UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA

SIMULAÇÃO DISTRIBUÍDA EM PLATAFORMAS DE PORTABILIDADE: VIABILIDADE DE USO E COMPORTAMENTO DO PROTOCOLO CMB

ROBERTA SPOLON ULSON

Tese apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo, para obtenção do título de Doutor em Ciências "Física Aplicada-Opção: Física Computacional"

Orientadora: Profa. Dra. Regina Helena Carlucci Santana

São Carlos – São Paulo
1999

USP/IFSC/SBI
8-2-001308

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Ulson, Roberta Spolon

"Simulação distribuída em plataformas de portabilidade: Viabilidade de uso e comportamento do protocolo CMB"/ Roberta Spolon Ulson – São Carlos, 1999

Tese (Doutorado) – Instituto de Física de São Carlos da Universidade de São Paulo, 1999
Área: Física Aplicada-Opção: Física Computacional
Profa. Dra. Regina Helena Carlucci Santana

1. Simulação. 2. Plataformas. 3. CMB.

I. Título

UNIVERSIDADE DE SÃO PAULO
Instituto de Física de São Carlos

Av. Dr. Carlos Botelho, 1465
CEP 13560-250 - São Carlos - SP
Brasil

Fone (016) 273-9333
Fax (016) 272-2218

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE ROBERTA SPOLON ULSON APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 17 DE DEZEMBRO DE 1999.

COMISSÃO JULGADORA:

Profa. Dra. Regina Helena Carlucci Santana/ICMC/USP

Prof. Dr. Carlos Antonio Ruggiero/IFSC/USP

Profa. Dra. Patrícia Magna/IFSC/USP

Prof. Dr. Edil Severiano Tavares Fernandes/UFRJ

Prof. Dr. Ricardo Anido/UNICAMP

e-mail: wladerez@ifsc.sc.usp.br

"O caminho do conhecimento é um caminho aberto a todos os homens, às pessoas comuns."
Paulo Coelho

Aos meus pais Dorival e Meiri

Aos meus irmãos Renata, Sérgio e Ricardo

Ao meu esposo Alfredo

Pelo amor, carinho e incentivo, que tornaram possível esta conquista.

AGRADECIMENTOS

À Regina, pela orientação sempre segura, pela amizade, incentivo e apoio. Por me ajudar a enfrentar os desafios e ajudar a vencê-los.

Aos meus pais, que representam a razão e o incentivo deste trabalho. Pelo seu amor, carinho e sobretudo pelo seu sacrifício.

Ao Marcos Santana, pela amizade, pelas valiosas contribuições, pelo incentivo e apoio indispensáveis na realização deste trabalho.

À Renata, pelo auxílio sempre constante e fundamental durante o desenvolvimento deste trabalho, por estar presente em todos os momentos e principalmente, por ser a minha grande amiga.

Ao Alfredo, pelo apoio em todos os momentos.

Ao Marcos Cavenaghi, pela amizade, competência e disposição com que sempre me ajudou em todos os momentos.

Ao Boca (João Carlos), pela amizade, pelas muitas dúvidas que ajudou a solucionar, pelas valiosas contribuições para este trabalho e pelo incentivo sempre constante.

Ao João Perea, pela amizade e pelas palavras sempre encorajadoras.

Ao Professor Eduardo Marques, pelo incentivo constante.

À Wladerez, da seção de Pós-graduação do IFSC-USP, pela simpatia, atenção e eficiência com que sempre me atendeu.

Aos amigos Márcio, Paulo Sérgio e Sarita pelas muitas dúvidas solucionadas e pelas valiosas contribuições para este trabalho.

Aos amigos do Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP, conquistados durante esses anos em São Carlos: Aletéia, Álvaro, Andrezza, Arion, Célia (Daniel), Daniel, Edimilson, Fernanda, Jorge, Kalinka, Mara, Omar, Regiane, Renato, Silmara, Simone (Felipe), Tatiana, Tomás, Vera e Walter.

Ao Centro de Informática de São Carlos da Universidade de São Paulo (CISC-USP), sem o qual este trabalho não seria possível. Ao Rogério, pelo ótimo atendimento e auxílio nos momentos de dúvidas. À Lurdinha, pela prestatividade e atenção.

Aos funcionários do ICMC-USP, pela atenção com que sempre me atenderam: Jacques, Paulinho, Beth, Laura, Zé Ângelo, Eduardo e Cabral.

Ao Departamento de Computação da UNESP, campus Bauru.

À Capes, pelo apoio financeiro.

Àqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

RESUMO

ABSTRACT

CAPÍTULO 1

INTRODUÇÃO 1

CAPÍTULO 2

COMPUTAÇÃO PARALELA E AMBIENTES DE PASSAGEM DE MENSAGENS

2.1 Introdução 7

2.2 Conceitos Básicos Sobre Computação Paralela 9

2.2.1 Tipos de Paralelismo 10

2.2.2 Granulosidade 12

2.2.3 *Speedup* (*Sp*) 12

2.2.4 Eficiência (E) 14

2.3 Arquiteturas Paralelas 14

2.3.1 Classificação de Flynn 14

2.3.2 Classificação de Duncan 18

2.3.2.1 Arquiteturas Síncronas 19

2.3.2.2 Arquiteturas Assíncronas 20

2.4 Paradigmas de Programação Paralela 22

2.5 Programação Paralela Utilizando Ambientes de Passagem de Mensagens 25

2.5.1 Computação Paralela em Sistemas Distribuídos 26

2.5.2 Ambientes de Passagem de Mensagens 27

2.6 *PVM* 31

2.6.1 O Modelo Computacional no *PVM* 33

2.6.2 O Modelo de Comunicação no *PVM* 35
2.6.3 Outras Características do *PVM* 37
2.7 *IBM PVMe* 38
2.8 Considerações Finais 40

