para que se possa utilizar essa regra, é necessário que as métricas estejam normalizadas.

Essa regra é bastante útil quando se trabalha com métricas cujos valores não possuem uma grande variação de grandezas (ou seja, a amplitude das possíveis observações é pequena), como por exemplo o *slowdown*. Em sistemas que não estejam sobrecarregados, essa métrica costuma variar entre 1 e 3, ou seja, a diferença entre os possíveis valores que ela pode possuir não é muito grande.

Com métricas cujos valores possíveis apresentam grande amplitude, como, por exemplo, o percentual de utilização de um determinado recurso (pode variar entre 0 e 100), geralmente não se aplica essa regra de decisão visto que os valores de coeficiente de variação são sempre elevados e pouco se pode concluir a respeito deles, como foi percebido nos experimentos conduzidos no desenvolver do trabalho descrito nesta tese.

Um estudo de caso da utilização dessa regra é mostrado no capítulo 7.

Regra da variação de frequências entre classes de métricas: Quando a amplitude dos valores possíveis que uma métrica pode assumir é elevada, é interessante reunir essas métricas em classes. Cada classe define um subconjunto de valores que uma determinada métrica pode assumir. Por exemplo, para uma métrica que calcule o percentual de utilização de um determinado recurso, que pode variar entre 0 e 100%, pode-se definir quatro classes:

- Classe I - Métrica entre 0% e 25%
- Classe II - Métrica entre 26% e 50%
- Classe III - Métrica entre 51% e 75%
- Classe IV - Métrica entre 76% e 100%

Medem-se então os valores das métricas em todos os processadores, e calcula-se a frequência de ocorrência dos valores em cada classe. Por exemplo, se forem medidas 5 métricas, e o valor de quatro delas estiverem na classe III e uma delas estiver na classe II, então a distribuição de frequências é a seguinte:

- Classe I – 0
- Classe II – 0,2
- Classe III – 0,8
- Classe IV – 0

A frequência de cada classe é calculada dividindo-se o número de métricas da classe pelo número total de métricas. Para que o sistema esteja com a carga balanceada, é importante que os valores de frequência estejam concentrados em uma mesma classe ou

classes consecutivas. No exemplo acima, a distribuição de frequências é boa, e portanto representa um sistema com um bom grau de balanceamento de carga.

A utilização dessa regra de decisão também é discutida no capítulo 7.

Regra do “valor mínimo” ou “valor máximo”: Através desta regra, define-se que todas as métricas coletadas devem possuir um valor mínimo (ou máximo), caso contrário considera-se o escalonamento ineficiente.

Essa regra pode ser útil para algoritmos de escalonamento que possuam como objetivo preservar determinados recursos computacionais disponíveis. Por exemplo, pode-se definir que um determinado processador poderá ser utilizado até que um valor máximo de *slowdown* seja alcançado.

A relação entre métricas e regras de decisão é flexível, e qualquer regra de decisão pode ser aplicada sobre qualquer métrica. Essa combinação de métricas e regras de decisão é feita pelo módulo de controle, que ao ativar o módulo de decisão define a regra de decisão a ser utilizada.

A definição de novas regras de decisão, ou a modificação de regras existentes para serem adaptadas às necessidades específicas de um determinado sistema computacional, é feita de maneira bastante simples, bastando apenas que a função receba como parâmetro de entrada um vetor de valores de métricas e tenha como saída um valor de desempenho que respeite a organização mostrada na tabela 5.1.

A definição de novas regras de decisão, novas métricas e a combinação entre as métricas e as regras é feita através da camada de console, através da qual o administrador do sistema interage com o PSMS, conforme é discutido na seção 5.7.

5.6.1 Descrição do Caso de Uso Desempenhado pelas Camadas de Coleta, de Análise e de Interpretação

As camadas de coleta, de análise e a camada de interpretação são responsáveis por desempenhar o caso de uso **Utilizar Regra de Decisão** (mostrado na figura 5.2), que é

descrito a seguir. Essas três camadas não desempenham funções que necessitem de interação direta com o administrador ou com o software de escalonamento, e por esse motivo são representadas por um caso de uso único (o diagrama de casos de uso é direcionado à interação entre o software sendo projetado e os elementos que o utilizam - usuários ou softwares).

5.6.1.1 Caso de Uso: Utilizar Regras de Decisão

Descrição: O PSMS aplica uma regra de decisão sobre as métricas, com o objetivo de calcular um valor de desempenho

Pré-condição: O software de escalonamento deve ter feito uma requisição de monitoração e as métricas devem estar disponíveis no histórico do sistema

Fluxo Principal de Eventos

- 1: PSMS calcula o desvio padrão das métricas medidas no mesmo processador
- 2: PSMS verifica que os valores do desvio padrão em todos os processadores estão dentro dos valores aceitos
- 3: PSMS forma um conjunto de métricas (sendo que o número de métricas equivale ao número de processadores, ou seja, para cada processador utiliza-se uma métrica)
- 4: PSMS normaliza os valores das métricas do conjunto
- 5: PSMS aplica uma regra de decisão sobre o conjunto de métricas
- 6: PSMS disponibiliza o valor de desempenho resultante da aplicação da regra de decisão

Fluxos Alternativos de Eventos

1º Fluxo

- 4: PSMS aplica uma regra de decisão sobre o conjunto de métricas,
- 5: PSMS disponibiliza o valor de desempenho resultante da aplicação da regra de decisão

2º Fluxo

2: PSMS verifica que o valor do desvio padrão de algum dos processadores está acima do valor mínimo aceitável

3: PSMS retorna um valor de desempenho nulo, que informa ao software de escalonamento que não se pode tirar conclusões seguras sobre as métricas coletas

4: PSMS disponibiliza o valor de desempenho nulo

3º Fluxo

1: PSMS aplica uma regra de decisão sobre um conjunto de métricas, sendo que o número de métricas equivale ao número de processadores, ou seja, para cada processador utiliza-se uma métrica

2: PSMS disponibiliza o valor de desempenho resultante da aplicação da regra de decisão

Pós-condição: O software de escalonamento pode buscar o valor de desempenho

5.7 Camada de Console

A camada de console compreende as funções do PSMS que são configuráveis através da ação do administrador de sistema. Como foi exposto anteriormente, um dos objetivos de projeto do PSMS é que ele seja flexível para se adaptar às diferentes necessidades de monitoração de diferentes sistemas computacionais.

Por isso, novas métricas e novas regras de decisão podem ser anexadas aos respectivos repositórios pelo administrador do sistema, conforme já mostrado nas seções anteriores. A cada nova métrica definida, o administrador deve escolher uma regra de decisão disponível ou criar uma nova, a qual será utilizada para avaliar a nova métrica. Também, deve ser definida a taxa de entrada do monitor, ou seja, o número de observações a serem feitas em um determinado período de tempo.

O módulo de controle mantém uma tabela relacionando métricas e regras de decisão, sendo que uma mesma métrica pode ser avaliada segundo diferentes regras de decisão. Cada combinação métrica/regra de decisão define um diferente objetivo de escalonamento a ser avaliado, e essa tabela é acessível ao software de escalonamento, e é através dela que é escolhida a abordagem de monitoração.

Por exemplo, suponha que sejam definidas duas métricas, *slowdown* de aplicações *CPU-bound* e percentual de utilização da memória. Essas métricas são catalogadas no repositório de métricas e cada uma é identificada por um identificador de métrica (M_1 e M_2 , respectivamente). Cada uma das métricas é coletada segundo a mesma taxa de entrada (T_1).

Sobre essas duas métricas são aplicadas duas regras de decisão, catalogadas no repositório de regras, cada uma identificada por um identificador de regra (R_1 e R_2).

Cada linha da tabela de métricas/regras contém as seguintes entradas: identificador de objetivo, descrição do objetivo, identificadores de métrica e de regra de decisão e taxa de entrada. Para esse exemplo, tem-se:

Id.	Descrição	Id. da Métrica	Id. da Regra de Decisão	Taxa de Entrada
O_1	Balancear a carga das CPUs	M_1	R_1	T_1
O_2	Balancear a utilização da memória	M_2	R_2	T_1

Tabela 5.2 – Tabela de Métricas e Regras

Além da definição da tabela de métricas/regras, cabe também ao administrador do sistema definir:

- O intervalo de medições que são feitas para que se retorne um valor de desempenho ao software de escalonamento;
- Um valor máximo de desvio padrão entre as últimas métricas medidas, caso onde a variabilidade das métricas obtidas não permite fazer uma análise conclusiva.

Outras opções de configuração do PSMS são discutidas no capítulo 7, que apresenta uma descrição do protótipo que foi implementado para testar as funcionalidades do software de monitoração aqui proposto.

Além da interação do administrador do sistema, o PSMS também interage com o software de escalonamento, que é o responsável por requisitar uma determinada monitoração, como é exposto na seção 5.8.

5.7.1 Descrição dos Casos de Uso Desempenhados pela Camada de Console

A camada de console é responsável por desempenhar os caso de uso *Atualizar PSMS*, *Inserir Métrica*, *Inserir Regra de Decisão* e *Atualizar Tabela de Métricas/Regras* (mostrados na figura 5.2), que são descritos a seguir.

5.7.1.1 Caso de Uso: Atualizar PSMS

Descrição: O administrador do sistema faz uma atualização do PSMS, com a intenção de estender as funcionalidades do PSMS

Fluxo Principal de Eventos

1: Administrador do sistema inicia uma atualização do sistema

2: Administrador escolhe entre três opções

Inserir uma nova métrica

Inserir uma nova regra de decisão

Atualizar a tabela de métricas/regras

3: Administrador reinicia o PSMS

Pós-condição: O PSMS possui uma nova configuração

5.7.1.2 Caso de Uso: Inserir Métrica

Descrição: O administrador do sistema faz uma atualização do PSMS, com a intenção de inserir uma nova métrica

Fluxo Principal de Eventos

1: Administrador do sistema inicia uma atualização do sistema

2: Administrador acrescenta uma nova métrica de desempenho ao repositório de métricas

3: Administrador reinicia o PSMS

Fluxo Alternativo de Eventos

3: **Extend:** Administrador atualiza tabela de métricas/regras

4: Administrador reinicia o PSMS

Pós-condição: O PSMS adquire suporte a uma nova métrica de desempenho

5.7.1.3 Caso de Uso: Inserir Regra de Decisão

Descrição: O administrador do sistema faz uma atualização do PSMS, com a intenção de inserir uma nova regra de decisão

Fluxo Principal de Eventos

1: Administrador do sistema inicia uma atualização do sistema

2: Administrador acrescenta uma nova regra de decisão ao repositório de regras de decisão

3: Administrador reinicia o PSMS

Fluxo Alternativo de Eventos

3: **Extend:** Administrador atualiza tabela de métricas/regras

4: Administrador reinicia o PSMS

Pós-condição: O PSMS adquire suporte a uma nova regra de decisão

5.7.1.4 Caso de Uso: Atualizar Tabela de Métricas/Regras

Descrição: O administrador do sistema faz uma atualização do PSMS, com a intenção de atualizar a tabela de métricas/regras

Pré-condição: As métricas de desempenho e as regras de decisão devem ter sido previamente inseridas

Fluxo Principal de Eventos

1: Administrador do sistema inicia uma atualização do sistema

2: Administrador atualiza tabela de métricas/regras, acrescentando uma nova linha à essa tabela, construindo uma relação entre uma métrica e uma regra de decisão

3: Administrador define um objetivo de escalonamento que caracteriza essa nova linha e define o intervalo de tempo entre medições consecutivas da métrica

4: Administrador reinicia o PSMS

Fluxo Alternativo de Eventos

2: Administrador modifica uma linha (um objetivo de escalonamento) já existente na tabela de métricas/regras

3: Administrador reinicia o PSMS

Pós-condição: O software de escalonamento pode iniciar uma monitoração para o novo objetivo de escalonamento recém inserido

5.8 Comunicação Entre o PSMS e o Software de Escalonamento

A comunicação entre o PSMS e o software de escalonamento ocorre em duas fases. Inicialmente, através da consulta à tabela de métricas/regras, o software de escalonamento escolhe um identificador que represente o seu objetivo de escalonamento. Por exemplo, supondo que o objetivo de escalonamento seja balancear a carga do sistema em relação à CPU, percorrendo a tabela 5.2, o software de escalonamento escolhe o objetivo 1, e envia ao módulo de controle uma requisição de monitoração com identificador O_1 .

O módulo de controle inicia então a coleta das métricas identificadas por M_1 (com uma taxa de entrada T_1) e envia ao módulo de decisão o número da regra de decisão a ser utilizada, ou seja, R_1 .

Depois de um determinado número de observações, o módulo de controle requisita ao módulo de decisão que calcule um valor de desempenho, que é retornado ao software de escalonamento.

Após verificar o valor de desempenho retornado pelo PSMS, o software de escalonamento tem a opção de requisitar ao PSMS que termine a monitoração, ou que inicie uma monitoração de um novo objetivo de escalonamento.

A função do PSMS termina nesse ponto. Se há a necessidade de fazer alguma mudança no algoritmo de escalonamento, tal decisão cabe ao software de escalonamento. A

camada de gerenciamento, que inclui as possíveis mudanças a serem feitas no sistema computacional baseado nos resultados da monitoração, não é definida pelo PSMS.

5.8.1 Descrição dos Casos de Uso Envolvidos na Comunicação entre o Software de Escalonamento e o PSMS

A comunicação entre o software de escalonamento e o PSMS envolve os caso de uso *Requisitar Monitoração* e *Buscar Valor de Desempenho* (mostrados na figura 5.2), que são descritos a seguir.

5.8.1.1 Caso de Uso: Requisitar Monitoração

Descrição: O software de escalonamento requisita uma monitoração ao PSMS

Pré-condição: As métricas de desempenho e as regras de decisão a serem utilizadas devem ter sido previamente inseridas

Fluxo Principal de Eventos

1: Software de escalonamento acessa tabela de métricas/regras

2: Software de escalonamento escolhe, dentre os objetivos cadastrados na tabela de métricas/regras, qual o seu objetivo de escalonamento

3: **Uses:** PSMS inicia a coleta das métricas

4: **Uses:** PSMS calcula o valor de desempenho sobre as métricas

5: PSMS disponibiliza o valor de desempenho para software de escalonamento

Pós-condição: O software de escalonamento pode buscar o valor de desempenho calculado pelo PSMS

5.8.1.2 Caso de Uso: Buscar Valor de Desempenho

Descrição: O software de escalonamento busca um valor de desempenho gerado pelo PSMS

Pré-condição: O valor de desempenho deve ter sido gerado pelo PSMS

Fluxo Principal de Eventos

1: Software de escalonamento busca valor de desempenho retornado pelo PSMS

2: Software de escalonamento analisa o valor de desempenho e espera por um novo valor de desempenho

Fluxo Alternativo de Eventos

2: Software de escalonamento analisa o valor de desempenho e termina a monitoração feita pelo PSMS

Pós-condição: O software de escalonamento obtém um valor de desempenho, e pode ou não tomar uma medida corretiva para melhorar o desempenho

Nas seções anteriores foram discutidos em detalhes todos os componentes do PSMS. Para concluir este capítulo, a próxima seção apresenta uma visão geral de toda a estrutura do PSMS através de dois diagramas UML de atividades. Através desses diagramas tem-se uma visão de toda a sistemática do funcionamento do PSMS, permitindo uma melhor compreensão de sua estrutura como um todo.

5.9 Diagramas de Atividades do PSMS

Um diagrama de atividades é utilizado para representar aspectos dinâmicos de um sistema de software, e é essencialmente um gráfico de fluxo, mostrando o fluxo de controle entre as diversas atividades desempenhas pelo software. O PSMS possui duas funcionalidades independentes entre si, que são: a atualização do software, feita pelo administrador do sistema e a própria monitoração em si. Cada uma delas é descrita nos dois diagramas de atividades apresentados nas figuras 5.7 e 5.8.

Figura 5.7 - Diagrama de Atividades da Função de Atualização do PSMS

Figura 5.8 – Diagrama de Atividades da Função de Monitoração do PSMS

5.10 Considerações Finais

O PSMS, cuja estrutura foi apresentada neste capítulo, foi desenvolvido com a intenção de ser um monitor flexível para a avaliação do escalonamento de processos. Mesmo os softwares de escalonamentos mais modernos, que possuem algoritmos de escalonamento bastante sofisticados, estão sujeitos a escalonar processos de uma maneira que não alcance os objetivos pretendidos.

Dentro desse contexto, a função principal do PSMS é ser um monitor que auxilie o software de escalonamento na tarefa de avaliar o desempenho do sistema em resposta ao algoritmo de escalonamento utilizado. Para que seja de uso geral, o PSMS foi projetado para ser flexível, permitindo ao administrador do sistema criar suas próprias métricas e regras de decisão.

Por ser flexível, o PSMS pode ser utilizado não só por administradores de sistema interessados na utilização eficiente do sistema computacional que administram, mas também por pesquisadores da área de escalonamento de processos. Por exemplo, protótipos de algoritmos de escalonamento podem ser avaliados através da definição de métricas apropriadas. Também, pode-se estudar a influência de um determinado algoritmo de escalonamento em um sistema computacional executando uma determinada carga de trabalho.

Em qualquer uma das situações de uso do PSMS, é importante que sejam escolhidas métricas adequadas e que essas métricas sejam interpretadas consistentemente. Portanto, a eficiência dos resultados obtidos através do uso do PSMS depende fundamentalmente da escolha da métrica correta.

O próximo capítulo apresenta uma análise crítica das métricas de desempenho comumente utilizadas para avaliar o escalonamento de processos, mostrando seus prós e contras. Essa discussão tem o objetivo de fornecer diretrizes que auxiliem na escolha de métricas adequadas às possíveis situações de avaliação do escalonamento..

Além da análise crítica das métricas, o capítulo 7 também apresenta alguns estudos de caso, mostrando exemplos da combinação de métricas e regras de decisão, fornecendo um panorama completo das possibilidades de monitoração oferecidas pelo PSMS.