CAPÍTULO 3
PROTOCOLOS DE SINCRONIZAÇÃO EM SIMULAÇÃO DISTRIBUÍDA
3.1 Introdução 42
3.2 Conceitos Básicos Sobre Simulação Discreta 44
3.2.1 Composição de Sistemas Discretos 45
3.3 Simulação Distribuída 48
3.4 O Problema Fundamental em Simulação Distribuída 50
3.5 Protocolos para Sincronização em Simulação Distribuída 53
3.5.1 *Lookahead* 56
3.5.2 Protocolos Conservativos 57
3.5.2.1 O Protocolo *CMB* (Prevenção do *Deadlock* Utilizando Mensagens Nulas 57
3.5.2.2 Transmissão de Mensagens Nulas Sob Demanda 60
3.5.2.3 Detecção e Recuperação do *Deadlock* 63
3.5.2.4 Considerações Sobre os Protocolos Conservativos 65
3.5.3 Protocolos Otimistas 65
3.5.3.1 *Time Warp* 66
3.5.3.2 Estratégias de Cancelamento 68
3.5.3.3 Técnicas para Gerenciamento de Memória 68
3.6 Comparação entre Protocolos Conservativos e Otimistas 69
3.7 O Estado da Arte em Simulação Distribuída 72
3.7.1 Linguagens para Simulação Distribuída 73
3.7.2 Bibliotecas para Simulação Distribuída 74
3.8 Considerações Finais 77

CAPÍTULO 4
DESCRIÇÃO DA EXTENSÃO FUNCIONAL *ParSMPL*
4.1 Introdução 79
4.2 Avaliação de Desempenho da Simulação Distribuída 80
4.3 *Benchmark* para Avaliação de Desempenho 83
4.4 A Extensão *ParSMPL* 86

4.5 Abordagem para o Desenvolvimento de um Programa de Simulação 89
4.5.1 Estrutura do *SMPL* 91
4.5.2 Primitivas do *SMPL* 94
4.6 Versão Seqüencial do *SMPL* para Plataforma *UNIX* - *SMPLX* 95
4.7 Implementação do Protocolo *CMB*: A Extensão Funcional *ParSMPL* 101
4.7.1 A Estrutura do *ParSMPL* 102
4.7.2 Principais Rotinas do *ParSMPL* 105
4.7.3 Tabela de Identificação de Processos 108
4.7.4 O Modelo de Programação no *ParSMPL* 109
4.7.5 Tipos de Mensagens 111
4.8 Implementação dos Processos e Comunicação no *ParSMPL* 113
4.9 Exemplo de Utilização do *ParSMPL* 117
4.10 Validação do *ParSMPL* 122
4.11 Considerações Finais 124

CAPÍTULO 5

DESENVOLVIMENTO DE MODELOS PARA A AVALIAÇÃO DE DESEMPENHO DO PROTOCOLO *CMB*

5.1 Introdução 127
5.2 Metodologia e Métricas Utilizadas 128
5.3 Plataformas Utilizadas 130
5.4 Modelos Propostos 133
5.4.1 Modelo 1: Servidor Central 134
5.4.2 Modelo 2: Servidor de Arquivos do Sistema Distribuído *Trice* 137
5.4.3 Modelo 3: Sistema Computacional Simplificado 142
5.4.4 Modelo 4: Sistema Computacional 144
5.4.5 Modelo 5: Sistema de Computação com Processamento por Transação 147
5.4.6 Modelo 6: Sistema de Computação com Processamento Interativo 148
5.4.7 Modelos Hipotéticos 150
5.4.7.1 Modelo Hipotético 1 151
5.4.7.2 Modelo Hipotético 2 153
5.4.7.3 Modelo Hipotético 3 156
5.4.7.4 Modelo Hipotético 4 157
5.4.7.5 Modelo Hipotético 5 159
5.5 Considerações Finais 161

CAPÍTULO 6

INFLUÊNCIA DAS CARACTERÍSTICAS DA PLATAFORMA NO DESEMPENHO DA SIMULACAO DISTRIBUÍDA

6.1 Introdução 164

6.2 Metodologia 168

6.3 Avaliação e Classificação das Plataformas Utilizadas 169

6.3.1 Comunicação 169

6.3.2 Elementos de Processamento 173

6.4 Modelos Utilizados para a Análise de Desempenho 175

6.5 Avaliação de Desempenho da Simulação Distribuída 178

6.5.1 Desempenho da Simulação Distribuída – *SP2/PVMe* 179

6.5.2 Desempenho da Simulação Distribuída – *Token Ring* 181

6.5.3 Desempenho da Simulação Distribuída – *SP2/PVM* 183

6.5.4 Desempenho da Simulação Distribuída – *Ethernet* 185

6.5.5 Desempenho da Simulação Distribuída – Mensagens Completas e Nulas 187

6.6 Estudo para Previsão do Comportamento da Simulação 190

6.7 Considerações Finais 193

CAPÍTULO 7

ANÁLISE DA INFLUÊNCIA DAS CARACTERÍSTICAS DO MODELO E PARTIÇÃO DOS PROCESSOS LÓGICOS NO DESEMPENHO DA SIMULAÇÃO DISTRIBUÍDA

7.1 Introdução 196

7.2 Análise do Desempenho da Simulação Distribuída 198

7.2.1 Partição do Modelo em Processos Lógicos 200

7.2.1.1 Hipótese 1: Partição do Modelo em Processos Lógicos (1) 201

7.2.1.2 Hipótese 2: Partição do Modelo em Processos Lógicos (2) 206

7.2.1.3 Hipótese 3: Partição do Modelo em Processos Lógicos (3) 214

7.2.1.4 Conclusão - Partição do Modelo em Processos Lógicos 216

7.2.2 Alteração da Granulosidade e Número de Processos Lógicos 217

7.2.2.1 Hipótese 4: Granulosidade e Número de Processos Lógicos 217

7.2.2.2 Hipótese 5: Granulosidade e Número de Processos Considerando Modelos com Estruturas em Série 223

7.2.2.3 Comentários - Granulosidade e Número de Processos Lógicos 225

7.2.3 Alteração nos Parâmetros da Aplicação 226

7.2.3.1 Hipótese 6: Probabilidade nos Pontos de Decisão 226

7.2.3.2 Comentários – Probabilidade nos Pontos de Decisão 230
7.2.3.3 Hipótese 7: Tempo entre Chegadas de Tarefas 231
7.2.3.4 Comentários – Tempo entre Chegadas de Tarefas 235
7.2.3.5 Hipótese 8: Tempo de Serviço dos Recursos 236
7.2.3.6 Comentários: Tempo de Serviço dos Recursos 238
7.2.3.7 Hipótese 9: Número de Tarefas em Sistema do Tipo Fechado 239
7.2.3.8 Comentários – Número de Tarefas em Sistemas do Tipo Fechado 243
7.3 Simulação Distribuída sem Sincronismo 243
7.3.1 Modelo Hipotético 4 244
7.4 Considerações Finais 247

CAPÍTULO 8

CONCLUSÕES, CONTRIBUIÇÕES E PROPOSTA DE TRABALHOS FUTUROS

8.1 Introdução 252
8.2 Conclusões 255
8.3 Contribuições 260
8.4 Propostas para Trabalhos Futuros 262

REFERÊNCIAS BIBLIOGRÁFICAS 265

APÊNDICE A 274

ÍNDICE DE FIGURAS

Figura 2.1	Tipos de Paralelismo.	11
Figura 2.2	Modelo Computacional *SISD*.	15
Figura 2.3	Modelo Computacional *SIMD*.	16
Figura 2.4	Modelo Computacional *MISD*.	17
Figura 2.5	Modelo Computacional *MIMD*.	18
Figura 2.6	Classificação de Duncan para Arquiteturas Paralelas.	19
Figura 2.7	Paradigma de Passagem de Mensagem.	28
Figura 2.8	Modelo Computacional e Visão Arquitetural do *PVM*.	35
Figura 3.1	Relação entre Evento, Processo e Atividade.	47
Figura 3.2	Ocorrência de Erros de Causa e Efeito.	52
Figura 3.3	Classificação dos Protocolos em Simulação Distribuída.	53
Figura 3.4	*LVT* de um Processo em uma Simulação Distribuída Conservativa	58
Figura 3.5	Situação de *Deadlock* em uma Simulação Distribuída.	59
Figura 3.6	Transmissão de Mensagens Nulas Sob Demanda.	62
Figura 3.7	Transmissão de Mensagens Nulas Sob Demanda.	63
Figura 4.1	*Benchmark Ping-Pong*.	86
Figura 4.2	Hierarquia dos Níveis de *Software* no *ParSMPL*.	88
Figura 4.3	Estrutura Básica de um Programa de Simulação em *SMPL*.	92
Figura 4.4	Elemento Básico de Dados do *SMPL*.	93
Figura 4.5	Estruturas de Dados do *SMPLX*.	97
Figura 4.6	Lista de Eventos Futuros no *SMPLX*.	99
Figura 4.7	Estrutura de Processos no *ParSMPL*	104
Figura 4.8	Hierarquia das Rotinas no *ParSMPL*	107
Figura 4.9	Modelo de Programação 1 no *ParSMPL*.	110
Figura 4.10	Modelo 2 de Programação no *ParSMPL*	111
Figura 4.11	Estrutura de uma Mensagem Completa.	112
Figura 4.12	Estrutura de uma Mensagem Nula.	113
Figura 4.13	Descrição de um Processo no *ParSMPL*.	114
Figura 4.14	Comunicação entre Processos no *ParSMPL*.	116
Figura 4.15	Modelo do Servidor Central – Código em *SMPL*.	118
Figura 4.16	Modelo do Servidor Central – Código Processo Lógico 0 – *ParSMPL*.	120