6 Análise de Métricas de Desempenho Aplicadas ao Escalonamento de Processos

6.1 Considerações Iniciais

Uma boa avaliação de desempenho, que permita obter resultados conclusivos, depende da correta escolha da métrica de desempenho a ser utilizada. Dentro do contexto definido por este trabalho, que compreende a avaliação de desempenho do escalonamento de processos através de monitoração, o papel da métrica é fundamental, pois é a partir dela que pode se concluir a respeito da qualidade ou não de um escalonamento de processos.

Como já foi exposto, um software de escalonamento possui um objetivo e a escolha da métrica deve estar atrelada a esse objetivo. Como existem diversos objetivos possíveis para um escalonamento, é importante que um software que se disponha a avaliar o escalonamento de processos tenha suporte a várias métricas.

Relacionar a métrica ao objetivo torna mais fácil a escolha da métrica, pois diminui-se o número possível de métricas que podem ser utilizadas. Por exemplo, se o objetivo de um software de escalonamento é otimizar o acesso à memória, a métrica escolhida provavelmente vai medir aspectos relacionados à memória. Porém, o problema da escolha da métrica apenas é minimizado, e não solucionado, pois ainda existem diversas possíveis métricas a serem consideradas dentro de um subconjunto específico de métricas. No caso de avaliar a utilização da memória, por exemplo, pode-se considerar: percentual de utilização da memória, quantidade de paginação, percentual de utilização de swap, etc.

Além de serem relacionadas aos objetivos de um escalonamento, as métricas de desempenho utilizadas pelo PSMS devem também ser adequadas ao tipo de monitoração que é proposta nesta tese. O PSMS é um monitor *on-line*, e as métricas são interpretadas no momento em que são coletadas.

Por exemplo, a métrica *throughput* pode ser usada para comparar dois algoritmos de escalonamento aplicados a uma mesma carga de trabalho. Terminada a execução da carga de trabalho, o algoritmo que possuir o melhor valor de *throughput* é considerado o mais eficiente. Nesse caso, a métrica de desempenho é interpretada no final da execução. O

PSMS, por outro lado, tem como objetivo interpretar os valores de *throughput* à medida que estão sendo coletados, procurando avaliar se o sistema, sob o ponto de vista dessa métrica, está com bom desempenho.

Para ambas abordagens pode-se utilizar monitoração e pode-se coletar a mesma métrica, mas o momento e a avaliação dos dados coletados são completamente diferentes.

Assim, a utilização de métricas de desempenho adequadas ao objetivo do escalonamento e a definição de regras de decisão apropriadas para interpretar essas métricas no momento em que são coletadas são pré-requisitos para que o PSMS possa retornar resultados satisfatórios.

Dentro desse contexto, este capítulo apresenta uma análise crítica das métricas de desempenho que podem ser utilizadas para avaliar um escalonamento de processos, enfocando a sua adequação à abordagem de monitoração definida pelo PSMS.

6.2 Análise Crítica das Métricas de Desempenho

A literatura científica computacional carece de trabalhos que discutam a relação entre métricas de desempenho e o escalonamento de processos. Uma exceção é o trabalho de Feitelson e Rudolph (1998), onde é apresentada uma análise das métricas de desempenho mais comuns usadas para avaliar o escalonamento de processos, discutindo a sua adequação a diferentes tipos de sistemas computacionais, definidos por eles como: *off-line*, *on-line* abertos e *on-line* fechados. É importante acrescentar que o trabalho de Feitelson e Rudolph é direcionado principalmente para o escalonamento de processos em computadores paralelos.

Uma descrição sucinta das conclusões obtidas por Feitelson e Rudolph é apresentada na próxima subseção, com o intuito de mostrar um ponto de vista diferente do definido por esta tese, que relaciona métricas a objetivos de escalonamento.

6.2.1 Métricas de Desempenho e Tipos de Sistema Computacional

Em sistemas *off-line*, considera-se que os processos estão todos disponíveis, e não haverá novas chegadas de processos no futuro. Em sistemas *on-line* fechados, não se tem o conhecimento prévio de todos os processos a serem escalonados, mas o sistema comporta

um número máximo de processos. Alcançado esse limite, só serão escalonados processos novos quando algum que estiver executando termine (e não seja reescalonado). Em sistemas *on-line* abertos não se tem o conhecimento prévio de todos os processos e não existe nenhum tipo de restrição. Os sistemas *on-line* abertos representam a maioria dos sistemas computacionais reais.

A seguir, são descritas as métricas citadas por Feitelson e Rudolph, analisando a sua adequação aos três tipos de sistemas computacionais considerados. A relação entre métricas e os tipos de sistemas é sumarizada na figura 6.1 (Feitelson & Rudolph; 1998).

Throughput: Essa métrica é mais dependente da taxa de chegada de processos do que do próprio escalonamento, e por esse motivo é pouco adequada para sistemas abertos. Em sistemas fechados, onde o número de processos é finito, e a taxa de chegada de processos depende da velocidade de execução das aplicações escalonadas, quanto maior o *throughput*, melhor será o desempenho do sistema.

Percentual de utilização de recursos de um sistema computacional: Exemplos de recursos são: CPU, memória, disco rígido e rede de comunicação. Da mesma maneira que o *throughput*, o percentual também é muito dependente da taxa de chegada de processos ao sistema, e por isso, é mais adequado para avaliar sistemas fechados.

Tempo de resposta: Tempo decorrido entre o início e o término de um processo. Métrica mais genérica que as duas citadas anteriormente, e por isso geralmente utilizadas para avaliar sistemas abertos.

Makespan: Tempo total de execução de um conjunto de processos, e por isso, é adequada para avaliar sistemas *off-line*, onde se procura minimizar o tempo de execução de um número finito de processos.

Slowdown: Variação da métrica tempo de resposta (definida na subseção 2.6.1), essa métrica também é adequada para sistemas abertos.

Relacionar métricas de desempenho a tipos de sistemas computacionais é bastante útil, mas não é adequado à abordagem de monitoração descrita nesta tese, que define que métricas devem ser relacionadas aos objetivos do escalonamento. Por exemplo, embora Feitelson e Rudolph argumentem que a métrica de *throughput* seja pouco adequada para

fazer uma avaliação genérica do escalonamento de processos em sistemas abertos, ela é adequada para avaliar algoritmos de escalonamento cujo objetivo seja relacionado ao *throughput*, mesmo em sistemas abertos.

Figura 6.1 – Relação Entre Tipos de Sistemas e Métricas de Desempenho

A próxima subseção apresenta uma discussão da relação entre métricas de desempenho e objetivos de escalonamento, que é a abordagem utilizada nesta tese.

6.2.2 Métricas de Desempenho e Objetivos de um Escalonamento de Processos

Como foi exposto no capítulo anterior, a avaliação feita pelo PSMS baseia-se na combinação entre métricas e regras de decisão, ambas escolhidas de acordo com o objetivo do escalonamento. Por isso, quando se utiliza um determinado algoritmo de escalonamento e tem-se a intenção de avaliar o comportamento desse algoritmo, deve-se definir com clareza qual é o seu objetivo, atrelando essa decisão ao tipo de monitoração oferecida pelo PSMS, que se baseia na análise *on-line* das métricas coletadas.

A enumeração dos possíveis objetivos que um escalonamento de processos pode possuir e das métricas que podem ser utilizadas para medir esses objetivos certamente levaria a uma extensa lista. Por exemplo, além das cinco métricas citadas por Feitelson e Rudolph, outras métricas menos frequentes (mas não menos importantes) podem ser encontradas em trabalhos científicos relacionados ao escalonamento de processos, como por exemplo: tamanho da fila de processos da CPU, tempo de fila (ou tempo de espera) de um processo, quantidade de migrações de processo, quantidade de mensagens transferidas pela rede de comunicação, *speed-up* de aplicações paralelas (Sinha & Kalé, 1992; Schnor

et. al, 1996; Topcuoglu et al., 2002), entre outras. Cada uma dessas métricas caracteriza um objetivo diferente, o que demonstra a grande variação de objetivos que um escalonamento pode ter.

Este trabalho não tem a intenção de apresentar uma lista exaustiva de todos os possíveis objetivos de um escalonamento e de todas as métricas de desempenho, até porque tal tarefa nunca estaria completa. Portanto, a análise apresentada aqui se relaciona às métricas e objetivos de escalonamento mais comuns encontrados nos trabalhos científicos.

Uma questão muito importante relacionada aos objetivos de um escalonamento de processos é a diferença que existe entre os diversos objetivos que são citados em trabalhos científicos, e os objetivos definidos por softwares de escalonamento reais. Grande parte das propostas encontradas em trabalhos científicos relacionados ao escalonamento de processos não se refletiu nos softwares de escalonamento reais, por diversos motivos. Por exemplo, geralmente essas propostas são de difícil implementação, necessitam da obtenção de informações que o sistema operacional não oferece e geram grande sobrecarga no sistema computacional.

Ou seja, embora na teoria existam diversos objetivos de escalonamento possíveis, na prática os softwares de escalonamento procuram distribuir processos de acordo com a necessidade de recursos que eles apresentem, procurando apenas balancear a carga do sistema em termos de utilização de CPU ou de memória. Portanto, geralmente o objetivo a ser alcançado por um algoritmo de escalonamento é otimizar o uso dos recursos, preferencialmente diminuindo-se o tempo de resposta das aplicações. Otimização do uso de um recurso e balanceamento de carga são considerados sinônimos neste trabalho. O termo balanceamento de carga é bastante utilizado com o sentido de balanceamento da carga imposta à CPU, por isso prefere-se usar o termo otimização do uso de um recurso, que pode ser qualquer recurso computacional, e não apenas à CPU.

O PSMS foi desenvolvido com o intuito de ser utilizado em sistemas distribuídos, e os softwares de escalonamento que executam sobre esses sistemas (discutidos na seção 2.6) basicamente possuem o objetivo de otimizar o uso dos recursos computacionais. Por esse motivo, a análise das métricas de desempenho aqui apresentada e a definição de regras de decisão são bastante focadas a (mas não limitadas a) esse objetivo.

As próximas subseções discutem as principais métricas de desempenho encontradas em trabalhos científicos, procurando analisar a adequação dessas métricas aos principais objetivos de escalonamento e à abordagem de monitoração do PSMS.

As métricas tempo de resposta e a sua variante *slowdown* têm uma subseção exclusiva por serem muito populares entre os pesquisadores da área de escalonamento de processos.

6.2.3 Tempo de Resposta e *Slowdown*

Em termos numéricos, a métrica tempo de resposta é a mais citada em trabalhos acadêmicos relacionadas ao escalonamento de processos. Geralmente, novas propostas relacionadas ao escalonamento de processos são avaliadas e têm sua eficácia comprovada através da métrica tempo de resposta, o que vem ao encontro das conclusões obtidas por Feitelson e Rudolph (1998), que citam a métrica tempo de resposta como a mais genérica.

Não existe uma estatística oficial sobre a utilização de métricas de desempenho em trabalhos de escalonamento de processos, mas dentre os artigos citados nas referências bibliográficas desta tese, a métrica tempo de resposta é citada muito mais vezes do que qualquer outra métrica.

Dois pontos podem ser destacados que explicam essa preferência pela métrica tempo de resposta:

- É mais fácil para um pesquisador provar que suas contribuições para o escalonamento de processos são realmente úteis apresentando os seus resultados práticos utilizando uma métrica bastante popular;
- Grande parte dos possíveis objetivos de um escalonamento de processos tem relação com a necessidade de diminuir o tempo médio de resposta.

A partir dessas constatações, é possível concluir que a métrica tempo de resposta é a melhor métrica para avaliar o desempenho de um escalonamento de processos? No contexto definido por esta tese de doutorado, a melhor resposta para essa pergunta é: o efeito desejado por grande parte dos algoritmos de escalonamento é diminuir o tempo

médio de resposta, mas nem sempre essa é a melhor métrica a ser utilizada para uma avaliação de desempenho *on-line*.

Por exemplo, os trabalhos de Giné et al. (2001) e Batat e Feitelson (2000) afirmam que um bom escalonamento de processos que leve em consideração a utilização de memória deve minimizar a quantidade de paginação¹ feita pelas aplicações do sistema. Para provar que um algoritmo de escalonamento direcionado à memória é melhor do que outro que seja direcionado à CPU, por exemplo, executam-se aplicações *memory-bound* e mostra-se que o tempo de resposta delas melhora quando se utiliza o algoritmo de escalonamento direcionado à minimizar a paginação.

Nesse caso, o algoritmo de escalonamento é validado (provou-se que ele funciona) através da métrica tempo de resposta, mas essa métrica não é a melhor escolha para monitorar o desempenho desse algoritmo durante sua utilização em sistemas computacionais. Para garantir que os tempos de resposta sejam bons, é importante garantir que a paginação seja minimizada. Então, um monitor *on-line* deve, idealmente, monitorar a quantidade de paginação do sistema, para garantir que o algoritmo de escalonamento esteja escalonando aplicações de maneira que se minimize a paginação.

Ou seja, os objetivos “minimizar tempo de resposta” e “minimizar paginação”, nesse exemplo, possuem forte relação. A escolha da métrica a ser utilizada para avaliar o algoritmo de escalonamento depende então do tipo de monitoração que se deseja fazer. Do ponto de vista de monitoração *on-line*, é mais útil monitorar a paginação. Para uma monitoração *off-line*, métricas relacionadas à paginação e ao tempo de resposta podem oferecer resultados úteis.

O que se quer concluir através desse exemplo é que embora a métrica tempo de resposta seja muito popular para avaliar o escalonamento de processos, e seja muito citada em trabalhos científicos, sua utilização para monitoração *on-line* é mais restrita. Porém, é muito importante ressaltar que aqui não se está diminuindo a importância dessa métrica, e sim analisando a sua eficácia para o tipo de monitoração oferecido pelo PSMS.

¹ Usada no gerenciamento de memória, consiste em transferir uma página da memória para o disco rígido, ou vice-versa.

A métrica tempo de resposta possui algumas peculiaridades interessantes relacionadas ao tipo de aplicação sobre a qual ela é medida e a maneira de calculá-la no sistema computacional sendo avaliado. Cada uma dessas peculiaridades é discutida a seguir.

Tempo de resposta e carga de trabalho: A métrica tempo de resposta pode oferecer resultados bastante interessantes para avaliar um algoritmo de escalonamento, mesmo quando se faz uma monitoração *on-line*. Porém, se for considerado um sistema distribuído, com cargas de trabalho heterogêneas, é difícil existir um algoritmo de escalonamento que garanta bons tempos de resposta para todas as possíveis cargas de trabalho. Ou seja, geralmente os objetivos de um algoritmo de escalonamento são direcionados a uma determinada carga de trabalho, e a métrica também deve ser calculada sobre essa carga de trabalho.

Por isso, quando se considera o tempo médio de resposta como métrica de desempenho, a carga de trabalho (sobre a qual serão calculados os valores de tempo de resposta) utilizada para avaliar o desempenho dos algoritmos de escalonamento deve ser adequada ao tipo de objetivo que esse algoritmo pretende alcançar. Por exemplo, para avaliar algoritmos de escalonamento direcionados à otimização de uso da memória do sistema, deve-se calcular o tempo de resposta de aplicações onde o uso de memória seja primordial.

Monitorar o tempo de resposta de um sistema computacional, geralmente implica na necessidade de instrumentação das aplicações. Além disso, o tempo de resposta é calculado no final da execução das aplicações. Para uma monitoração *on-line*, existe a necessidade de que se estime o tempo de resposta de uma aplicação no sistema em um determinado instante sem que nenhuma aplicação seja instrumentada. Com esse intuito, o PSMS trabalha com *benchmarks* e com a métrica *slowdown*, que é uma variação da métrica tempo de resposta que fornece resultados mais úteis para uma monitoração *on-line*.

Benchmarks e slowdown: Como já foi exposto no capítulo anterior, o cálculo de métricas que não sejam obtidas diretamente no sistema operacional é feito através da utilização de *benchmarks*. Ou seja, calculando o tempo de resposta de uma aplicação sintética, que possua as mesmas características de uma aplicação real, é possível fazer uma

estimativa do tempo de resposta das aplicações no momento da medição. O mesmo princípio das cargas de trabalho diferenciadas de acordo com o objetivo de escalonamento se aplica para os *benchmarks*. Ou seja, o *benchmark* deve possuir as características das aplicações que são o alvo principal do escalonamento de processos.

O PSMS define quatro tipos de *benchmarks*, descritos na subseção 5.4.1, cada um representando uma aplicação típica de um sistema computacional: *CPU-bound*, *memory-bound*, *disk-bound* e *network-bound*. Considerando, por exemplo, que o objetivo de um software de escalonamento seja otimizar o uso da CPU (balancear a carga do sistema com ênfase na utilização da CPU), a medição do tempo de resposta do *benchmark CPU-bound* fornece uma excelente aproximação do tempo de resposta de uma aplicação *CPU-bound* real em um determinado processador. E pelo tempo de resposta de uma aplicação *CPU-bound* é possível estimar o quanto a CPU de um processador está carregada com boa exatidão.

A abordagem de monitoração definida pelo PSMS utiliza preferencialmente a métrica *slowdown*, em vez do tempo de resposta, pois essa métrica fornece resultados mais conclusivos para uma monitoração *on-line*. Prefere-se o *slowdown* por dois motivos:

- Um tempo de resposta de valor 2 segundos, por exemplo, medido em um processador específico em um determinado instante, não permite tirar nenhuma conclusão sobre a carga do sistema. Por outro lado, um valor de *slowdown* igual a 2, indica que as aplicações executadas nesse processador estão demorando o dobro de tempo para terminar;
- Independente da duração da aplicação, o *slowdown* será sempre interpretado da mesma maneira, ou seja, valor de *slowdown* 1 significa que a aplicação está executando no melhor tempo possível.

A métrica *slowdown*, calculada sobre os *benchmarks*, pode ser utilizada para avaliar um escalonamento de processo cujo objetivo seja relacionado a qualquer um dos principais recursos computacionais (CPU, memória, disco e rede), bastando selecionar o *benchmark* apropriado ao recurso computacional.

A adequação do uso da métrica *slowdown* em relação aos possíveis objetivos de um escalonamento é discutida nas próximas seções, onde são apresentadas outras métricas citadas com menor frequência em trabalhos científicos, mas que possuem igual importância para uma monitoração *on-line*. Na próxima seção essas métricas são discutidas em conjunto, de acordo com o recurso ao qual elas sejam relacionadas, para que se possa fazer uma análise comparativa delas.

Inicialmente, apenas se discute a adequação das métricas aos possíveis objetivos de escalonamento. A sistemática de avaliação das métricas, através de regras de decisão, será discutida no próximo capítulo, onde são apresentados casos de estudo da utilização do PSMS.

6.2.4 Métricas Relacionadas à CPU

Basicamente, pode-se definir três métricas principais relacionadas à CPU, que são: *slowdown* de aplicações *CPU-bound*, percentual de utilização da CPU (percentual de tempo em que uma CPU mantém-se ocupada durante um determinado período de tempo) e tamanho da fila de processos da CPU. Cada uma dessas métricas é adequada para avaliar um algoritmo de escalonamento cujo objetivo seja diretamente ligado à métrica. Por exemplo, um algoritmo de escalonamento que tenha como objetivo diminuir o tamanho médio da fila de processos da CPU naturalmente seria avaliado através de sua respectiva métrica.

Um escalonamento de processos cujo objetivo seja otimizar a utilização das CPUs (ou seja, balancear a carga) disponíveis em um sistema distribuído pode ser avaliado através das métricas de *slowdown*, tamanho da fila de processos ou percentual de utilização, que fornecem uma boa aproximação da carga da CPU de cada um dos processadores do sistema.

Os valores das métricas de *slowdown* e percentual de utilização têm a vantagem de ser independentes de sistema operacional, enquanto que o tamanho da fila da CPU depende do tipo de sistema operacional e da maneira que se implementa o acesso à CPU. Além disso, as métricas *slowdown* e percentual de utilização levam em conta apenas a influência das aplicações que acessam a CPU, enquanto que a métrica tamanho da fila da CPU sofre

influência de quaisquer aplicações que estejam executando no processador, independente de estarem usando a CPU ou não.

A métrica percentual de utilização da CPU tem a desvantagem de não representar a carga da CPU em determinadas situações. Por exemplo, em um sistema que esteja executando apenas aplicações *CPU-bound*, uma CPU com dois processos possuirá o mesmo percentual de utilização que uma CPU com apenas um processo, ou seja, 100%.

Objetivo de Escalonamento	Métricas
Diminuir o tempo médio de resposta de aplicações <i>CPU-bound</i>	<ul style="list-style-type: none">• <i>Slowdown</i> de aplicações <i>CPU-bound</i>• Tamanho da fila de processos da CPU
Diminuir o tamanho médio da fila de processos da CPU	<ul style="list-style-type: none">• Tamanho da fila de processos da CPU
Otimizar o uso das CPUs (balancear a carga)	<ul style="list-style-type: none">• <i>Slowdown</i> de aplicações <i>CPU-bound</i>• Tamanho da fila de processos da CPU• Percentual de utilização da CPU
Compartilhar a carga entre as CPUs	<ul style="list-style-type: none">• Percentual de utilização da CPU• Tamanho da fila de processos da CPU• <i>Slowdown</i> de aplicações <i>CPU-bound</i>

Tabela 6.1. Objetivos e Métricas Relacionados à CPU

Portanto, embora a métrica *slowdown* tenha a desvantagem de incorrer em maior sobrecarga no sistema computacional sendo avaliado, ela é uma métrica que consegue fazer uma excelente estimativa da carga de uma CPU, independente do tipo de carga de trabalho que esteja sendo executada, e é a métrica padrão utilizada pelo PSMS.

As métricas percentual de utilização e tamanho da fila de CPU podem ser utilizadas, por exemplo, para algoritmos de escalonamento que tenham como objetivo compartilhar a carga, ou seja, não existe a preocupação de balancear a carga, apenas procura-se evitar que existam processadores livres. O software de escalonamento Condor escala processos com esse objetivo (Condor Team, 2004).

A tabela 6.1 mostra uma relação entre possíveis objetivos de escalonamento e métricas adequadas para avaliá-los. Boa parte dos objetivos pode ser avaliada por diferentes métricas, e na tabela são mostradas em negrito as métricas consideradas mais apropriadas para cada objetivo, segundo a filosofia de monitoração do PSMS.

6.2.5 Métricas Relacionadas à Memória

Existem duas maneiras de se analisar a utilização da memória de um sistema computacional. Uma delas é relacionada à quantidade de acessos que se faz à memória, e a outra é relacionada à quantidade de memória utilizada.

Algoritmos de escalonamento direcionados à quantidade de acessos levam em consideração basicamente a velocidade de acesso à memória. Ou seja, é interessante que os processadores que possuam memórias mais rápidas tenham preferência de escalonamento. Medir a velocidade de acesso à memória é possível através do uso de um *benchmark memory-bound*. Quanto menor o *slowdown* calculado sobre ele, menor será o tempo de acesso à memória do processador.

Porém, o problema da velocidade de acesso à memória em *clusters* de computadores tem sido minimizado através da utilização de memórias cache² com taxas de acerto muito grandes (90% em média) e tempos de acesso compatíveis com a velocidade da CPU. Portanto, a velocidade da CPU acaba determinando a carga de um processador, tornando desnecessário avaliar a velocidade da memória. Em sistemas distribuídos com essas características, por exemplo, pode-se calcular a métrica de *slowdown* sobre o *benchmark CPU-bound*, que o resultado será satisfatório, embora esteja se avaliando um objetivo de escalonamento relacionado à memória.

Por outro lado, quando se analisa a quantidade de memória utilizada, métricas de *slowdown* são pouco úteis. Nesse caso, o objetivo mais importante de um escalonamento de processos é evitar que um determinado processador tenha sua memória totalmente

² Memória eletrônica de menor tamanho e grande velocidade de acesso, onde são armazenados os dados que estejam sendo acessados com maior frequência.

preenchida e inicie a fazer *swap*³, o que geraria uma degradação de desempenho muito grande. Então, é importante que se utilize uma métrica que permita estimar se está havendo *swap* ou não. Se o objetivo for exclusivamente evitar o *swap*, pode-se utilizar uma métrica que meça a quantidade de acessos ao *swap*.

Em caso de um objetivo mais abrangente, pode-se avaliar a utilização da memória através de duas métricas: o percentual de utilização de memória e a quantidade de paginação.

A métrica de percentual de utilização da memória é útil para objetivos relacionados a otimizar a utilização de memória do sistema, balanceando a quantidade de memória utilizada em todos os processadores. Porém, é importante notar que esse objetivo não tem relação direta com a diminuição do tempo de resposta do sistema. Por exemplo, supondo que existam dois processadores disponíveis, onde se queira executar um processo que ocupe muita memória e 5 processos que usem pouca memória, a tendência é que se escale 5 processos no mesmo processador, degradando o tempo de resposta do sistema. Ou seja, esse objetivo é útil para sistemas computacionais que executem predominantemente processos que ocupem bastante memória.

A tendência atual de pesquisa em algoritmos de escalonamento direcionados à utilização da memória é considerar a quantidade de paginação de um sistema (Batat & Feitelson, 2000; Giné et al., 2000, Xiao et al., 2003). Quanto menor o número de paginações, melhor será o tempo médio de resposta do sistema. Porém, uma métrica que calcule o número de paginações é altamente dependente do sistema operacional e das técnicas de gerenciamento de memória que sejam utilizadas. Portanto, na prática, essa métrica oferece bastante dificuldade de interpretação, principalmente em sistemas heterogêneos.

A tabela 6.2 mostra uma relação entre possíveis objetivos de um escalonamento relacionado à memória e métricas adequadas para avaliá-los. Novamente, a métrica principal está escrita em negrito.

³ Porção do disco rígido utilizada quando termina o espaço disponível na memória principal. Por ser localizada no disco rígido, possui velocidade de acesso bastante inferior se comparada com a memória principal.

Objetivo de Escalonamento	Métricas
Diminuir o tempo médio de resposta de aplicações <i>memory-bound</i>	<ul style="list-style-type: none"> • <i>Slowdown</i> de aplicações <i>memory-bound</i> • Métrica relacionada à CPU
Otimizar o uso da memória em termos de número de acessos (balancear a carga)	<ul style="list-style-type: none"> • <i>Slowdown</i> de aplicações <i>memory-bound</i> • Métrica relacionada à CPU
Evitar que exista acesso ao <i>swap</i>	<ul style="list-style-type: none"> • Números de acessos ao <i>swap</i> • Percentual de utilização da memória
Otimizar o uso da memória em termos de quantidade utilizada (balancear a carga)	<ul style="list-style-type: none"> • Percentual de utilização da memória
Otimizar o uso da memória em termos de quantidade de paginação	<ul style="list-style-type: none"> • Quantidade de paginação

Tabela 6.2. Objetivos e Métricas Relacionados à Memória

6.2.6 Outras Métricas

Métricas relacionadas a disco e rede: A métrica *slowdown*, calculada sobre *benchmarks disk-bound* e *network-bound* pode ser utilizada para avaliar a otimização do uso de disco e rede de um sistema distribuído. Por exemplo, se o *slowdown* de um *benchmark disk-bound* em um determinado processador é pequeno, significa que o disco deste não está sobrecarregado. Essa métrica é especialmente útil, pois ela considera também a velocidade da CPU, além da capacidade de disco ou rede, e a capacidade da CPU tem grande influência sobre a velocidade de acesso ao disco ou à rede.

Também pode-se avaliar os acessos ao disco e à rede através de métricas quantitativas considerando, por exemplo, que processadores que estejam fazendo um maior número de acessos ao disco, por exemplo, estejam mais carregados. É importante que o número de acessos seja medido em bytes, ou seja, deve-se mensurar o número de bytes escritos (ou lidos) em disco ou transmitidos (ou recebidos) pela rede. O número de acessos que se faz a um dispositivo de E/S depende da organização de hardware e do sistema operacional que esteja executando, e não existe relação direta entre o número de acessos e a quantidade de bytes envolvida em cada acesso.

Um possível problema em analisar uma métrica quantitativa é que processadores mais poderosos tendem a comunicar mais rápido, mesmo que estejam fazendo um maior número de acessos. Ou seja, otimizar o uso do recurso só baseado em quantidade de acessos não necessariamente vai garantir melhor desempenho das aplicações. Uma alternativa para minimizar esse problema é normalizar as métricas medidas de acordo com a capacidade computacional do processador. Dessa maneira, a métrica considera, além do número de bytes, também a velocidade a qual esses bytes são transferidos.

Throughput: A métrica de *throughput* é basicamente encontrada em trabalhos científicos relacionados ao escalonamento de processos em computadores paralelos, sendo indicada basicamente para avaliar objetivos que sejam diretamente relacionados a ela.

Uma importante questão a ser discutida sobre as métricas de desempenho é analisar o quanto elas se adaptam a sistemas heterogêneos, tanto em nível arquitetural quanto configuracional. Na próxima subseção são feitas algumas ponderações sobre essa questão em relação às métricas discutidas nas subseções anteriores.

6.2.7 Métricas e Sistemas Distribuídos Heterogêneos

Para concluir a análise crítica das métricas de desempenho, é importante apresentar algumas considerações sobre a utilização delas em sistemas heterogêneos. Algumas métricas são diretamente utilizáveis em sistemas heterogêneos, outras não.

A métrica *slowdown* pode ser utilizada em sistemas distribuídos com heterogeneidade arquitetural, visto que não depende do sistema operacional para que sejam calculadas.

Métricas que mensuram características do sistema operacional, como por exemplo, quantidade de paginação e tamanho da fila de CPU, não são muito adequadas para avaliar sistemas distribuídos com diferentes arquiteturas, visto que é difícil compará-las uma vez que diferentes sistemas operacionais implementam de diferentes maneiras essas atividades. Idealmente, um sistema distribuído que seja avaliado por essas métricas deve executar inclusive a mesma versão de sistema operacional, e deve ser heterogêneo apenas em termos de capacidade e tamanho dos recursos.

Métricas que medem o percentual de utilização de um recurso, embora sejam dependentes do sistema operacional, se adaptam melhor a sistemas distribuídos heterogêneos visto que a sua interpretação é sempre a mesma, independente da arquitetura ou da configuração do processador sendo avaliado. E também não necessitam de normalização, pois como são expressas em valores de porcentagem, já se encontram representadas de uma maneira uniforme.

Concluindo, métricas que medem quantidade de bytes transferidos de um recurso também se adaptam bem a sistemas heterogêneos, embora nesse caso, em grande parte dos casos é importante que as métricas sejam normalizadas de acordo com a velocidade de acesso a esses recursos.

6.3 Considerações Finais

Este capítulo apresentou uma discussão da utilização das métricas de desempenho considerando os seguintes aspectos: a popularidade da métrica nos trabalhos científicos, a adequação da métrica aos diferentes objetivos de escalonamento, a adequação da métrica à abordagem de monitoração do PSMS e a adequação da métrica a sistemas heterogêneos.

Dentro do contexto definido por este trabalho de doutorado, em casos onde existe dúvida sobre qual métrica utilizar, recomenda-se que seja utilizada a métrica *slowdown* pelos seguintes motivos:

- É derivada de uma métrica muito popular (tempo de resposta);
- É adequada para um grande número de objetivos;
- Leva em consideração a capacidade computacional do processador;
- Adapta-se bem a sistemas heterogêneos.

Essa constatação não desmerece a utilidade das outras métricas, que também podem oferecer resultados bastante úteis. Grande parte das demais métricas, por exemplo, geram uma sobrecarga bastante inferior à sobrecarga gerada pelo *slowdown*. Porém, é importante fazer algumas considerações sobre elas.

Métricas que calculam o percentual de utilização (memória, disco, rede) têm a desvantagem de ignorar a capacidade computacional do processador (nem sempre a máquina que esteja com menor percentual de utilização será a melhor escolha em termos de rapidez de execução). Essa mesma característica é apresentada por métricas que são baseadas em quantidade de comunicação (por exemplo, quantidade de bytes transferidos/recebidos da rede e do disco rígido).

Métricas que calculam aspectos relacionados ao gerenciamento do sistema operacional, como por exemplo, tamanho da fila de acesso a um recurso ou quantidade de paginação, dependem da maneira como o sistema operacional implementa essas atividades, e devem ser utilizadas com grande cuidado em sistemas heterogêneos.

Todas as conclusões obtidas neste capítulo têm como objetivo principal fornecer diretrizes básicas que possam guiar a escolha de métricas adequadas aos propósitos do PSMS. Não se tem aqui o objetivo de esgotar o assunto, e grande parte dos trabalhos futuros a serem desenvolvidos a partir desta tese de doutorado são relacionados ao estudo das métricas de desempenho.

Dentro deste contexto, um dos objetivos do PSMS é exatamente facilitar o estudo de métricas de desempenho, fornecendo uma maneira sistemática de avaliar a utilização dessas métricas em sistemas computacionais. Para isso, além de permitir a coleta das métricas, pode-se também interpretá-las, ou seja, pode-se tirar conclusões relacionadas ao desempenho do sistema computacional a partir dessas métricas. Essa interpretação é possível através do uso de regras de decisão.

No próximo capítulo, são apresentados alguns estudos de caso da utilização do PSMS, onde são mostrados alguns exemplos da utilização das métricas de desempenho e sua interpretação utilizando regras de decisão. Os estudos de caso foram feitos utilizando o protótipo do PSMS, que é discutido na parte inicial do capítulo.

7 Protótipo e Estudos de Caso do PSMS

Este capítulo apresenta uma descrição do protótipo desenvolvido para o PSMS e alguns estudos de caso (relacionados à CPU e à memória) de sua utilização para a avaliação do escalonamento de processos.

7.1 Considerações Iniciais

O principal objetivo deste trabalho é propor uma abordagem de monitoração que seja dinâmica e extensível, mas também que seja factível de implementação em sistemas computacionais reais, e que possa futuramente ser utilizada por instituições de pesquisa e universidades distribuídas pelo mundo. Por esse motivo, a maior parte da abordagem proposta nesta tese foi implementada em um protótipo, o qual foi nomeado PSMS.

A implementação desse protótipo permite fazer uma avaliação das idéias propostas nesta tese, de maneira que se obtenham resultados práticos que venham mostrar a funcionalidade e utilidade da abordagem aqui descrita. Permite também que se façam cálculos da sobrecarga que o PSMS causa ao sistema, visto que essa é uma característica muito importante a ser considerada quando se implementa um monitor por software.

Através da utilização do protótipo, foram feitos alguns estudos de caso, mostrando exemplos do processo de monitoração completo, incluindo a definição dos parâmetros do PSMS e do relacionamento entre métricas de desempenho e regras de decisão. Essas tarefas são de fundamental importância para que os resultados retornados pelo PSMS sejam confiáveis e corretos.

A primeira seção deste capítulo apresenta uma descrição do protótipo, e a seguir são apresentados os estudos de caso desenvolvidos nesta tese, mostrando exemplos da utilização do PSMS para monitorar sistemas computacionais cujos escalonamentos têm objetivos relacionados à otimização do acesso à CPU e à memória.

7.2 Protótipo do PSMS

O desenvolvimento do protótipo do PSMS foi realizado com três objetivos de projeto principais, que são: portabilidade, modularidade e baixa sobrecarga. Cada um deles é descrito a seguir.