Figura 4.17	Modelo do Servidor Central – Código Processo Lógico 1 – *ParSMPL*.	121
Figura 5.1	O Ambiente de Desenvolvimento.	132
Figura 5.2	Modelo de Redes de Filas: Servidor Central (configuração 1).	136
Figura 5.3	Modelo de Redes de Filas: Servidor Central (configuração 2).	136
Figura 5.4	Modelo de Redes de Filas: Servidor de Arquivos do Sistema Distribuído *Trice* (configuração 1).	140
Figura 5.5	Modelo de Redes de Filas: Servidor de Arquivos do Sistema Distribuído *Trice* (configuração 2).	141
Figura 5.6	Modelo de Redes de Filas: Servidor de Arquivos do Sistema Distribuído *Trice* (configuração 3).	141
Figura 5.7	Modelo de Redes de Filas: Sistema Computacional Simplificado.	143
Figura 5.8	Modelo de Redes de Filas: Sistema Computacional (configuração 1).	145
Figura 5.9	Modelo de Redes de Filas: Sistema Computacional (configuração 2).	145
Figura 5.10	Modelo de Redes de Filas: Sistema Computacional (configuração 3).	146
Figura 5.11	Modelo de Redes de Filas: Sistema de Computação com Processamento por Transação.	148
Figura 5.12	Modelo de Redes de Filas: Sistema de Computação com Processamento Interativo.	150
Figura 5.13	Modelo de Redes de Filas: Modelo Hipotético 1 (configuração 1).	152
Figura 5.14	Modelo de Redes de Filas: Modelo Hipotético 1 (configuração 2).	152
Figura 5.15	Modelo de Redes de Filas: Modelo Hipotético 2 (configuração 1).	154
Figura 5.16	Modelo de Redes de Filas: Modelo Hipotético 2 (configuração 2).	155
Figura 5.17	Modelo de Redes de Filas: Modelo Hipotético 3 (configuração 1).	156
Figura 5.18	Modelo de Redes de Filas: Modelo Hipotético 3 (configuração 2).	157
Figura 5.19	Modelo de Redes de Filas: Modelo Hipotético 4 (configuração 1).	158
Figura 5.20	Modelo de Redes de Filas: Modelo Hipotético 4 (configuração 2).	158
Figura 5.21	Modelo de Redes de Filas: Modelo Hipotético 5 (configuração 1).	160
Figura 5.22	Modelo de Redes de Filas: Modelo Hipotético 5 (configuração 2).	160
Figura 5.23	Modelo de Redes de Filas: Modelo Hipotético 5 (configuração 3).	160
Figura 6.1	Eficiência de Comunicação.	172
Figura 6.2	Tempo de Comunicação – *Trice* (1).	173
Figura 6.3	Tempo Médio de Execução – Simulação Seqüencial.	174
Figura 6.4	Eficiência de Processamento.	175
Figura 6.5	Variação do *Speedup*.	178
Figura 6.6	Tempo de Comunicação e Processamento – *SP2/PVMe*.	179
Figura 6.7	Tempo de Comunicação e Processamento – *Token Ring*.	182
Figura 6.8	Tempo de Comunicação e Processamento – *SP2/PVM*.	184

Figura 6.9	Tempo de Comunicação e Processamento – *Ethernet*.	186
Figura 6.10	Total de Mensagens Nulas, Completas e *Speedup*.	188
Figura 6.11	Diagrama Comunicação x Processamento.	190
Figura 6.12	Desempenho da Simulação Distribuída.	192
Figura 7.1	*Speedup* - *Trice* (Partição dos Processos Lógicos).	202
Figura 7.2	*Speedup* - Sistema Computacional (Partição dos Processos Lógicos).	204
Figura 7.3	*Speedup* - Modelo Hipotético 1 (Partição dos Processos Lógicos).	207
Figura 7.4	*Speedup* - Modelo Hipotético 2 (Partição dos Processos Lógicos).	209
Figura 7.5	*Speedup* - Modelo Hipotético 4 (Partição dos Processos Lógicos).	210
Figura 7.6	*Speedup* - Modelo Hipotético 5 (Partição dos Processos Lógicos).	212
Figura 7.7	*Speedup* - *Trice* (Partição dos Processos Lógicos).	215
Figura 7.8	*Speedup* – Servidor Central (Granulosidade e Número de Processos Lógicos).	219
Figura 7.9	*Speedup* – Sistema Computacional (Granulosidade e Número de Processos Lógicos).	221
Figura 7.10	Tempo de Comunicação e Processamento – Servidor Central.	222
Figura 7.11	*Speedup* - Modelo Hipotético 3 (Granulosidade e Número de Processos Lógicos).	224
Figura 7.12	*Speedup* – Sistema Computacional (1) (Probabilidade nos Pontos de Decisão).	227
Figura 7.13	Total de Mensagens– Sistema Computacional (1) (Probabilidade nos Pontos de Decisão).	228
Figura 7.14	*Speedup* – Sistema Computacional (2) (Probabilidade nos Pontos de Decisão).	228
Figura 7.15	Total de Mensagens– Sistema Computacional (2) (Probabilidade nos Pontos de Decisão).	229
Figura 7.16	*Speedup* – Sistema Computacional Simplificado (Tempo entre Chegadas de Tarefas).	232
Figura 7.17	*Speedup*– Sistema de Computação com Processamento por Transação (Tempo Entre Chegadas de Tarefas).	234
Figura 7.18	*Speedup* – Sistema Computacional Simplificado (Tempo de Serviço dos Recursos).	237
Figura 7.19	*Speedup* – Servidor Central (1) (Número de Tarefas).	240
Figura 7.20	*Speedup* – Servidor Central (2) (Número de Tarefas).	241