Portabilidade: Qualquer sistema computacional que possua características semelhantes às apresentadas na subseção 5.2.1. pode ser monitorado pelo PSMS. Além disso, a heterogeneidade de arquiteturas é uma característica bastante comum em sistemas distribuídos. Por esses motivos, é importante que o PSMS seja implementado utilizando uma linguagem de programação que permita portá-lo facilmente para diferentes arquiteturas computacionais com diferentes sistemas operacionais. Optou-se então por utilizar a linguagem C, padrão ANSI¹, que é uma linguagem de programação altamente portátil. O PSMS é um monitor distribuído, e as rotinas de iniciação de *daemons*, sincronização e troca de mensagens foram implementadas utilizando a biblioteca PVM (Beguelin et al., 1994).

Modularidade: O PSMS foi projetado de maneira modular, como foi discutido no capítulo 5, e a sua implementação seguiu as mesmas diretrizes, sendo que os módulos do PSMS foram implementados separadamente. A interface entre os diferentes módulos é definida de maneira bem clara, para que possam ser feitas modificações isoladas sem a necessidade de reestruturar todo o código fonte. Em especial, as funções de obtenção de métricas e as regras de decisão são totalmente desvinculadas de qualquer outro módulo e possuem uma estrutura bem definida, para que seja fácil a extensão do PSMS para incluir novas métricas e regras.

Baixa sobrecarga: Um requisito de muita importância no projeto de qualquer software de monitoração é a sobrecarga imposta por ele no sistema computacional. Para que a sobrecarga seja mínima, o software deve ser simples e a taxa de entrada do monitor deve ser baixa, ou seja, deve-se coletar o mínimo necessário de dados para que se obtenham

¹ Padronização da sintaxe da linguagem C que é reconhecida pela grande maioria dos compiladores disponíveis.

conclusões satisfatórias. Uma discussão sobre a sobrecarga gerada pelo PSMS é apresentada na subseção 7.2.2.

Na próxima subseção, é feita uma descrição da implementação do PSMS, mostrando diversos detalhes de sua implementação.

7.2.1 Descrição da Implementação

O PSMS foi implementado utilizando a estrutura de diretórios mostrada na figura 7.1:

Figura 7.1 – Estrutura de Diretórios do Protótipo do PSMS

O diretório *PSMS/library* contém as bibliotecas onde estão armazenadas as funções mais freqüentes utilizadas na implementação das rotinas do PSMS. Quatro bibliotecas são definidas nesta versão, que são:

- *Estat.h*: Funções estatísticas utilizadas pelos diversos módulos do PSMS. Podem também ser utilizadas para a implementação de regras de decisão. Alguns exemplos de funções estatísticas definidas nesta biblioteca são: média aritmética, média exponencial, desvio padrão, coeficiente de variação, etc.
- *Rules.h*: Contém todas as regras de decisão utilizadas pelo PSMS.
- *Metrics.h*: Contém todas as funções de obtenção das métricas.
- *Monitor.h*: Contém funções de uso geral do PSMS, como por exemplo, funções para coletar o tempo do sistema, criar arquivos no disco, etc.

O diretório *PSMS/bin* contém os arquivos executáveis. O programa² *PSMS.e* executa as funções do módulo central do PSMS, em apenas um processador e deve ser recompilado a cada nova regra de decisão inserida na biblioteca de regras (*rules.h*).

Os *daemons* de observação são implementados através do programa *PSMS.e.daemon*, e são responsáveis por executar as funções de obtenção das métricas. A cada nova métrica anexada ao sistema, esse programa deve ser recompilado. As métricas coletadas através das funções de obtenção de métricas são armazenadas em arquivos-texto localizados no diretório *PSMS/history*.

A atual estrutura do PSMS segue uma organização cliente-servidor, onde o servidor é o módulo central e os *daemons* de observação são os clientes, conforme discutido na seção 5.3. A comunicação entre eles é feita através de arquivos de configuração compartilhados entre o módulo central e os *daemons*, e por esse motivo, a atual versão do PSMS necessita que o sistema operacional possua um sistema de arquivos compartilhado, como o *NFS*³.

A tabela 7.1 apresenta o relacionamento entre os componentes do PSMS apresentados no capítulo 5 e os respectivos arquivos mostrados na figura 7.1, que formam a base de implementação do PSMS:

<i>PSMS.e</i>	Módulo central (módulo de controle e módulo de decisão)
<i>PSMS.daemon.e</i>	<i>Daemons</i> de observação
<i>Metrics.h</i>	Repositório para as funções de obtenção das métricas
<i>Rules.h</i>	Repositório para as regras de decisão
<i>/PSMS/History</i>	Histórico do sistema

Tabela 7.1 – Implementação dos Componentes do PSMS

De todos os arquivos da implementação do PSMS, apenas as funções de obtenção das métricas obtidas diretamente no sistema operacional não são portáteis, visto que dependem da maneira que os diferentes sistemas operacionais as oferecem. O protótipo do

² Programa aqui é definido como um arquivo executável, já compilado e *linkeditado*

³ Network File System é um protocolo para compartilhamento para sistemas de arquivos

PSMS executa sobre um sistema operacional *Linux*, que por ser um sistema operacional cujo código fonte é livre permite que se obtenham métricas bastante precisas sobre a utilização dos recursos computacionais.

A atual versão do PSMS faz uma monitoração bastante completa de um sistema computacional *Linux*, e a tabela 7.2 apresenta uma lista de todas as métricas que são coletadas, incluindo o arquivo de sistema de onde elas são coletadas.

Métrica	Obtida Em
Percentual de utilização da CPU	/proc/stat
Percentual de utilização da memória ou de <i>swap</i>	/proc/meminfo
Quantidade de paginações	/proc/stat
Quantidade de acessos ao <i>swap</i>	/proc/stat
Quantidade de bytes lidos ou transferidos do disco rígido	/proc/partitions
Tempo de serviço médio para uma operação de E/S no disco	/proc/partitions
<i>Throughput</i>	/proc/stat

Tabela 7.2 – Métricas Obtidas Pelo PSMS

Na próxima subseção são discutidos aspectos relacionados à sobrecarga imposta no sistema computacional pelo atual protótipo do PSMS.

7.2.2 Sobrecarga do PSMS Sobre o Sistema Computacional

O protótipo do PSMS foi desenvolvido com o intuito principal de não causar grande sobrecarga sobre o sistema computacional. Qualquer tipo de monitoração feita por software implica em alguma sobrecarga, e é importante garantir que o ganho, em termos de desempenho, proveniente da utilização do monitor seja superior à sobrecarga que ele produz. Ou seja, é importante que a relação custo/benefício do monitor seja positiva.

No caso específico do PSMS, cada processador do sistema computacional executa um *daemon* de observação, que de acordo com a taxa de entrada do monitor, calcula uma determinada métrica no sistema. Como foi exposto, o PSMS trabalha com métricas obtidas de duas maneiras: diretamente no sistema e através de *benchmarks*.

Métricas obtidas diretamente no sistema geralmente são coletadas através de consultas ao sistema operacional, que é uma tarefa que praticamente não causa sobrecarga no sistema. Segundo os dados experimentais obtidos, mesmo com taxas de entrada altas (uma métrica coletada a cada 5 segundos), a sobrecarga é bastante baixa, praticamente não influenciando nos tempos de execução das aplicações.

Por outro lado, métricas obtidas através de *benchmarks* podem causar uma sobrecarga considerável. Para tornar essa sobrecarga controlável, é recomendado que todo *benchmark* executado através do PSMS possua a mesma duração em todos os processadores em que é executado, de maneira que a sobrecarga seja igual em todos os processadores. Porém, essa é apenas uma recomendação, e não existe a obrigação de que essa regra seja seguida.

Mesmo que possua sobrecargas iguais nos processadores, um *benchmark* que execute por um tempo muito grande e tenha uma taxa de entrada alta, pode influenciar de maneira proibitiva no desempenho do sistema computacional. Por isso, é importante que os *benchmarks* não executem por muito tempo e que o período de tempo entre duas execuções consecutivas não seja demasiadamente pequeno.

O PSMS foi projetado para ser utilizado em sistemas computacionais com pouca interatividade do usuário e cuja carga de trabalho costuma variar com menor frequência. Por esse motivo, não existe a necessidade de definir altas taxas de entrada. Também, não há necessidade dos *benchmarks* possuírem longa duração.

Nos experimentos conduzidos para esta tese de doutorado, as sobrecargas experimentadas nos diversos processadores pela utilização do PSMS foi bastante baixa. Comparando-se os tempos de execução das aplicações utilizadas nos experimentos, com e sem a execução do PSMS, houve uma sobrecarga média da ordem de 1,5%, uma sobrecarga bastante aceitável analisando-se a relação custo/benefício obtida da utilização do PSMS.

Porém, é importante ressaltar que a sobrecarga causada pelo PSMS no sistema computacional é altamente dependente da configuração imposta a ele pelo administrador do sistema, e não há como fazer um cálculo exato da quantidade de sobrecarga que será gerada. Porém, em condições normais, a sobrecarga deverá ser pequena.

Considerando as funções do módulo central, que são de controle do monitor e de aplicação de funções matemáticas sobre os dados, a sobrecarga esperada é bastante pequena. Recomenda-se, porém, que o módulo central seja executado no processador mais poderoso do sistema, de preferência em um servidor isolado.

Esta seção apresentou uma descrição do protótipo implementado para investigar a viabilidade das idéias propostas neste doutorado. Utilizando esse protótipo, foram executados diversos estudos de caso, os quais são detalhados na próxima seção.

7.3 Estudos de Caso

O PSMS, conforme foi projetado, permite que sejam feitos inúmeros experimentos, através da combinação de métricas, regras de decisão, tipos de aplicações e tipos de sistemas computacionais. Abrem-se, então, diversos caminhos a serem trilhados por trabalhos futuros, que se beneficiarão da infraestrutura oferecida pelo PSMS. Não há a intenção de esgotar aqui toda as possibilidades de monitoração oferecidas pelo PSMS, por isso foram escolhidos dois exemplos da utilização do PSMS em um sistema computacional real, organizados em dois estudos de caso.

Os estudos de caso compreendem todas as fases do planejamento de uma monitoração, que vai desde a configuração do PSMS até a escolha da métrica de desempenho e das regras de decisão a serem utilizadas.

Considera-se, nos estudos de caso, que o objetivo do software de escalonamento é balancear a carga do sistema computacional. O primeiro estudo de caso é relacionado à utilização da CPU e o segundo é relacionado à utilização da memória. Como foi exposto anteriormente, esses objetivos são bastante comuns em sistemas distribuídos e a maioria dos softwares de escalonamento se baseia neles para escalonar processos. Ou seja, os estudos de caso são aplicados em uma grande variedade de situações de monitoração do escalonamento de processos feito por softwares de escalonamento de processos.

A sistemática dos estudos de caso é organizada da seguinte maneira:

- Considera-se um conjunto de aplicações paralelas a serem executadas no sistema computacional;
- As aplicações são escalonadas de acordo com um determinado algoritmo de escalonamento;
- Monitora-se o sistema computacional, através do protótipo do PSMS, utilizando uma métrica adequada;
- Calculam-se valores de desempenho através da utilização das regras de decisão;
- Os valores de desempenho coletados são mostrados graficamente, mostrando os pontos onde o PSMS detecta o problema de desempenho.

Os estudos de caso envolvem os seguintes elementos: um sistema computacional distribuído, algumas aplicações paralelas a serem executadas nesse sistema, e o protótipo do PSMS. Cada um desses elementos é descrito a seguir.

Processador	Modelo de CPU	Memória	Fator de Desempenho da CPU
P1	Pentium-S 166Mhz	32 Mb	110
P2	Pentium-II 233 Mhz	128 Mb	150
P3	Pentium-III 450 Mhz	256 Mb	280

Tabela 7.3 – Configuração do Sistema Computacional Utilizado nos Estudos de Caso

Sistema computacional considerado: Os estudos de caso foram realizados sobre uma rede de computadores pessoais heterogêneos, sob o controle do sistema operacional *Linux* (versão 2.4 de *kernel*), cuja configuração é mostrada na tabela 7.3. Para cada processador são descritos:

- Modelo de sua CPU e sua frequência de relógio;
- Quantidade de memória, medida em *Mbytes*;
- Fator de desempenho calculado através do *benchmark CPU-bound* (ver subseção 5.4.1).

Aplicações paralelas: Para que os resultados obtidos a partir dos estudos de caso representem com bastante acurácia o comportamento de um sistema computacional real, é importante que se utilize aplicações paralelas encontradas em ambientes de pesquisa e desenvolvimento reais. Neste trabalho foram utilizadas três aplicações incluídas em um pacote de *benchmarks* distribuído pela Nasa, chamado NPB (*Nasa Parallel Benchmarks*).

Um *benchmark* encontrado no NPB é definido como uma aplicação simplificada que simula o comportamento de aplicações reais utilizadas em pesquisas aeroespaciais desenvolvidas pela NASA. Essas aplicações são utilizadas em problemas relacionados à **dinâmica de fluidos computacionais**, que envolvem a solução numérica de um sistema de equações diferenciais parciais não lineares em duas ou três dimensões. As equações diferenciais parciais citadas, conhecidas como equações *Navier-Stocks*, representam as leis de conservação de massa, momento e energia aplicadas a fluidos em movimento. Para que possam ser resolvidas computacionalmente, essas equações diferenciais têm de ser discretizadas, através de métodos de diferenças finitas, de volumes finitos, de elementos finitos ou espectrais (Bayley et al., 1993; Bayley et al., 1996).

As aplicações utilizadas neste trabalho são exemplos de implementações de métodos de resolução iterativos para os sistemas lineares resultantes da discretização das equações de *Navier-Stocks*, organizados da seguinte maneira:

- **LU** : Utiliza super relaxação sucessiva simétrica (*SSOR*) na resolução de sistemas de equações lineares triangulares superiores e inferiores;
- **BT**: Resolve sistemas lineares bloco-tridiagonais com blocos de tamanho 5 X 5;
- **SP**: Resolve sistemas lineares pentadiagonais obtidos pela diagonalização total do sistema discretizado.

As três são aplicações do tipo *batch*, *CPU-bound*, que trabalham sobre conjuntos de dados de tamanhos variáveis. Essas aplicações podem ser configuradas de maneira que se controle a duração delas e a quantidade de memória que utilizam.

Configuração do protótipo do PSMS: Diversos parâmetros são configuráveis no protótipo do PSMS, como foi exposto no capítulo 5 e na seção anterior, e dois desses parâmetros influem diretamente na qualidade da monitoração, e por isso são discutidos em maiores detalhes. Esses parâmetros são: tamanho da janela da média móvel exponencial e taxa de entrada do monitor.

Como já foi exposto na seção 5.4, a média móvel exponencial representa uma média ponderada cujos pesos variam de acordo com o tamanho da janela. Quanto menor o tamanho da janela, maior serão os pesos dados às últimas métricas coletadas.

O cálculo da média móvel exponencial obtida no tempo x é feito pela equação

$$M_x = (1 - k)M_{x-1} + k.D_x \quad (7.1)$$

onde $k = 2/(1+J)$ e J é o valor da janela. O cálculo dessa média depende apenas do valor da média móvel exponencial calculada no tempo anterior ($x-1$) e da métrica medida no tempo x (D_x).

Por exemplo, suponha que se queira calcular a média móvel exponencial no tempo 7 (M_7). Até o tempo 7, foram coletadas 7 métricas ($D_7, D_6, D_5, D_4, D_3, D_2, D_1$), sendo que D_1 foi coletada antes de D_2 , D_2 antes de D_3 , e assim por diante. Ao desenvolvermos a equação 7.1, conforme apresentado no apêndice A, obtem-se uma métrica ponderada da seguinte forma:

$$M_7 = p_7.D_7 + p_6.D_6 + p_5.D_5 + p_4.D_4 + p_3.D_3 + p_2.D_2 + p_1.D_1 \quad (7.2)$$

onde $p_1, p_2, \dots, p_7$ são os pesos aplicados a cada uma das métricas usadas para calcular a média ponderada, e $p_7 + p_6 + p_5 + p_4 + p_3 + p_2 + p_1 = 1$

Os valores dos pesos variam segundo uma curva exponencial, e quanto menor o valor da janela, maior será o peso p_7 , o qual é aplicado sobre a última métrica coletada.

A tabela 7.4. apresenta os valores dos pesos aplicados às 7 métricas calculadas no exemplo, variando-se o valor da janela.

Janela Peso	2	3	4	5	6
P_7	0,6667	0,500	0,400	0,333	0,286
P_6	0,2222	0,250	0,240	0,222	0,204
P_5	0,0741	0,125	0,144	0,148	0,146
P_4	0,0247	0,063	0,085	0,099	0,104
P_3	0,0082	0,031	0,052	0,066	0,074
P_2	0,0027	0,016	0,031	0,044	0,053
P_1	0,0009	0,008	0,019	0,029	0,038

Tabela 7.4 – Pesos Definidos para Média Móvel Exponencial de Acordo com o Tamanho da Janela

Por exemplo, considerando uma janela de tamanho 3, a média exponencial (M_7) calculada no tempo 7, seria calculada através da seguinte média ponderada:

$$M_7 = 0,5.D_7 + 0,25.D_6 + 0,125.D_5 + 0,063.D_4 + 0,031.D_3 + 0,016.D_2 + 0,008.D_1 \quad (7.3)$$

A escolha de um valor adequado para a janela da média exponencial depende de dois fatores: a variabilidade da carga de trabalho de um sistema computacional e a taxa de entrada do monitor.

Se a carga de trabalho varia com muita frequência (mas não a ponto de caracterizar um sistema interativo), é interessante que a janela tenha um valor pequeno (dando maior peso às últimas métricas coletadas) e a taxa de entrada seja pequena (gerando um maior número de dados coletados) para que a média exponencial consiga se adequar rapidamente

à nova situação do sistema, possibilitando uma detecção mais rápida de um possível problema de desempenho.