ÍNDICE DE TABELAS

Tabela 2.1	Ambientes de Passagem de Mensagens.	30
Tabela 3.1	Comparação das Características dos Protocolos Conservativos e Otimistas.	71
Tabela 3.2	Linguagens para Simulação Distribuída.	73
Tabela 3.3	Comparação entre as Diferentes Linguagens para Simulação Distribuída.	74
Tabela 3.4	Bibliotecas para Simulação Distribuída.	74
Tabela 3.5	Principais Características das Bibliotecas para Simulação Distribuída.	76
Tabela 4.1	Mecanismos para Avaliação de Desempenho.	81
Tabela 4.2	Descrição das Estruturas de Dados do *SMPLX* – Recurso.	98
Tabela 4.3	Descrição das Estruturas de Dados do *SMPLX* - Lista_Servidores.	98
Tabela 4.4	Descrição das Estruturas de Dados do *SMPLX* – Fila_Eventos.	98
Tabela 4.5	Descrição - Lista de Eventos Futuros.	99
Tabela 4.6	Execução - Servidor Central.	100
Tabela 4.7	Execução – Sistema Computacional Simplificado.	100
Tabela 4.8	Estruturas de Dados dos Processos Lógicos no *ParSMPL*.	104
Tabela 4.9	Tabela de Identificação de Processos.	108
Tabela 4.10	Validação do *ParSMPL* – Servidor Central.	123
Tabela 4.11	Validação do *ParSMPL* – Sistema Computacional Simplificado.	123
Tabela 5.1	Características: *IBM SP2.*	131
Tabela 5.2	Características: *IBM RISC System/6000.*	131
Tabela 5.3	Identificação dos Modelos.	163
Tabela 6.1	Tempo de Transmissão de Mensagens.	170
Tabela 6.2	Resultados da Execução dos Modelos.	177
Tabela 7.1	*Speedup –Trice* (Partição dos Processos Lógicos).	202
Tabela 7.2	*Speedup* – Sistema Computacional (Partição dos Processos Lógicos).	204
Tabela 7.3	*Speedup* – Modelo Hipotético 1 (Partição dos Processos Lógicos).	207
Tabela 7.4	*Speedup* – Modelo Hipotético 2 (Partição dos Processos Lógicos).	208
Tabela 7.5	*Speedup* – Modelo Hipotético 4 (Partição dos Processos Lógicos).	210
Tabela 7.6	*Speedup* – Modelo Hipotético 5 (Partição dos Processos Lógicos).	212
Tabela 7.7	*Speedup – Trice* (Partição dos Processos Lógicos).	215
Tabela 7.8	*Speedup* – Servidor Central (Granulosidade e Número de Processos	218

Lógicos).

Tabela 7.9 *Speedup* – Sistema Computacional (Granulosidade e Número de Processos Lógicos). 220

Tabela 7.10 *Speedup* – Modelo Hipotético 3 (Granulosidade e Número de Processos Lógicos). 224

Tabela 7.11 *Speedup* – Sistema Computacional Simplificado (Tempo Entre Chegadas de Tarefas). 232

Tabela 7.12 *Speedup* – Sistema de Computação com Processamento por Transação (Tempo Entre Chegadas de Tarefas). 234

Tabela 7.13 *Speedup* – Sistema Computacional Simplificado (Tempo de Serviço dos Recursos). 237

Tabela 7.14 *Speedup* – Servidor Central (1) (Número de Tarefas). 240

Tabela 7.15 *Speedup* – Servidor Central (2) (Número de Tarefas). 241

Tabela 7.16 *Speedup* – Modelo Hipotético 4. 245

Tabela 7.17 Estatísticas - Modelo Hipotético 4. 246

Tabela 7.18 Diretrizes para Desenvolvimento da Simulação Distribuída Conservativa. 248

LISTA DE ABREVIATURAS

ASiA	Ambiente de Simulação Automático
ATM	*Asynchronous Transfer Mode*
CMB	*Chandy-Misra-Bryant*
CISC	Centro de Informática de São Carlos
CPU	*Central Processing Unity*
E/S	Entrada/Saída
FDDI	*Fiber Distributed Data Interface*
FIFO	*First In, First Out*
GVT	*Global Virtual Time*
ICMC	Instituto de Ciências Matemáticas e de Computação
IP	*Internet Protocol*
LEF	Lista de Eventos Futuros
LVT	*Local Virtual Time*
MIMD	*Multiple Instruction Stream, Multiple Data Stream*
MISD	*Multiple Instruction Stream, Single Data Stream*
MPCI	*Message Passing Client Interface*
MPI	*Message Passing Interface*
MPMD	*Multiple Program, Multiple Data*
MPP	*Massively Parallel Processor*
PFE	Processador *Front-End*
PVM	*Parallel Virtual Machine*
RPC	*Remote Procedure Call*
SIMD	*Single Instruction Stream, Multiple Data Stream*
SISD	*Single Instruction Stream, Single Data Stream*
SP2	*IBM Risc Sytem/6000 Scalable POWERparallel System*
SPEEDS	*Synchronous Parallel Environment for Emulation and Discrete Event Simulation*
SPMD	*Single Program, Multiple Data*
TCP	*Transmission Control Protocol*
TRICE	*Transportable Integrated Computing Environment*
UDP	*User Datagrama Protocol*
USP	Universidade de São Paulo
US	*User Space*
VLSI	*Very Large Scale Integration*
XDR	*EXternal Data Representation*