Por outro lado, uma variação que ocorra como um pico, e retorne ao normal rapidamente pode erroneamente levar a conclusão de um problema de desempenho que não existe. Porém, sistemas que possuem picos de utilização frequentes caracterizam sistemas interativos, os quais não são os alvos principais de ação do PSMS.

É importante destacar que valores de janela pequenos permitem taxas de entrada maiores (ou seja, as métricas são coletadas em instantes de tempo maiores), causando menor sobrecarga no sistema. Porém, uma taxa de entrada muito grande fatalmente incorreria em demora para encontrar um problema de desempenho. A solução mais adequada, embora não seja a indicada para todos os casos, é escolher um valor de janela pequeno, o qual permite uma taxa de entrada maior e é adequado para sistemas com baixa ou alta variação da carga.

Seguindo essa lógica, nos estudos de caso, utilizou-se uma janela de tamanho 3, para que os valores das médias fossem sempre próximos ao valor da última métrica coletada, permitindo também que as aplicações paralelas utilizadas nos experimentos práticos pudessem ter uma duração menor. Como são do tipo *batch*, as aplicações têm um comportamento bastante estável, e não há a necessidade de executá-las por várias horas, como aconteceria em um sistema computacional real.

A taxa de entrada do monitor foi definida em uma observação a cada quatro minutos, e essa taxa foi definida de acordo com a duração das aplicações. Escolher uma taxa de entrada adequada depende das características do sistema computacional sendo monitorado.

A próxima subseção apresenta os resultados obtidos no primeiro estudo de caso, relacionados à otimização da utilização da CPU de um sistema distribuído.

7.3.1 Estudo de Caso nº 1 – Otimizando a Utilização da CPU

Conforme foi discutido na subseção 6.2.4, a métrica mais indicada para avaliar a otimização da utilização da CPU de um sistema computacional distribuído é a métrica *slowdown*, calculada sobre um *benchmark CPU-bound*. Por isso, nos estudos de caso

envolvendo a utilização da CPU, mostrados nesta subseção, o PSMS foi configurado para coletar a métrica *slowdown*.

Como a métrica *slowdown* não possui uma amplitude muito grande (os valores possíveis que ela pode assumir não são tão distantes entre si), utilizou-se a regra de decisão baseada no coeficiente de variação, conforme descrito na seção 5.6. Ou seja, quanto menor a distância entre os valores das métricas medidas em cada um dos processadores, mais balanceada estará a carga do sistema. Como o sistema computacional onde os testes foram feitos é heterogêneo, é importante que as métricas sejam normalizadas, para que possam ser comparadas na mesma escala. Todos os valores de *slowdown* foram normalizados através da utilização do fator de desempenho da CPU, mostrados na tabela 7.3.

Definido que vai se usar a regra de decisão baseada no coeficiente de variação, é necessário definir quais os valores de coeficiente de variação caracterizam um sistema com má distribuição de carga. A definição desses valores é configurável pelo administrador de sistema, e é possível definir níveis aceitáveis de balanceamento de carga. Através de estudos empíricos baseados na observação do comportamento de aplicações *CPU-bound* em relação ao valor da métrica *slowdown*, e em intensa experimentação prática, foram definidas três faixas para os valores de *slowdown*, mostradas na tabela 7.5.

Valor de Desempenho	Faixa de Valores do Coeficiente de Variação	Interpretação
2	0 até 0,35	Escalonamento eficiente
3	0,35 até 0,40	Escalonamento no limite de eficiência
4	0,40 em diante	Escalonamento ineficiente

Tabela 7.5 – Valores de Desempenho em Função do Coeficiente de Variação dos *Slowdowns*

É importante ressaltar que esses valores não são fixos, e podem ser modificados para sistemas computacionais que possuam maior exigência em termos de balanceamento. Porém, em todos os experimentos práticos realizados durante o desenvolvimento do PSMS, esses valores demonstraram ser bastante adequados.

O estudo de caso descrito nesta subseção é composto por dois experimentos, discutidos a seguir.

Experimento nº 1: No primeiro experimento, uma aplicação paralela (BT) composta por três processos iguais é executada em um sistema distribuído composto por três processadores, definidos na tabela 7.3 como P1, P2 e P3. Esses três processos foram escalonados de três maneiras distintas, descritas na tabela 7.6. O tempo total de execução da aplicação BT nas três distribuições também é mostrado na tabela 7.6.

	P1	P2	P3	Tempo Total de Execução
Distribuição A	1 processo	1 processo	1 processo	92 min
Distribuição B	-	1 processo	2 processos	52 min
Distribuição C	-	2 processos	1 processo	108 min

Tabela 7.6 – Distribuição de Processos - Experimento nº 1

A fim de fazer uma investigação da sobrecarga imposta pelo monitor no sistema computacional, calculou-se também o tempo de execução das aplicações sem a execução do PSMS. Conforme exposto na seção anterior, o tempo de execução da aplicação manteve-se na ordem de 1% maior.

A distribuição A é feita por um algoritmo de escalonamento do tipo *round-robin*, que caracteriza o tipo mais simples de escalonamento que pode ser feito. As outras duas distribuições são direcionadas a balancear a carga, evitando o processador P1, que possui uma capacidade inferior aos outros dois.

A monitoração das três distribuições durante o tempo resultou nos coeficientes de variação mostrados no gráfico 7.1. O eixo y mostra os coeficientes de variação (que podem variar entre 0 e 1) e o eixo x contém o tempo no qual o coeficiente foi coletado, onde cada valor da escala corresponde a um intervalo de 4 min (taxa de entrada do monitor).

Gráfico 7.1 – Coeficientes de Variação – Experimento nº 1

Quando o sistema computacional está ocioso, os coeficientes de variação têm valor 0. A aplicação é escalonada no tempo 2, e a partir do tempo 3 inicia a sua execução. No tempo 3, a distribuição A apresenta um coeficiente de variação igual a 0,55, o que representa uma má distribuição de carga. A partir do tempo 3, o PSMS já poderia alertar o software de escalonamento sobre um problema de desempenho. A distribuição C, no início de sua execução apresenta bons resultados, até porque nos instantes iniciais a carga do sistema estaria balanceada. No tempo 8, quando o processador P3 termina a sua parte na execução da aplicação, a carga do sistema se desbalanceia novamente, e no tempo 11 os coeficientes de variação da distribuição C ultrapassam o valor de 0,50, caracterizando um sistema com um mau balanceamento de carga. A distribuição B não ultrapassa o limite de 0,45 até o final da execução de BT, quando o valor do coeficiente de variação volta ao valor 0.

Pelos tempos de execução da aplicação BT mostrados na tabela 7.6, percebe-se que as distribuições A e C não apresentam bom desempenho, o que é natural considerando que elas foram feitas de maneira a não balancear a carga do sistema. Ou seja, ou o algoritmo de escalonamento não é apropriado ou algum tipo de condição anormal do sistema levou-o a fazer um escalonamento errado, que pode ser corrigido.

Se, nos tempos 4 (para a distribuição A) e 12 (para a distribuição C) fossem feitas duas migrações de processo, rebalanceando a carga do sistema, o tempo de execução da aplicação BT seria reajustado, conforme mostrado no gráfico 7.2.

O gráfico 7.2 apresenta os resultados considerando que sejam feitas migrações logo que se perceba o problema de desempenho. Também, considera-se que o tempo de migração seja muito pequeno em relação ao tempo total de execução da aplicação, o que é aceitável considerando-se que é possível implementar abordagens de migração de processos bastante eficientes (Harchol-Balter & Downey, 1997; Douglass & Ousterhout, 1991; Milojevic et al., 1993; Stellner & Trinitis, 1997; Song et al., 1997; Wibisono, 2002). Naturalmente, para processos cuja duração seja mais curta, a vantagem de utilizar migrações de processo é discutível. Porém, é importante ressaltar que não é função do PSMS fazer ou sugerir tal migração de processo. Aqui apenas quer se mostrar um exemplo de uma possível ação a ser tomada pelo software de escalonamento para corrigir o escalonamento de processos.

Gráfico 7.2 – Tempo de Execução da Aplicação BT – Experimento nº 1

Como pode ser observado no gráfico 7.2, as duas migrações de processo possibilitam um ganho de desempenho de até 50%, o que é bastante significativo em um sistema computacional onde aplicações *batch* de grande porte executam por horas.

A distribuição A, que utiliza um algoritmo de escalonamento *round-robin*, embora seja simples, é bastante eficiente se utilizada em sistemas distribuídos homogêneos

(abordagem encontrada em diversos *clusters* de computadores, por exemplo), pois evita-se a sobrecarga imposta por algoritmos de escalonamento mais sofisticados, que verificam o estado do sistema computacional antes de escalonar processos. Porém, um algoritmo de escalonamento *round-robin* não é uma boa escolha quando se trabalha com sistemas distribuídos heterogêneos, onde o escalonamento deve seguir um critério de dar preferência aos processadores mais potentes no momento de escalonar.

Softwares de escalonamento modernos utilizam algoritmos de escalonamento especializados em distribuir processos em sistemas heterogêneos, fazendo um escalonamento que leva em consideração a potência computacional dos processadores e considerando índices de carga que garantem o balanceamento de carga. Um bom algoritmo de escalonamento teria então feito uma distribuição correta da aplicação BT neste primeiro experimento.

Generalizando, grande parte dos algoritmos de escalonamento modernos acertariam, pelo menos nos instantes iniciais, o escalonamento de aplicações *CPU-bound*, de maneira que a carga do sistema computacional se mantivesse balanceada. O problema de desempenho ocorreria se houvesse uma mudança na carga de trabalho, conforme é discutido no próximo experimento.

Experimento nº 2: Este segundo experimento mantém a mesma estrutura do primeiro experimento, ou seja, o software de escalonamento tem como objetivo balancear a carga do sistema com ênfase na utilização da CPU e utiliza-se a métrica *slowdown* em conjunto com a regra de decisão dos coeficientes de variação.

Neste experimento, porém, considera-se que o algoritmo de escalonamento foi projetado de maneira a escalonar processos balanceando a carga do sistema, considerando a capacidade de cada processador e utilizando como índice de carga o tamanho da fila da CPU. Duas aplicações paralelas são executadas no sistema computacional (composto novamente por P1, P2 e P3, descritos na tabela 7.3), BT e LU, cada uma delas composta por três processos. Portanto, são escalonados seis processos.

Supondo que ambas aplicações sejam escalonadas ao mesmo tempo, o algoritmo de escalonamento faria a seguinte distribuição: 3 processos em P3, 2 processos em P2 e 1 processo em P1. Como as aplicações chegam ao sistema ao mesmo tempo, existem 6

possíveis distribuições de processos, todas com igual probabilidade de acontecer. As distribuições são mostradas na tabela 7.7. O tempo total de execução das duas aplicações em cada uma das distribuições também é mostrado na tabela 7.7.

	P1	P2	P3	Tempo Total de Execução
Distribuição A	1 LU	2 LU	3 BT	39 min
Distribuição B	1 LU	1 BT e 1 LU	2 BT e 1 LU	38 min
Distribuição C	1 BT	2 LU	2 BT e 1 LU	46 min
Distribuição D	1 LU	2 BT	1 BT e 2 LU	54 min
Distribuição E	1 BT	1 BT e 1 LU	1 BT e 2 LU	46 min
Distribuição F	1 BT	2 BT	3 LU	54 min

Tabela 7.7 Distribuição de Processos - Experimento nº 2

A monitoração das 6 distribuições é mostrada no gráfico 7.3. O eixo y contém os valores de coeficiente de variação (que podem variar entre 0 e 1) e o eixo x contém o tempo no qual o coeficiente foi coletado, onde cada valor da escala corresponde a um intervalo de 4 min.

Como pode ser observado no gráfico 7.3, todas as distribuições apresentam bom desempenho até aproximadamente o tempo 5, quando FT termina e ocorre um desbalanceamento na maior parte das distribuições. Somente as distribuições A e B mantêm bom desempenho, as outras 4 degradam o desempenho do sistema. Tais resultados já eram esperados, em virtude dos tempos de execução mostrados na tabela 7.7. O tempo total de execução apresentado nas distribuições D e F é pelo menos 30% maior do que o tempo da distribuição B, a mais eficiente de todas as distribuições. As distribuições C e E executam as aplicações em um tempo 21% maior do que B.

Gráfico 7.3 - Coeficientes de Variação - Experimento nº 2

No tempo 7, os valores do coeficiente de variação de D, E e F já apresentam sinais de mau desempenho. No tempo 9, C também assume um valor de coeficiente de variação acima do limite aceitável. Se, novamente, no momento quando o PSMS descobre o problema de desempenho são feitas migrações de processo, as aplicações podem ter melhor desempenho, conforme apresentado na tabela 7.4.

Gráfico 7.4 - Tempo de Execução das Aplicações BT e LU - Experimento nº 2

Como foi exposto nos dois experimentos deste estudo de caso, um software de escalonamento que possua como objetivo balancear cargas nem sempre fará o melhor

escalonamento. Pode-se escolher um algoritmo de escalonamento que não seja adequado, como mostrado no experimento nº 1, e mesmo um algoritmo de escalonamento adequado não pode garantir que o desempenho seja sempre eficiente, conforme mostrado no experimento nº 2. Para ambos casos, a utilização do PSMS permitiu a detecção do problema de desempenho, o que permitiria ao software de escalonamento, corrigir o mau escalonamento, através de migrações de processos.

Nesta subseção foram apresentados dois experimentos relacionados ao primeiro estudo de caso, onde se analisaram algoritmos de escalonamento que possuam como objetivo balancear cargas com ênfase na utilização da CPU. A próxima subseção apresenta um estudo de caso relacionado à memória.

7.3.2 Estudo de Caso nº 2 – Otimizando a Utilização da Memória

O segundo estudo de caso relaciona-se a otimizar a utilização da memória, ou seja, balancear a carga do sistema considerando a utilização de memória. É importante ressaltar que o objetivo desta subseção é mostrar um exemplo da utilização do PSMS, e não se tem a intenção de fazer uma discussão aprofundada sobre a adequação das métricas relacionadas à memória. Esse assunto é bastante complexo para ser abordado com profundidade em apenas um estudo de caso.

Conforme foi discutido na subseção 6.2.5, existem diferentes pontos de vista sobre como escalonar processos considerando a utilização da memória. Por exemplo, Kunz (1991) e Voourluys et al. (2003) argumentam que a distribuição de processos deve ser feita de maneira que a quantidade de memória utilizada nos diversos processadores seja balanceada. Por outro lado, Giné et al. (2000) e Xiao et al. (2003) definem que é importante que a quantidade de paginação seja minimizada.

Porém, antes mesmo de balancear a carga do sistema, é importante garantir que não ocorra *swap* em nenhum dos processadores disponíveis, pois existe uma grande perda de desempenho pela intensa utilização do disco. Por isso, é importante que o algoritmo de escalonamento, independente de sua estrutura, evite o *swap*.

Garantindo que não exista *swap*, o algoritmo de escalonamento pode então controlar a quantidade de paginação do sistema ou manter o percentual de utilização da

memória balanceada. Porém, na prática, os valores de paginação de memória retornados pelo sistema operacional utilizado no estudo de caso (no caso, o *Linux*) não seguem um padrão lógico, impossibilitando uma análise sistemática desses valores.

Por exemplo, a aplicação SP ocupa praticamente toda a memória de um processador que possua 128 Mb de memória, mas mesmo assim os valores de paginação mantêm-se semelhantes aos coletados no sistema disponível. Alguns picos de utilização ocorrem, mas não existe uma lógica aparente nesses picos. Por isso, embora essa métrica seja bastante citada e utilizada em trabalhos científicos, sua utilização esbarra na impossibilidade prática de interpretar os valores coletados no sistema computacional considerado.

Portanto, neste estudo de caso, considera-se que o objetivo do software de escalonamento é otimizar a utilização da memória em termos de quantidade utilizada, e emprega-se a métrica percentual de utilização da memória. Neste estudo de caso, escolheu-se avaliar a utilização da memória em termos de percentual, mas poderia-se avaliar a utilização da memória em termos de quantidade de bytes disponíveis. Ambas métricas podem ser utilizadas, e aqui se tem a intenção de mostrar um exemplo da utilização do PSMS, não significando que a métrica escolhida seja a melhor. Além disso, avaliar a utilização da memória em termos de quantidade de bytes utilizados requer uma abordagem diferente de interpretação.

Também é importante ressaltar que objetivos que sejam relacionados ao balanceamento do uso da memória não tem relação direta com tempos de execução melhores. Apenas procura-se evitar que a memória fique sobrecarregada, de maneira que se diminua a paginação do sistema e evite-se a ocorrência de *swap*.

Para avaliar o desempenho do sistema baseado nesse objetivo é utilizada a regra de decisão que contabiliza a variação de frequências entre classes de métricas, conforme discutido na seção 5.6.

São consideradas então quatro classes, definidas pelos seguintes valores:

- Classe I - Métrica entre 0% e 25%
- Classe II - Métrica entre 26% e 50%
- Classe III - Métrica entre 51% e 75%
- Classe IV - Métrica entre 76% e 100%

Todo processador cuja utilização de memória esteja entre 0% e 25%, por exemplo, é contabilizado na classe 1. O processador cuja utilização esteja entre 51% e 75%, é contabilizado na classe 3. E assim por diante.

Para cada classe, calcula-se a frequência de métricas. Por exemplo, se uma determinada classe contiver 9 de 10 métricas medidas, a frequência dessa classe será 0.9. Ou seja, 90% das métricas medidas estão nessa classe. O valor de desempenho é calculado sobre essas classes, de acordo com a tabela 7.8.