RESUMO

Este trabalho apresenta um estudo sobre a viabilidade de utilização do protocolo conservativo *CMB* para sincronização de simulações distribuídas em diferentes plataformas *MIMD* com memória distribuída, utilizando granulosidade grossa com poucos processos paralelos. A técnica utilizada para a realização das análises consiste em efetuar coleta de dados na execução da simulação de um grande número de modelos. Para a simulação desses modelos foi desenvolvida, e é apresentada neste trabalho, a extensão funcional *ParSMPL*, que implementa o protocolo de sincronização *CMB*.

Pode-se subdividir os resultados obtidos com a análise dos dados das simulações em enfoques que apresentam contribuições distintas. O primeiro enfoque avalia como a plataforma adotada e o modelo a ser simulado influenciam no *speedup* alcançado. Neste caso, define-se quando uma aplicação tem tendência a ser eficiente com o paradigma da simulação distribuída utilizando o protocolo *CMB*.

O segundo enfoque refere-se à necessidade do usuário conhecer a melhor forma de se utilizar a simulação distribuída. A partir das análises realizadas foi estabelecido um conjunto de diretrizes que visam auxiliar o processo de desenvolvimento de uma simulação distribuída conservativa. Com a ajuda dessas diretrizes e a disponibilidade da extensão *ParSMPL* espera-se auxiliar o usuário a desenvolver aplicações eficientes utilizando o paradigma da simulação distribuída conservativa, sem exigir desse usuário o conhecimento das características e particularidades do protocolo de sincronização utilizado.

ABSTRACT

This thesis approaches a study about the viability of using the CMB conservative protocol for distributed simulation synchronization on different distributed memory MIMD platforms and considering coarse granularity and few parallel processes. The technique used to analyze the results comprises data acquisition during simulation execution for a large number of models. The simulation of these models is performed on a special-built distributed simulation environment (ParSMPL) developed and presented in this work, being a CMB synchronization protocol implementation.

The results obtained in this work are split according to different views taking to distinct sets of contributions. The first view evaluates the influence of the model and the execution platform on the speedup reached. In this case it is defined when an application can reach efficiency through the adoption of the distributed simulation paradigm using the CMB protocol.

The second view refers to the need of the users knowing the best way to make use of distributed simulation. Thus, following the analysis performed in this thesis, it is established a set of procedures to help in the distributed simulation development process adopting the conservative approach. By following the procedures proposed and using the ParSMPL an user can hope precious help in the development of efficient conservative distributed simulation programs, without the need of knowing the features and particularities of the CMB protocol.

CAPÍTULO 1

INTRODUÇÃO

O crescente avanço tecnológico que possibilitou o desenvolvimento de máquinas com grande poder computacional e redes de comunicação de alta velocidade constitui um fator determinante para a popularização do paradigma do processamento paralelo. A motivação principal para a utilização desse paradigma é a possibilidade de aumento do desempenho na execução de diferentes tipos de aplicações, quando comparadas com o paradigma tradicional da computação seqüencial.

Esse crescente desenvolvimento das arquiteturas de computadores vem tornando imprescindível a avaliação de desempenho, e diversas ferramentas tais como protótipos, *benchmarks* e coleta de dados têm sido utilizadas para esse fim. Porém, quando o sistema que se deseja avaliar constitui um projeto, ou mesmo quando as experimentações com o sistema são inviáveis, as técnicas de aferição não podem ser

utilizadas. Nessas situações as técnicas de modelagem permitem realizar a avaliação desejada, e a simulação surge como uma ferramenta adequada para esse fim.

Entretanto, na avaliação de modelos complexos, o tempo de execução necessário para a obtenção de resultados confiáveis pode tornar a utilização da simulação tradicional (seqüencial) inviável. Com a união das áreas de simulação e de computação paralela foi possível o desenvolvimento de novos paradigmas, tais como a simulação distribuída e a técnica de múltiplas replicações em paralelo, que surgem como alternativas para o desenvolvimento de aplicações complexas que requerem resultados precisos e, portanto, demandam grande poder computacional.

Contudo, o processo de migração da simulação seqüencial para o paradigma distribuído exige não apenas uma adaptação das estruturas utilizadas na simulação, mas também novos mecanismos para garantir a execução correta dos eventos da simulação. Ou seja, como na simulação seqüencial, o mecanismo da simulação distribuída deve garantir que os eventos sejam executados em ordem não decrescente do tempo de ocorrência dos eventos. Isso implica na inclusão de protocolos para garantir a sincronização entre os processos paralelos e evitar a ocorrência dos chamados erros de causa e efeito.

A técnica de simulação utilizando múltiplas replicações em paralelo constitui uma alternativa à complexidade imposta pela necessidade de protocolos de sincronização na simulação distribuída. Porém, a utilização desse método exige o estudo de técnicas estatísticas a serem empregadas para a análise dos resultados obtidos.

Basicamente, os protocolos de sincronização são divididos em dois grupos, os protocolos conservativos e os protocolos otimistas. Os protocolos conservativos

evitam a ocorrência dos erros de causa e efeito, enquanto os otimistas permitem a ocorrência desses erros, e utilizam um mecanismo para retornar a um estado seguro.

Uma grande variedade de plataformas computacionais, englobando desde máquinas paralelas virtuais a arquiteturas verdadeiramente paralelas, vem sendo utilizada para a execução da simulação distribuída, abrangendo diferentes domínios de aplicações e protocolos de sincronização. O enfoque deste trabalho está voltado para os protocolos conservativos, particularmente o protocolo *CMB*, considerando plataformas *MIMD* com memória distribuída e explorando aplicações com granulosidade grossa, constituídas por poucos processos lógicos. Para o desenvolvimento do trabalho foram considerados modelos representativos de diferentes tipos de sistemas computacionais.

Nos protocolos conservativos diferentes fatores exercem influência no desempenho da simulação. Assim, para alcançar um bom desempenho em uma simulação distribuída, o usuário deve não apenas conhecer profundamente o sistema a ser simulado e seu modelo de forma a explorar todo o paralelismo, mas também conhecer as características arquiteturais da plataforma utilizada, para que seja possível avaliar o compromisso entre balanceamento de carga e comunicação. Todos esses requisitos dificultam o trabalho do usuário no desenvolvimento da simulação distribuída, em particular com a utilização de protocolos conservativos.

Desta forma, o objetivo deste trabalho consiste em analisar o comportamento do protocolo conservativo *CMB*, abrangendo dois enfoques principais. O primeiro enfoque consiste em avaliar o impacto das características da plataforma utilizada e do modelo desenvolvido no desempenho da simulação distribuída, estabelecendo uma metodologia que possibilita prever a tendência do comportamento da simulação em

função dessas características. O segundo enfoque consiste em analisar a influência de diferentes fatores no desempenho, como a forma de partição do modelo e o número de processos lógicos, de maneira a determinar as condições que possibilitem maximizar o desempenho da simulação.

Assim, é estabelecido um conjunto de diretrizes que possibilita auxiliar o usuário no desenvolvimento de aplicações eficientes utilizando o paradigma da simulação distribuída conservativa, sem exigir desse usuário o conhecimento das características e particularidades do protocolo de sincronização utilizado. Conhecendo as características arquiteturais da plataforma a ser utilizada e do modelo a ser desenvolvido, é possível para o usuário analisar a tendência do comportamento da simulação e otimizar o desempenho da aplicação em função das diretrizes estabelecidas.

A metodologia adotada para o desenvolvimento do trabalho é baseada na utilização de técnicas de aferição, que possibilitam a obtenção de resultados precisos por meio da execução de *benchmarks* e da coleta de dados efetuada diretamente na execução da aplicação. Ressalta-se que a coleta dos dados foi realizada de forma a não interferir na execução das simulações.

A técnica de prototipação foi empregada com o desenvolvimento de duas extensões funcionais para simulação, *SMPLX* e *ParSMPL*, em sistema operacional *UNIX*. *SMPLX* oferece as primitivas necessárias para o desenvolvimento de uma simulação discreta orientada a eventos. *ParSMPL* utiliza os ambientes de passagem de mensagens *PVM* e *PVMe* para implementar as primitivas necessárias para uma simulação distribuída conservativa com o protocolo *CMB*.

Para a realização das análises foram desenvolvidos modelos representativos de diferentes tipos de sistemas computacionais, constituindo casos de testes que podem ser utilizados como referência para a avaliação de desempenho de diversos tipos de protocolos de sincronização em simulação distribuída.

Os capítulos 2 e 3 desta tese apresentam uma revisão bibliográfica sobre tópicos importantes dentro do contexto do trabalho desenvolvido. O capítulo 2 faz uma revisão sobre conceitos de computação paralela e ambientes de passagem de mensagens. O capítulo 3 apresenta uma revisão sobre conceitos de simulação discreta e simulação distribuída, descrevendo as principais características dos protocolos de sincronização conservativos e otimistas. No final do capítulo é apresentada uma comparação entre esses protocolos e um resumo das principais linguagens e extensões funcionais utilizadas atualmente para o desenvolvimento da simulação distribuída.

As principais características das extensões *ParSMPL* e *SMPLX* são descritas no capítulo 4. No capítulo 5 são apresentados os modelos de redes de filas desenvolvidos, considerando diferentes tipos de sistemas computacionais, e as plataformas utilizadas neste trabalho.

Os capítulos seguintes descrevem as análises realizadas para a avaliação do comportamento do protocolo *CMB*. O capítulo 6 apresenta um estudo sobre o desempenho obtido para a simulação distribuída considerando diferentes modelos e as influências das características da plataforma e do modelo no desempenho da simulação. Em função das análises realizadas descreve-se uma metodologia que possibilita verificar a tendência do comportamento da simulação distribuída. O capítulo 7 mostra uma análise sobre os fatores determinantes para o desempenho da simulação, considerando as características da aplicação e a configuração dos

processos lógicos. Nesse capítulo são apresentadas também as diretrizes estabelecidas para o desenvolvimento de uma simulação distribuída conservativa eficiente.