Valor de Desempenho	Faixa de Valores Para a Frequência de Classes	Interpretação
2	Se frequência de duas classes consecutivas for maior do que 0,8	Escalonamento eficiente
3	Se frequência de duas classes consecutivas estiver entre 0,8 e 0,7	Escalonamento no limite de eficiência
4	Se frequência de duas classes consecutivas for menor que 0,7	Escalonamento ineficiente

Tabela 7.8. Interpretação dos Valores de Frequência de Classes

Conforme é mostrado na tabela 7.8, através da utilização dessa regra de decisão, procura-se manter a maior parte dos processadores com um percentual de utilização semelhante, evitando que existam processadores que estejam com a sua memória totalmente ocupada enquanto existam processadores com a sua memória livre.

Os valores apresentados na tabela 7.8 foram definidos de acordo com o tipo de sistema computacional utilizado no estudo de caso, e podem ser redefinidos para garantir distribuições com maior exigência em termos do balanceamento da utilização da memória.

Para exemplificar a aplicação dessa regra de decisão, considera-se um sistema computacional composto por três processadores quase homogêneos, cuja configuração é descrita na tabela 7.9.

Processador	Modelo de CPU	Memória
P1	Pentium-II 250 Mhz	128 Mb
P2	Pentium-II 250 Mhz	128 Mb
P3	Pentium-II 250 Mhz	256 Mb

Tabela 7.9 – Configuração do Sistema Computacional

Considere um algoritmo de escalonamento que tenha sido projetado para escalonar processos de acordo com a quantidade de memória disponível, e se queira executar duas aplicações paralelas: SP (composta por três processos) e LU (composta por dois processos). A aplicação SP é escalonada no tempo 1, e ocupa a maior parte da memória de P1 e P2, mas não de P3. No tempo 14, a aplicação LU é escalonada, e como apenas P3 possui espaço disponível de memória, os dois processos componentes de LU são escalonados em P3. Os dados coletados no sistema nas 34 coletas iniciais do PSMS, para cada processador, são mostrados no gráfico 7.5.

Gráfico 7.5 – Utilização da Memória – Estudo de Caso nº 2

Considerando os valores de desempenho definidos na tabela 7.8, o sistema estaria com utilização da memória uniforme até a coleta feita no tempo 20, onde haveria um desbalanceamento do uso da memória, pois existe um processador com sua memória sobrecarregada (P3) enquanto outros dois estão com a memória livre (P1 e P2). Portanto, o PSMS nesse ponto retornaria um valor de desempenho 4, mostrando um problema de desempenho no sistema computacional.

Nesse exemplo em particular, monitorou-se o sistema utilizando a mesma configuração apresentada no estudo de caso anterior, ou seja, utilizou-se a métrica *sl_vdown* com uma regra de decisão baseada nos coeficientes de variação. Os resultados foram basicamente os mesmos, ou seja, com ambas abordagens de monitoração, o PSMS detectou um problema de desempenho no mesmo instante.

Isso acontece em grande parte dos sistemas computacionais, visto que o tamanho da memória disponível geralmente é proporcional à qualidade da CPU. Ou seja, é muito pouco provável que se encontre uma CPU modelo *Pentium* sendo utilizada em um computador que possua 128 Mbytes de memória, porque na época que ele foi desenvolvido, esse tamanho de memória não era comum.

Cada época caracteriza-se por uma determinada configuração de hardware. Portanto, um computador que possua 128 Mbytes de memória, provavelmente possuirá uma CPU mais moderna, como por exemplo, um *Pentium II*. Não é possível generalizar essa regra, mas baseado nela, pode-se definir que quando há dúvidas sobre qual métrica utilizar, deve-se dar preferência às métricas que são relacionadas à CPU.

Concluindo esta seção, é importante frisar que aqui foram apenas apresentados alguns exemplos de situações onde o PSMS pode ser útil. Existem diversas situações possíveis onde a abordagem de monitoração aqui proposta pode oferecer resultados úteis, conforme é discutido no próximo capítulo.

Pelos experimentos mostrados, porém, percebe-se com clareza que um escalonamento mal feito, fatalmente incorre em perda de desempenho. A escolha de um algoritmo de escalonamento correto é muito importante, mas, como foi mostrado nos estudos de caso, mudanças na carga de trabalho podem causar problemas de desempenho mesmo quando se utilizam algoritmos de escalonamento eficientes.

7.4 Considerações Finais

Através dos exemplos mostrados neste capítulo, pode-se concluir que a principal função do PSMS é descobrir problemas de desempenho, sendo que os problemas de desempenho são definidos de acordo com o objetivo do escalonamento de processos.

Assim que o PSMS descobre um problema de desempenho, ele informa ao software de escalonamento, e o tipo de ação que deve ser tomada para corrigir esse problema fica a cargo do software de escalonamento. Em caso de problema de desempenho, basicamente o software de escalonamento pode tomar três atitudes:

- Trocar o algoritmo de escalonamento;
- Reconfigurar um algoritmo de escalonamento adaptável;
- Fazer migrações de processos.

Diferentes cenários podem ser citados que exemplificam essa relação entre software de escalonamento e monitor. Entre eles:

Cenário 1: O software de escalonamento inicia suas atividades, e logo nas primeiras coletas os valores de desempenho são desfavoráveis. Nesse caso, provavelmente o algoritmo de escalonamento não foi bem escolhido, e uma possível solução para o problema é a troca de algoritmo de escalonamento. Um exemplo desse problema é apresentado no experimento nº 1, do estudo de caso nº 1. O algoritmo *round-robin*, no exemplo, já mostra problemas de desempenho logo nas primeiras coletas, sinal de que o algoritmo é inadequado para o objetivo proposto.

Cenário 2: Se os valores de desempenho apresentam problemas depois de várias coletas, existe uma grande probabilidade do algoritmo ser adequado, e o problema de desempenho ter sido gerado por uma modificação do perfil da carga de trabalho. Nesse caso, duas abordagens de correção são possíveis: utilizar um algoritmo que seja mais adequado à carga de trabalho ou corrigir o desempenho do sistema através de migrações de processos. Um exemplo dessa segunda abordagem é mostrado no experimento nº 2 e no estudo de caso nº 2.

Mesmo que o software de escalonamento não seja flexível a ponto de poder modificar o seu algoritmo de escalonamento, os valores de desempenho retornados pelo PSMS podem ser utilizados com outras funções.

Por exemplo, considere que exista um limite máximo de carga que um determinado processador possa suportar. Nesse caso, quando esse limite for alcançado, o software de escalonamento é alertado, e pode descobrir os processadores sobrecarregados através de uma consulta ao histórico do sistema. Ou seja, não existe a necessidade de verificar a carga do sistema computacional antes de cada escalonamento, sendo necessário apenas esperar que o PSMS avise quando um nível de carga acima de um determinado limite tenha sido alcançado.

Independente da resposta do software de escalonamento ao alerta do PSMS, para que esse alerta seja correto, é muito importante que se escolha criteriosamente a métrica a ser utilizada.

Como foi exposto no capítulo anterior, em muitos casos a métrica tem relação direta com o objetivo, mas nem sempre isso ocorre. Por exemplo, determinadas métricas são de difícil análise quando coletadas em determinados sistemas operacionais, como por exemplo a quantidade de paginação de um processador que execute o sistema operacional *Linux*, conforme discutido no estudo de caso nº 2. Geralmente, através de substituições adequadas pode-se resolver o problema. Uma métrica que permite aproximar o valor de paginações de um processador é a métrica que calcula o percentual de utilização da memória. Quanto menor a quantidade de memória utilizada, menor será a paginação.

Em um caso onde existe dúvida sobre qual métrica utilizar para avaliar um determinado objetivo de escalonamento, métricas que podem ser utilizadas com maior probabilidade de acerto são métricas relacionadas à CPU. Geralmente, processadores evoluem de maneira gradual, e aqueles que possuem as melhores CPUs possuem também os outros recursos (memória, disco e interfaces de rede) de boa qualidade. Essa característica pode ser verificada no estudo de caso nº 2.

Além disso, conforme foi concluído no capítulo anterior, a métrica que tem a maior probabilidade de oferecer resultados satisfatórios é o *slowdown*. Portanto, pode-se concluir

que, em caso de dúvida, deve-se utilizar a métrica *slowdown*, calculada sobre o *benchmark CPU-bound*.

O PSMS, na sua versão atual, já pode ser utilizado para uma grande variedade de situações de monitoração do escalonamento, e vários aperfeiçoamentos estão previstos para trabalhos futuros, que permitirão uma utilização muito mais abrangente. Assim, dá-se um passo significativo ao aprimoramento da atividade de escalonar processos, permitindo que os sistemas distribuídos sejam utilizados com bom desempenho a maior parte do tempo.

8 Conclusões

Este capítulo apresenta as ponderações finais desta tese, incluindo as suas principais contribuições e sugestões de trabalhos futuros.

8.1 *Ponderações Finais Desta Tese*

A computação é uma ciência que vive em constante metamorfose, em virtude principalmente da incomparável evolução tecnológica que as últimas décadas do século 20 experimentaram. Os conceitos relacionados aos sistemas computacionais têm se modernizado para que se adaptem a uma das palavras chaves da computação moderna: flexibilidade.

Para a computação distribuída, que define o universo de aplicação deste trabalho, flexibilidade é um ponto chave. E a flexibilidade de um sistema distribuído é dependente, em grande parte, de sua abordagem de escalonamento de processos.

Por isso, a pesquisa em escalonamento de processos tem se direcionado à criação de algoritmos de escalonamento flexíveis que possam se adaptar às características da carga de trabalho de um sistema distribuído. Softwares de escalonamento, por sua vez, têm sido projetados de maneira que possam utilizar diferentes algoritmos de escalonamento, de acordo com as necessidades de um determinado sistema computacional distribuído.

Se o escalonamento de processos é flexível, é importante que se possa fazer uso dessa flexibilidade para que o sistema distribuído sempre mantenha um bom nível de desempenho, garantindo que na maior parte do tempo os usuários estejam satisfeitos. Dentro deste contexto, esta tese propôs um software de monitoração que tem a função de avaliar o desempenho do sistema distribuído sob o ponto de vista do escalonamento de processos, de maneira que o elemento que seja responsável pelo escalonamento (uma pessoa, um software ou o próprio sistema operacional) possa avaliar se o sistema distribuído está respondendo bem ao algoritmo de escalonamento utilizado.

Grande parte da pesquisa em escalonamento de processos tem sido direcionada a pesquisar técnicas de avaliação de desempenho que auxiliem ao escalonamento de

processos, conforme discutido nos capítulos 3 e 4. Esta tese de doutorado aborda o problema de avaliação do escalonamento considerando as seguintes diretrizes:

- Utiliza-se a técnica de monitoração, pois os dados são coletados diretamente no sistema computacional;
- As métricas de desempenho são avaliadas no momento em que são coletadas, caracterizando uma monitoração *on-line*;
- A métrica de desempenho é escolhida de acordo com o objetivo do escalonamento.

A idéia de utilizar monitoração *on-line* para melhorar o desempenho do escalonamento de processos não é nova, e foi explorada por renomados grupos de pesquisa mundiais. NWS e Autopilot são exemplos de softwares que fazem tal tarefa. Porém, a abordagem de monitoração que esses softwares oferecem não é adequada à abordagem proposta nesta tese de doutorado, que se propõe a oferecer uma infraestrutura de monitoração que seja de propósito geral, ou seja, possa ser utilizada por qualquer software de escalonamento e possa ser utilizada para avaliar escalonamentos com diferentes objetivos.

Por isso, o software de monitoração implementado durante o desenvolvimento desta tese, que foi nomeado PSMS (*Process Scheduling Monitoring System*) e que é apresentado nos capítulos 4 e 5, foi projetado para que seja altamente flexível e que possa ser estendido pela inclusão de novos módulos.

Essa flexibilidade oferecida pelo PSMS estendeu, de certo modo, os objetivos iniciais do PSMS. Enquanto que, inicialmente, desejava-se projetar um software de monitoração flexível que pudesse ser utilizado por softwares de escalonamento, ele também pode ser utilizado como ferramenta auxiliar no desenvolvimento de novos algoritmos de escalonamento por pesquisadores da área e na pesquisa de métricas de desempenho. Neste sentido, os resultados obtidos por esta tese de doutorado são úteis tanto em nível prático quanto teórico, sendo uma ferramenta útil no estreitamento da relação entre o que se pesquisa em computação (teoria) e o que é efetivamente implementado em softwares de escalonamento (prática).

Independente do nível que se use o PSMS, quando se faz uma avaliação de desempenho, é muito importante que sejam utilizadas métricas adequadas. Por esse motivo, esta tese teve como preocupação primária fazer uma análise bastante abrangente da utilização de métricas no escalonamento de processos. Não houve em nenhum momento a intenção de fazer um trabalho definitivo sobre esse assunto, mas iniciar uma discussão aprofundada que possibilite que novos trabalhos possam se dedicar a esse tema, pois essa é uma área bastante promissora de pesquisa. Também, espera-se que a própria existência de um software de monitoração como o PSMS seja um grande incentivo ao surgimento de trabalhos especificamente direcionados a essa vertente de pesquisa em escalonamento de processos.

Não só a escolha das métricas é importante, mas também a maneira de interpretá-las (ou seja, tirar conclusões a partir de seus valores) é de fundamental importância. Por esse motivo, o PSMS foi projetado para permitir a interpretação dos dados coletados através de regras de decisão. A flexibilidade, fator chave da estrutura do PSMS, também se aplica às regras de decisão, de maneira que novas regras podem ser anexadas ao software de monitoração. Também sob esse aspecto, os resultados e contribuições desta tese não têm a intenção de esgotar o assunto, e sim facilitar e incentivar a definição de trabalhos futuros que venham complementar os resultados aqui obtidos.

O capítulo 6 apresentou alguns estudos de caso que mostram a viabilidade da utilização da infraestrutura definida pelo PSMS. Os experimentos discutidos nesse capítulo mostram que é possível que um algoritmo de escalonamento, mesmo quando concebido visando um determinado objetivo, pode ser prejudicado por diversos fatores, como por exemplo, uma mudança na carga de trabalho.

Essa mudança pode ser ocasionada por processos que não foram escalonados pelo software de escalonamento (considerando a hipótese de ser possível um usuário executar um processo sem utilizar o software de escalonamento) e sob a qual ele não tem controle, ou mesmo por processos escalonados pelo próprio software de escalonamento. Isso é possível quando não existe a possibilidade de migração de processos, ou seja, o software de escalonamento não procura reajustar o desempenho do sistema quando determinada aplicação termina ou quando a carga imposta por uma aplicação não é a que foi prevista.

Independente do motivo que venha a causar o problema de desempenho, as regras de decisão propostas nesta tese permitiram ao PSMS detectar o problema, de maneira que o software de escalonamento pudesse corrigir o seu desempenho.

Os estudos de caso apresentados nesta tese são baseados em algoritmos de escalonamento comumente usados por softwares de escalonamento reais, de maneira que uma implementação definitiva do PSMS poderá ser disponibilizada para utilização por laboratórios de pesquisa em todo o mundo. Portanto, além da importância teórica desta tese (pela discussão das métricas, regras de decisão e diretrizes de monitoração), a implementação de um protótipo também permite a sua utilização prática, reforçando o caráter de união entre a teoria e a prática que guiou o desenvolvimento desta tese.

Na próxima seção são apresentadas as principais contribuições desta tese de doutorado.

8.2 Contribuições Desta Tese

As principais contribuições desta tese de doutorado são:

- Proposta de um monitor *on-line* e de propósito geral para a avaliação do desempenho do escalonamento de processos, baseado na coleta de métricas de desempenho em um sistema distribuído e da aplicação de regras de decisão sobre essas métricas, permitindo tirar conclusões sobre o desempenho do sistema;
- Análise crítica das métricas de desempenho utilizadas para avaliar o escalonamento de processos, estudando a relação entre métricas e objetivos do escalonamento e discutindo a viabilidade de utilizá-las para uma monitoração *on-line*;
- Proposta de regras de decisão para a interpretação das métricas de desempenho coletadas no sistema. As regras de decisão retornam um valor de desempenho, que é um valor que varia entre 1 e 5 e quantifica o desempenho do sistema computacional;

- Análise do estado da arte da utilização de monitores para auxiliar o escalonamento de processos, através da apresentação de exemplos e também da discussão de softwares de monitoração que possuam objetivos semelhantes aos do PSMS;
- Implementação de um protótipo, que contém grande parte do software proposto, mostrando a sua viabilidade e permitindo a experimentação prática dos conceitos propostos nesta tese. Quando o software estiver pronto, será disponibilizado para *download* pela comunidade acadêmica mundial;
- Apresentação de estudos de caso onde são discutidos exemplos da utilização das métricas de desempenho e respectivas regras de decisão, com sugestões de valores possíveis de configuração das regras;
- Análise bibliográfica abrangente mostrando conceitos básicos de escalonamento de processos e avaliação de desempenho, discutindo a relação que existe entre as duas áreas de pesquisa.

8.3 Propostas de Trabalhos Futuros

Devido a grande flexibilidade do software de monitoração aqui proposto, ele pode ser utilizado para a investigação de métricas de desempenho aplicadas ao escalonamento de processos, área de pesquisa pouco explorada, e também para o desenvolvimento de novos algoritmos de escalonamento. Exemplos de trabalhos futuros relacionados a esta tese de doutorado são:

- Estudo de métricas de desempenho para a avaliação de algoritmos de escalonamento com diversos objetivos, estendendo os resultados obtidos nos estudos de caso em relação à CPU e à memória;
- Estudo da monitoração aplicada a disco e à interface de rede, através da proposta de métricas de desempenho e regras de decisão adequadas;
- Aperfeiçoamento das regras de decisão definidas nesta tese e proposta de novas regras utilizando técnicas estatísticas e também utilizando técnicas de inteligência artificial, como lógica nebulosa e sistemas especialistas;

- Implementação completa do PSMS, incluindo aí uma interface gráfica para a configuração dos parâmetros de monitoração;
- Comparação de softwares de escalonamento de domínio público, como por exemplo o *Sun Grid Engine*, *PBS* e *AMIGO*, utilizando a infraestrutura de monitoração oferecida pelo PSMS;
- Extensão do PSMS para a sua utilização na computação em *Grid*, estudando métricas de desempenho e regras de decisão adequadas, e também a modificação de sua implementação para se adaptar a essa nova função;
- Implementação do PSMS para outros sistemas operacionais e também para arquiteturas paralelas.

9 Referências Bibliográficas

- ANDERSON, T.E.; CULLER, D.E.; PATTERSON, D.A. A Case for NOW (Networks of Workstations). *IEEE Micro*, pp. 54-64, fevereiro, 1995.
- ARAÚJO, A.P.F., SANTANA, M.J., SANTANA, R.H.C., SOUZA, P.S.L., DPWP - A New Load Balancing Algorithm. *5th International Conference on Information Systems Analysis and Synthesis - ISAS'99*, agosto, 1999.
- ARPACI-DUSSEAU, A.C.; CULLER, D.E; MAINWARING, A.M. Scheduling with Implicit Information in Distributed Systems. *Proceedings of ACM SIGMETRICS'98*, pp. 233-243, 1998.
- BAKER, M.A.; FOX, G.C.; YAU, H.W. Review of Cluster Management Software. *NHSE Review*, First Issue, <http://nhse.cs.rice.edu/NHSEreview>, 1996.
- BAKER, M.A.; BUYYA, R. Cluster Computing at a Glance, in: BUYYA, R., ed. *High Performance Cluster Computing*. vol. 1, cap. 1, Prentice Hall, maio, 1999.
- BASNEY J.; LIVNY, M. Deploying a High Throughput Computing Cluster, in: BUYYA, R., ed. *High Performance Cluster Computing*. vol. 1, cap. 5, Prentice Hall, maio, 1999.
- BATAT, A.; FEITELSON, D. Gang scheduling with memory considerations. *Proceedings of the 14th Intl. Parallel Distributed Processing Symposium*, pp. 109-114, 2000.
- BAUMGARTNER, K.M.; WAH, B. W. Computer Scheduling Algorithms: Past, Present and Future. *Information Sciences, Elsevier Science Pub. Co., Inc.*, vol.57 & 58, p.319-345, setembro/dezembro, 1991.
- BAYLEY, D.; HARRIS, T.; SAPHIR, W.; WIJNGAART, R.; WOO, A.; YARROW, M. The NAS Parallel Benchmarks 2.0. *NAS Technical Report NAS-95-029*, dezembro, 1996.
- BAILEY, D.; BARTON, J.; LASINSKI, T.; SIMON, H. The NAS Parallel Benchmarks. NASA. Technical Memorandum 103863, CA 94035-1000, Julho, 1993.

- BEGUELIN, A.; GEIST, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAM, V. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*, The MIT Press, 1994.
- BERMAN(a), F.; WOLSKI, R.; CASANOVA, H.; CIRNE, W.; DAIL, H.; FAERMAN, M.; FIGUEIRA, S.; HAYES, J.; OBERTELLI, G.; SCHOPF, J.; SHAO, G.; SMALLEN, S.; SPRING, S.; SU, A.; ZAGORODNOV, D. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4), 369--382, 2003.
- BERMAN(b), F.; FOX, G.; HEY, T. The Grid: Past, Present, Future. in BERMAN, F.; FOX, G.; HEY, T. *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, 2003.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do Usuário*. Editora Campus, 2000.
- BRANCO, K.R.L.J.C. *Índice de Carga e Desempenho em Ambientes Paralelos/Distribuídos - Modelos e Métricas*. Trabalho de doutorado em andamento no ICMC-USP, São Carlos, 2000.
- BUYYA, R.; ABRAMSON, D.; GIDDY, J. PARMON: A comprehensive cluster monitoring system. *Proceedings of the AUUG'98 Conference*, 1998.
- BUYYA, R.; MURSHED, N. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practical and Experience*. vol. 14, 2002.
- 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.
- CHAPIN, J.; CIRNE, W.; FEITELSON, Dror; JONES, J.; LEUTENEGGER, S.; SCHIWIEGELSHOHN, U.; SMITH, W.; TALBY, D. Benchmarks and Standards for the Evaluation of Parallel Job Schedulers, in. FEITELSON D.; RUDOLPH, L. eds. *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, pp. 67-90, 1999.

- CHENG; DANDAMUDI. Scheduling in Parallel Systems with a Hierarchical Organization of Tasks. *Proceedings of the 1992 international conference on Supercomputing*, pp. 377 – 386, 1992.
- CLARK, D.L.; CASAS, J.; OTTO, S.W.; PROUTY, R.M.; WALPOLE, J. Scheduling of Parallel Jobs on Dynamic, Heterogeneous Networks. janeiro, 1995. Disponível em: <http://www.cse.ogi.edu/DISC/projects/cpe/>
- COMER, D. E. *Inteligência em Rede com TCP/IP - vol. 1*, Editora Campus, 1998.
- CONDOR TEAM. *Condor Version 6.6.1 Manual*, University of Wisconsin-Madison, 2004.
- COULOURIS, G., DOLLIMORE, J., KINDBERG, T. *Distributed Systems: Concepts and Design*. 3.ed., Addison-Wesley, 2001.
- CRAY RESEARCH, INC. *Introducing NQE*. 2000.
- DANTAS, M.A.R.; ZALUSKA, E.J. Efficient Scheduling of MPI applications on Networks of Workstations. *Future Generation Computer Systems - FGCS*, vol.13, p.489-499, 1998.
- DEROSE, L.; ZHANG, Y.; REED, D., SvPablo: A Multi-Language Performance Analysis System. *Computer Performance Evaluation - Modelling Techniques and Tools*. Springer-Verlag, (10th International Conference - Performance Tools'98), pp. 352-355, setembro, 1998.
- DEVARAKONDA, M.V.; IYER, R.K. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Transactions on Software Engineering*, vol.15, n.12, p.1579-1586, dezembro, 1989.
- DOUGLIS, F.; OUSTERHOUT, J. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software - Practice and Experience*, vol.21, n.8, p.757-785, agosto, 1991.
- ESHAGHIAN, M.M. *Heterogeneous Computing*. Artech House Computer Science Library, 1996.

- FEITELSON, D.G.; RUDOLPH, L. Parallel Job Scheduling: Issues and Approaches. *IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science vol.949*, Santa Barbara, CA, USA, abril, 1995.
- FEITELSON, D.G.; RUDOLPH, L. Toward Convergence in Job Schedulers for Parallel Supercomputers. *IPPS' 96 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science vol.1162*, Santa Barbara, Honolulu, Hawaii, abril, 1996.
- FEITELSON, D.G.; RUDOLPH, L. Metrics and Benchmarking for Parallel Job Scheduling. *ISSPP'98, Lecture Notes in Computer Science*, n. 1459, p. 1-24, 1998.
- FEITELSON, D. Metrics for parallel job scheduling and their convergence. *Lecture Notes in Computer Science*, vol. 2221, pp. 188-206, 2001.
- FEITELSON, D. Metric and workload effects on computer systems evaluation. *Computer*, 36(9), pp. 18-25, Setembro, 2003.
- FERRARI, D.; ZHOU, S. An Empirical Investigation of Load Indices for Load Balancing Applications. *Proceedings of Performance'87, the 12th Int'l Symposium on Computer Performance Modeling, Measurement, and Evaluation*, p.515-528, 1987.
- FERSTL, F. Job-and resource-management systems in heterogeneous clusters. *Future Generation Computer Systems - FGCS*, 12, p.39-51, 1996.
- FRACHTENBERG, E.; FEITELSON, D.; PETRINI, F.; FERNANDEZ, F. Flexible CoScheduling: mitigating load imbalance and improving utilization of heterogeneous resources. *Proceedings of the 17th Intl. Parallel and Distributed Processing Symposium*, Abril, 2003.
- FRANCÊS, C. R. L. *Stochastic Feature Charts - Uma Extensão Estocástica para os Statecharts*. Dissertação (Mestrado). ICMC-USP, São Carlos, maio, 1998.
- GIBBONS, R. A Historical Application Profiler for Use by Parallel Schedulers. *IPPS' 97 Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, vol.1291, abril, 1997.

- GINÉ, F.; SOLSONA, F.; HERNANDEZ, P.; LUQUE, E. Coscheduling under memory constraints in a NOW environment. *Lecture Notes on Computer Science*, vol. 2221, 2001.
- HARCHOL-BALTER, M.; DOWNEY, A. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems* vol.15, n.3, agosto, 1997.
- HAREL, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, n. 8, pp. 231-274, North-Holland, 1987.
- HLAVACS, H.; UEBERHUBER, C.W. Simulating Load Balancing on Heterogeneous Workstation Clusters. *Lecture Notes on Computer Science*, vol. 1557, fevereiro, 1999.
- HOCKNEY, R. W. *The Science of Computer Benchmarking*. Philadelphia: Society for Industrial and Applied Mathematics, 1996.
- HOFMANN, R.; KLAR, R.; MOHR, B.; QUICK, A.; SILGLE, M. Distributed Performance Monitoring: Methods, Tools, and Applications. *IEEE Transactions on Parallel Distributed Systems*, vol. 5, n. 6, p 585-598, junho, 1994.
- JAIN, R. *The art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Inc, 1991.
- JONES, J. P. Evaluation of Job Queuing/Scheduling Software: Phase 1 Report. *NAS Technical Report NAS-96-009*. julho, 1996.
- JOYCE, J.; LOMOW, G.; SLIND, K.; UNGER, B. Monitoring Distributed Systems. *ACM Transactions on Computer Systems*, vol. 5(2), p. 121-150, 1987.
- KANT, K. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.
- KAPLAN, J.A.; NELSON, M.L. A comparison of Queuing, Cluster and Distributed Computing Systems. *Nasa Langley Research Center*, junho, 1994.

- KRALLMANN, J.; SCHWIGELSHOHN, U. On the design and evaluation of job scheduling algorithms. in. FEITELSON D.; RUDOLPH, L. eds. *Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, pp. 17-42, 1999.
- KRONE, O.; RAAB, M.; HIRSBRUNNER, B. Load Balancing for Network Based Multithreaded Applications. *5th European PVM/MPI User's Group Meeting, Lecture Notes in Computer Science vol.1497*, p.206-214, setembro, 1998.
- KUNZ, T. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. *IEEE Transactions on Software Engineering*, vol.17, n.7, p.725-730, julho, 1991.
- LIVNEY, M.; MELMAN, M. Load Balancing in Homogeneous Broadcast Distributed Systems. *Proceedings of the ACM Computer Network Performance Symposium*, pp. 47-55, 1982.
- 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.
- MACDOUGALL, M. H. *Simulating Computing Systems Techniques and Tools*. MIT, 1987.
- MACHADO, F.B.; MAIA, L.P. *Introdução à Arquitetura de Sistemas Operacionais*. 3 ed. Rio de Janeiro, LTC-Livros Técnicos e Científicos, 2003.
- MCBRYAN, O. A. An overview of message passing environments. *Parallel Computing*, vol. 20, pp. 417-444, 1994.
- MEHRA, P.; WAH, B.W. Automated Learning of Load-Balancing Strategies for a Distributed Computer System. University of Illinois at Urbana-Champaign, 1993.
- 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.
- NAIK, V.K.; SETIA, S.K.; SQUILLANTE, M.S. Performance Analysis of Job Scheduling Policies in Parallel Supercomputing Environments. *Proceedings of the conference on Supercomputing '93*, pp. 824 – 833, 1993.
- NGUYEN, T.D.; VASWANI, R.; ZAHORJAN, J. Using runtime measured workload characteristics in parallel processor scheduling. *Proceedings of IPPS '96 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 93--104, April, 1996.
- NUTT, G. J. Tutorial: Computer System Monitors. *IEEE Computing*, vol. 8, n. 11, pp. 51-61, novembro, 1972.
- ORLANDI, R.C.G.S. *Ferramentas para Análise de Desempenho de Sistemas Computacionais Distribuídos*. Dissertação (mestrado). ICMC-USP, São Carlos, março, 1995.
- PALLAS. *Vampir 2.0 User's Guide*. 1999.
- PLATFORM COMPUTING CORPORATION. *LSF Administrator's Guide*. junho, 2000.
- QUINN, M.J. *Parallel Computing: Theory and Practice*. 2.ed., McGraw-Hill Inc., 1994.
- RIBLER, R.L.; SIMITCI, H.; REED, D.A. The Autopilot Performance-Directed Adaptive Control System. *Future Generation Computer Systems*, special issue (Performance Data Mining), 18(1), setembro, 2001.
- SANTANA, M.J.; ZALUSKA, E.J. Load Balancing in a Session-based Distributed File-Store Architecture. *Software Practice and Experience*, vol. 18(11), p. 1091-1107, novembro, 1988.
- SETIA, S. K.; SQUILLANTE, M. J.; TRIPATHI, S. K. Processor Allocation in Multiprogrammed Distributed Memory Parallel Systems. *Journal of Parallel and Distributed Computing*, janeiro, 1998.
- SETIA, S. K.; SQUILLANTE, M. J., NAIK, V. The Impact of Job Memory Requirements on Gang-Scheduling Performance. *Performance Evaluation Review*, março, 1999.

- SCHNOR, B.; PETRI, S.; LANGENDÖRFER, H. Load Management for Load Balancing on Heterogeneous Platforms: A Comparison of Traditional and Neural Network Based Approaches. *Lecture Notes in Computer Science*, vol.1124, pp. 611-614, agosto, 1996.
- SCHNOR, B.; PETRI, S. Using On-line Job Monitoring for Dynamic Load Balancing of Parallel Applications. *Proceedings of The International Conference on Parallel and Distributed Processing Techniques and Applications PDPTA '99*, julho, 1999.
- SHAO, G.; WOLSKI, R.; BERMAN, F. *Performance Effects of Scheduling Strategies for Master/Slave Distributed Applications*. UCSD CSD Dept. Technical Report, setembro, 1998.
- SHAO, G. *Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources*. Tese (doutorado), University of California, 2001.
- SHIRAZI, B.; HURSON, A.R. Special Issue on Scheduling and Load Balancing: guest editor's introduction. *Journal of Parallel and Distributed Computing*, vol.16, Iss 4, p.271-275, 1992.
- 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.
- SHIVARATRI, N.G.; KRUEGER, P.; SINGHAL, M. Load Distributing for Locally Distributed Systems. *IEEE Computer*, dezembro, 1992.
- SILVA, F.A.B. *Balanceamento de Carga em Ambientes Paralelos Virtuais com Aplicações no PVM-W95*. Dissertação (mestrado), ICMC/USP, São Carlos, 1997.
- SILVA, F.A.B.; SCHERSON, I. D. Improving Parallel Job Scheduling Using Runtime Measurements. *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing*, maio, 2000.

- SINHA, A.; KALE V. A Load balancing strategy for Prioritized Execution of Tasks. *Proceedings of the Workshop on Dynamic Object Placement and Load Balancing, {ECOOP}*, 1992.
- SMITH, W.; FOSTER, I.; TAYLOR, V. Predicting application run times using historical information. *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
- SNIR, M.; OTTO, S.; STEVEN, H.; WALKER, D.; DONGARRA, J.J. *MPI: The Complete Reference*. 2.ed. The MIT Press, Massachusetts, 1998.
- SOARES, L. F.G. *Modelagem e Simulação Discreta de Sistemas*. Editora Campus Ltda, 1992.
- SONG, J.; CHOO, H.K.; LEE, K.M. Application-level load migration and its implementation on top of PVM. *Concurrency: Practice and Experience*, vol.9(1), p.1-19, janeiro, 1997.
- SOUZA, M. A. *Avaliação das Rotinas de Comunicação Ponto-a-Ponto do MPI*. Dissertação (mestrado), ICMC/USP, São Carlos, 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*, junho, 1999.
- SOUZA(b), P. S.; SANTANA, M. J.; SANTANA, R. H. C. AMIGO – A Dynamical Flexible Scheduling Environment. *Proceedings of the 5th International Conference on INFORMATION SYSTEMS, ANALYSIS AND SYNTHESIS: ISAS'99*, junho, 1999.
- SOUZA(c), P.S.L., SANTANA, M.J., SANTANA, R.H.C., ARAÚJO, A.P.F.. PVM and a Viable and Flexible Scheduling. *Proceedings of Eleventh IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'99)*, novembro, 1999.
- SOUZA, P.S.L.; SANTANA, M.J.; SANTANA, R.H.C. Escalonamento de Processos: uma contribuição para a Convergência da Área. *Notas do ICMC*, ICMC/USP, São Carlos, maio, 2000.

- SOUZA(a), P. S. L. AMIGO: *Uma Contribuição para a Convergência na Área de Escalonamento de Processos*. Tese (Doutorado). IFSC-USP, São Carlos, Junho de 2000.
- SOUZA(b), J. A. M. *Avaliação de Desempenho de Políticas de Escalonamento de Processos*. Dissertação (mestrado), ICMC-USP, São Carlos, dezembro, 2000.
- 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 vol.949*, Santa Barbara, CA, USA, abril, 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 vol.1300*, agosto, 1997.
- SUN MICROSYSTEMS, INC. *Sun Grid Engine 5.3 Administration and User's Guide*, 2002.
- TANENBAUM, A. S. *Distributed Operating Systems*, Prentice Hall, 1995.
- TANENBAUM, A.S.; STEEN, M.V. *Distributed Systems – Principles and Paradigms*. New Jersey, Prentice Hall International, Inc., 2001.
- TANENBAUM, A.S. *Sistemas Operacionais Modernos*. Prentice Hall, Inc., 2003.
- THAIN, D.; TANNENBAUM, T.; LIVNY, M. Condor and the Grid, in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, ed., *Grid Computing: Making The Global Infrastructure a Reality*, cap. 11, John Wiley, 2003.
- TOPCUOGLU, H.; HARIRI, S.; WU, M. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, n. 3, 2002.
- TRIPATHI, A.R.; KARNIK, N.M. Trends in multiprocessor and distributed operating-systems designs. *Journal of supercomputing*, vol.9, Iss 1-2, p.23-49, 1995.
- VERIDIAN SYSTEMS. *PBS – Portable Batch Systems*. <http://pbs.mrj.com>, 2000.
- VOORSLUYS, W.; SOUZA, P.S.L.; SOUZA, SRS. A specific scheduling algorithm for memory bound applications on heterogeneous clusters. 2003.

- WANG, Y.T.; MORRIS, R.J.T. Load Sharing in Distributed Systems. *IEEE Transactions on Computers*, p.204-217, março, 1985.
- WIBISONO, A. *Cross Cluster Migration using Dynamite Remote File Access Support*. Dissertação (mestrado). University of Amsterdam, 2002.
- WOLSKI, R.; SPRING, N.; and HAYES J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, vol. 15, n. 5-6, Outubro, 1999.
- XIAO, L.; CHEN, S.; ZHANG, X. Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, n. 3, março, 2003.
- XU, C.; LAU, F.C.M. *Load Balancing in Parallel Computers: Theory and Practice*. Kluwer Academic Publishers, Boston, USA, 1997.
- ZHANG, Y.; SIVASUBRAMANIAM, A.; MOREIRA, J.; FRANKE, H. A simulation-based study of scheduling mechanisms for a dynamic cluster environment. *Proceedings of the 2000 international conference on Supercomputing*, pp 100-109, 2000.
- ZHOU, S.; ZHENG, X.; WANG, J.; DELISLE, P. Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software: Practice and Experience*, vol.23(12), p.1305-1336, dezembro, 1993.

Apêndice A – Glossário

A ciência da computação se caracteriza pela falta de uma padronização em relação à definição de termos comumente usados em livros e artigos em geral, e a área de escalonamento de processos não foge à regra. Neste trabalho utiliza-se a seguinte nomenclatura:

Sistema computacional: definição geral e ampla de um conjunto formado por hardware, sistema operacional e softwares básicos, que caracteriza o ambiente computacional que se está utilizando.

Processador: refere-se a um elemento de processamento, considerado como sinônimo de computador, recurso de processamento, *host* ou nó. Um computador paralelo (ou arquitetura paralela) é definido como um conjunto de processadores.

CPU: Unidade central de processamento. Evita-se o uso do termo *processador* para não haver confusão com a definição acima.

Processo: abstração de um programa em execução, composto de código fonte, entrada, saída, dados do programa, pilha e contador de programa.

Aplicação: maior entidade de execução, sinônimo de *job*, que pode ser composta por diversos processos, tarefas ou programas.

CPU-Bound: aplicações que utilizam predominantemente a *CPU*. Mantém-se a nomenclatura em inglês por estar consagrada pelo uso.

I/O-Bound: aplicações onde predominam operações de Entrada/Saída (*In/Out*). Por ser uma definição genérica, prefere-se definir as aplicações de acordo com a sua utilização de memória, disco rígido e rede de comunicação.

Memory-Bound: aplicações caracterizadas por um intenso acesso à memória. Mantém-se a nomenclatura em inglês por esta estar consagrada pelo uso.

Disk-Bound: aplicações caracterizadas por acessar intensamente o disco rígido. Mantém-se a nomenclatura em inglês por esta estar consagrada pelo uso.

Network-Bound: aplicações caracterizadas por acessar intensamente a rede de comunicação. Mantém-se a nomenclatura em inglês por esta estar consagrada pelo uso.

Aplicações batch: aplicações com poucas requisições de E/S e grande quantidade de processamento, comuns em *mainframes*.

Aplicações interativas: aplicações que requerem interação com o usuário, por exemplo, através do mouse ou teclado.

Aplicações paralelas: aplicações compostas por diversos processos que trabalham cooperativamente para a resolução de um determinado problema, e que podem ser executadas em computadores paralelos ou sistemas distribuídos.

Apêndice B – Noções de Estatística

Este apêndice apresenta uma descrição das rotinas estatísticas utilizadas pelo PSMS, e foi baseado nas seguintes referências: (Jain, 91; Soares, 91)

A estatística pode ser definida como a ciência da aprendizagem a partir de um conjunto de dados. Ou seja, tem-se um grande conjunto de dados, e tem-se a intenção de tomar uma decisão a partir desses dados. A estatística, no caso, oferece diversos métodos que auxiliam esse processo de decisão.

A estatística é dividida em três grandes áreas, embora não se trate de ramos distintos. São elas: **amostragem e planejamento de experimentos**, que são responsáveis pela coleta dos dados a serem analisados; **estatística descritiva**, que se ocupa da organização, apresentação e sintetização dos dados; **estatística inferencial**, onde são definidos os métodos que serão utilizados na tomada de decisões quando há situações de incerteza e variação.

Neste trabalho de doutorado, foram utilizadas algumas **medidas estatísticas**, que são utilizadas dentro da estatística descritiva, e que dividem-se em: medidas de **locação** ou de **tendência central** e medidas de **dispersão** ou de **variabilidade**. Cada uma delas é discutida nas próximas seções.

B.1 Medidas de Locação

As medidas de locação, quando aplicadas sobre um conjunto de dados, mostram um valor representativo em torno do qual os dados tendem a se agrupar, ou seja, esse valor dá uma característica chave do grupo de dados, de maneira que sintetiza-se em apenas um número o conjunto de dados observados.

Existem três tipos principais de medidas de locação, que são: média, mediana e modo. A mais utilizada entre elas é a média, e na próxima subseção é apresentada uma discussão sobre esta medida e algumas variações dela utilizadas neste trabalho, que são: média ponderada e média móvel exponencial.

B.1.1 Média

O tipo mais básico de média é a **média aritmética**, que é calculada através da divisão entre a soma de todos os valores dos dados e o número total de dados. Por exemplo, para um conjunto de n dados, $x_1, x_2, \dots, x_n$, a média aritmética, denotada por $\bar{x}$, é calculada pela seguinte equação:

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum x_i}{n} \quad (\text{b.1})$$

Em algumas situações os dados sobre os quais se quer calcular a média têm graus de importância diferentes. Nesse caso, utiliza-se a **média aritmética ponderada**, e definem-se pesos para cada um dos valores. Para um conjunto de n dados, $x_1, x_2, \dots, x_n$, com pesos $p_1, p_2, \dots, p_n$, a média aritmética ponderada, denotada por $\bar{x}_p$, é calculada pela seguinte equação:

$$\bar{x}_p = \frac{x_1 p_1 + x_2 p_2 + \dots + x_n p_n}{p_1 + p_2 + \dots + p_n} \quad (\text{b.2})$$

Um tipo especial de média, utilizada principalmente para prever tendências de queda ou alta no valor de ações da bolsa de valores, é a **média móvel**. A média móvel é aplicada sobre um conjunto de valores (preços de uma ação) calculados em um período de dias consecutivos (esse período é chamado de **janela**), que será chamado de série temporal.

Por exemplo, a média móvel de **janela** j (ou seja, período de j dias), calculada no dia d , é obtida calculando a média dos j valores da série temporal disponíveis no dia d e nos $j-1$ dias imediatamente anteriores. Por exemplo, designando por x_i o dado medido no dia i , a média móvel simples (aritmética) de janela 5 é, calculada no dia 5:

$$(x_1 + x_2 + x_3 + x_4 + x_5)/5 \quad (\text{b.3})$$

No dia 6, calcula-se:

$$(x_2 + x_3 + x_4 + x_5 + x_6)/5 \quad (\text{b.4})$$

Genericamente, no dia j

$$(x_{j-4} + x_{j-3} + x_{j-2} + x_{j-1} + x_j)/5 \quad (\text{b.5})$$

Portanto, a média se move com o tempo, de maneira que a cada novo dia, um valor é acrescentado e um é descartado no cálculo da média. Existem vários tipos de médias móveis, e não se tem a intenção de descrever todos eles, e sim o tipo de média móvel que é utilizado pelo PSMS, que é chamado de **média móvel exponencial**.

Médias móveis exponenciais funcionam como uma média ponderada, e colocam maior peso nos últimos dados coletados. Também, os valores antigos não são descartados, mas contam no cálculo da média com menor peso.

Quanto menor o tamanho da janela, maior será o peso dado aos últimos valores. A média móvel de janela j , calculada no tempo i , denotada por M_i , é calculada pela seguinte equação:

$$M_i = (1 - k)M_{i-1} + k.x_i \quad (\text{b.6})$$

onde: M_i = Média exponencial calculada no i -ésimo dia

k = Constante exponencial, que equivale a $2/(1+j)$

M_{i-1} = Média exponencial calculada no dia anterior ($i-1$)

x_i = Dado coletado na i -ésima observação

j = Janela

Embora não fique claro pela equação b.6, esse tipo de média representa uma média ponderada cujos valores dos pesos formam uma curva exponencial, como pode ser visto no desenvolvimento da equação b.6, mostrado a seguir.

Fixa-se $j \geq 1$, e define-se $k = \frac{2}{1+j}$, onde $0 < k \leq 1$

De $M_i = (1 - k)M_{i-1} + k.x_i$, substituindo-se M_{i-1} pela sua respectiva fórmula, tem-se:

$$\begin{aligned}
M_i &= (1-k)[(1-k)M_{i-2} + kx_{i-1}] + kx_i = \\
M_i &= (1-k)^2 M_{i-2} + (1-k)kx_{i-1} + kx_i = \\
M_i &= (1-k)^2 [(1-k)M_{i-3} + kx_{i-2}] + (1-k)kx_{i-1} + kx_i = \\
M_i &= (1-k)^3 M_{i-3} + (1-k)^2 kx_{i-2} + (1-k)kx_{i-1} + kx_i \\
M_i &= (1-k)^p M_{i-p} + k \sum_{z=0}^{p-1} (1-k)^z x_{i-z}
\end{aligned}$$

onde: $p=i-1$

Portanto, a média móvel exponencial M_i , considerando os valores dos dados obtidos nos últimos p dias, $x_i, x_{i-1}, x_{i-2}, \dots, x_{i-p}$, é calculada multiplicando a cada valor de x a expressão $k(1-k)^z$, onde z varia entre 0 e $p-1$. Portanto, caracteriza-se aí uma média ponderada com um peso variando exponencialmente.

A aplicação desse conceito à abordagem do PSMS é direta, visto que as métricas de desempenho são calculadas em períodos de tempo consecutivos e fixos, de maneira similar ao mercado de ações, onde considera-se um valor para o preço de uma ação por dia.

B.2 Medidas de Dispersão ou Variabilidade

As medidas de dispersão, quando aplicadas sobre um conjunto de dados, apresentam uma conclusão sobre a variabilidade dos dados. Isso ocorre porque dois conjuntos de dados podem ter a mesma média aritmética, por exemplo, mas pode interessar escolher o conjunto cujos valores não possuam muita variação.

Existem três tipos principais de medidas de locação, que são: variância, desvio padrão e coeficiente de variação. Cada uma delas é descrita nas subseções seguintes

B.2.1 Variância e Desvio Padrão

A **variância** de um conjunto de n dados, $x_1, x_2, \dots, x_n$, denotada por S^2 , é calculada através da seguinte fórmula:

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}, \text{ onde } \bar{x} \text{ é o valor da média dos } n \text{ dados.}$$

Calculando-se a raiz quadrada da variância, consegue-se uma medida de dispersão cuja unidade é a mesma dos dados originais. Essa medida, chamada de **desvio padrão** é denotada por S .

B.2.2 Coeficiente de Variação

Muitas vezes, é interessante representar a variação em termos relativos, ou seja, tem-se a intenção de comparar as variações de dois conjuntos de dados. O desvio padrão possui a mesma unidade do conjunto de dados, e um desvio padrão de 10 pode ser altamente significativo para um conjunto de dados que possua valores próximos a 20, mas é insignificante para um conjunto de dados que possua valores próximos a 10.000. Para comparar a variabilidade de dois conjuntos distintos de dados, utiliza-se uma medida de dispersão adimensional, calculada dividindo-se o desvio padrão pela média aritmética, chamada de **coeficiente de variação**.

Portanto, o coeficiente de variação, denotado por cv , de um conjunto de dados com desvio padrão igual a S e média $\bar{x}$ é igual a:

$$cv = \frac{S}{\bar{x}}$$

B.3 Referências Bibliográficas

SOARES, J. F.; FARIAS, A. A.; CESAR, C. C. *Introdução à Estatística*. Editora Guanabara Koogan, 1991.

JAIN, R. *The art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Inc, 1991.

Apêndice C – UML (*Unified Modeling Language*)

Este apêndice apresenta algumas definições básicas sobre UML e uma descrição dos dois diagramas utilizados nesta tese. Baseia-se nas seguintes referências (Booch et al, 2000; Kobryn, 1999; Ambler, 2000).

O desenvolvimento da UML teve início em 1994 com o objetivo de tornar-se um padrão internacional de linguagem para a especificação de sistemas de software. Esta linguagem pretende substituir todos os métodos de projeto e análise de software (tais como Booch, Coad, Jacobsen, Odell, Rumbaugh, etc.) por uma única notação, apoiada pela OMG (*Object Management Group*).

A UML é uma linguagem gráfica padronizada destinada à especificação, à construção, à visualização e à documentação de sistemas de software. Tem como objetivo oferecer a qualquer pessoa envolvida na produção, instalação e manutenção de software uma notação padronizada para expressar o projeto de um sistema de software.

UML é formada por três elementos principais: os blocos de construção básicos da UML, as regras que determinam como esses blocos poderão ser combinados e alguns mecanismos comuns aplicados na UML. Os blocos de construção básicos são divididos em três tipos: itens, relacionamentos e diagramas.

Os itens são as abstrações identificadas como cidadãos de primeira classe em um modelo UML. Relacionamentos reúnem esses itens, e os diagramas são conjunto de itens combinados de maneira específica.

Não serão aqui citados todos os itens básicos de UML, mas apenas os que são citados nesta tese de doutorado, e que foram utilizados para construir dois tipos de diagramas, que são: diagrama de casos de uso e de atividades. Esses dois diagramas são descritos a seguir.

C.1 Diagramas de Casos de Uso

O diagrama de casos de uso especifica o comportamento de um sistema, sem a necessidade de detalhar como esse comportamento será implementado. O comportamento

do sistema é representado pela descrição de um conjunto de atividades (cada uma representada por um caso de uso) cujos resultados sejam observáveis por um ator (que pode ser um usuário ou um software que interage com o sistema). Um exemplo de um diagrama de casos de uso é apresentado na figura C.1.

Um caso de uso é representado por uma elipse e um ator é representado por um símbolo gráfico que representa um pequeno ser humano. Os atores e os casos de uso possuem entre si um relacionamento de *associação*, representando que existe uma comunicação entre eles. Uma associação é representada por uma linha sólida. Os casos de uso também mantêm entre si relacionamentos de *dependência* de dois tipos: de inclusão e de extensão.

Figura C.1 – Exemplo de um Diagrama de Casos de Uso

Uma dependência é representada por uma seta tracejada, e o tipo de dependência vem expresso através de um rótulo na flecha que a representa. Um relacionamento de inclusão (rótulo *uses*) indica que um determinado caso de uso inclui um outro caso de uso. Por exemplo, na figura C.1, o caso de uso **fazer pedido** inclui o caso de uso **validar usuário**. Um relacionamento de extensão (rótulo *extends*) representa que um determinado caso de uso pode incluir, opcionalmente, um caso de uso que o estenda. Por exemplo, na

figura C.1, o caso de uso **fazer pedido** pode, se desejar, incluir o caso de uso **pedido de urgência**.

Além da associação de dependência, um caso de uso pode ser definido como uma generalização de outro. Por exemplo, na figura c.1, o caso de uso **validar usuário** é uma generalização dos casos de uso **verificar digital** e **digitar senha**, e por isso pode ser substituído por qualquer um deles.

C.2 Diagrama de Atividades

Um diagrama de atividades representa basicamente um gráfico de fluxo, representando o fluxo de controle de uma atividade para outra. Uma atividade representa uma determinada ação que é desempenhada pelo sistema de software sendo projetado. A figura C.2 apresenta um exemplo de um diagrama de atividades.

O fluxo de atividades inicia-se em um **estado inicial**, representado na figura por uma bola sólida de cor negra. O fluxo de controle entre as diversas atividades é representado por uma seta (chamada **transição**), e o sentido da flecha indica o sentido que o fluxo caminha. Em determinados pontos, pode haver a necessidade do fluxo seguir por dois caminhos distintos de acordo com uma determinada condição. Esse tipo de **ramificação sequencial** é representado por um losango, e a escolha da transição a ser seguida é feita de acordo com rótulos que são utilizados para identificar as diferentes transições. Por exemplo, na figura C.2, os rótulos são definidos como: *[rejeitado]* e *[else]*. A palavra reservada *else* representa um caminho alternativo quando a primeira condição é falsa.

Figura C.2 – Exemplo de um Diagrama de Atividades

Determinadas atividades podem ocorrer paralelamente, e nesse caso utiliza-se uma **bifurcação concorrente**, que representa o ponto onde os diversos fluxos de controle correm paralelamente. No final, os fluxos se sincronizam através de uma **união concorrente**.

Figura C.3 – Exemplo de Duas Atividades que Terminam em um Ponto Comum

Determinados fluxos alternativos, que não são executados paralelamente, e que também terminam em um ponto comum, terminam em uma estrutura representada por um losango semelhante a uma ramificação, conforme mostra a figura C.3. Ao final das atividades, o fluxo de controle termina em um estado final, representado por uma bola negra com borda branca.

C.3 Referências Bibliográficas

- AMBLER, S.W. How the UML models fit together. *Software Development's special report on the UML*, março, 2000.
- BOOCH, G; RUMBAUGH, J.; JACOBSON, I. *UML: Guia do Usuário*. Editora Campus, 2000.
- KOBYRN C. UML 2001: A Standardization Odyssey. *Communications of the ACM*, v. 42, n. 10, outubro, 1999.