O capítulo 8 encerra o trabalho, apresentando as contribuições resultantes, as conclusões obtidas e os planos para continuidade do trabalho.

CAPÍTULO 2

COMPUTAÇÃO PARALELA E AMBIENTES DE PASSAGEM DE MENSAGENS

Este capítulo apresenta os conceitos básicos de computação paralela, ambientes de passagem de mensagens e as principais características do PVM e do *IBM PVMe.*

2.1 Introdução

A idéia de incluir paralelismo nos sistemas computacionais é tão antiga quanto o próprio desenvolvimento dos computadores. Trabalhos desenvolvidos por von Neumann na década de 40 já discutiam a possibilidade de se utilizar algoritmos paralelos para a solução de equações diferenciais. A primeira máquina paralela a ser projetada e implementada foi o ILLIAC IV, desenvolvida na década de 60 na Universidade de Illinois, Estados Unidos, constituída por 64 processadores.

A computação de alto desempenho tem apresentado um aumento crescente na utilização em diversas áreas como por exemplo análise estrutural, simulação,

sensoriamento remoto e automação industrial, uma vez que oferece o poder computacional necessário para solucionar esses problemas complexos. Sem o poder computacional dos computadores atuais, muitos desses problemas não poderiam ser solucionados em períodos de tempo adequados. O advento da computação paralela possibilitou não apenas que problemas complexos pudessem ser solucionados, como também meios de se obter maior desempenho na execução dessas aplicações. As arquiteturas paralelas representam o futuro no projeto de sistemas computacionais [Hwang; Xu 1998].

Hwang e Briggs [Hwang; Briggs 1984] definem processamento paralelo como a forma eficiente do processamento de informações, com ênfase na exploração de eventos concorrentes no processo computacional. O processamento paralelo implica na divisão de uma determinada aplicação em partes, de maneira que essas partes possam ser executadas simultaneamente, por vários elementos de processamento. Esses elementos de processamento devem cooperar entre si (comunicação e sincronismo), buscando eficiência, por meio da quebra do paradigma de execução seqüencial do fluxo de instruções, ditado pela filosofia de von Neumann.

Vários fatores justificam a necessidade do processamento paralelo, e a busca por maior desempenho é o principal entre esses fatores [Almasi; Gottlieb 1994, Quealy 1994]. As diversas áreas nas quais a computação se aplica requerem poder computacional cada vez maior, em virtude dos algoritmos complexos que são utilizados e do tamanho do conjunto de dados a ser processado. Além desse fator, outros também motivaram o desenvolvimento da computação paralela [Almasi; Gottlieb 1994, Amorim et al. 1988]:

- O desenvolvimento tecnológico, principalmente o surgimento da tecnologia *VLSI* (*Very Large Scale Integration*) de projeto de microprocessadores, permitiu a construção de microprocessadores de alto desempenho que agrupados possibilitavam um ganho significativo de poder computacional e uma melhor relação custo/desempenho, quando comparado aos supercomputadores de custo elevado [Zaluska 1991];
- A configuração modular permite que vários elementos de processamento sejam agrupados de acordo com as características da aplicação e a expansão com a inclusão de novos elementos de processamento;
- A possibilidade de desenvolvimento de sistemas tolerantes a falhas, por intermédio da redundância de *hardware*.

Além disso, várias aplicações são inerentemente paralelas, e transformá-las em seqüenciais pode implicar na perda de desempenho e no aumento da complexidade dos algoritmos, o que justifica a necessidade de novas maneiras de organização do processamento computacional [Almasi; Gottlieb 1994].

Os conceitos básicos sobre programação paralela e ambientes de passagem de mensagens são apresentados nas seções a seguir.

2.2 Conceitos Básicos Sobre Computação Paralela

Nesta seção são apresentados conceitos básicos sobre tipos de paralelismo, granulosidade, *speedup* e eficiência.

ICMC-USP BIBLIOTECA INFORMAÇÃO

2.2.1 Tipos de Paralelismo

Existe execução concorrente de eventos sempre que, em um determinado instante, dois ou mais eventos tiverem sido iniciados mas ainda não terminados, o que significa que estão em andamento. Os eventos em execução podem apresentar paralelismo real ou pseudo-paralelismo [Quinn 1987]. A figura 2.1 ilustra esses tipos de concorrência e a possibilidade da existência de pseudo-paralelismo e paralelismo real em conjunto.

No pseudo-paralelismo tem-se a ocorrência intercalada de partes de eventos concorrentes, de forma que em qualquer instante, apenas um evento estará em andamento no único recurso disponível. No paralelismo real tem-se a ocorrência sobreposta dos eventos. Todos os eventos paralelos estão em andamento simultaneamente, utilizando os múltiplos recursos existentes.

O paralelismo real pode ser classificado em: espacial, temporal (*pipeline*) ou combinado. O paralelismo espacial está ligado à sobreposição de execução de tarefas, podendo ser subdividido em síncrono ou assíncrono. No paralelismo síncrono vários elementos de processamento executam funções idênticas e são sincronizados por uma única unidade de controle. No paralelismo assíncrono os elementos de processamento executam funções diferentes e interagem entre si. O paralelismo temporal está relacionado com a execução de tarefas como uma cascata ou estágios de sub-tarefas, de forma que execuções diferentes ocorram com sobreposição e defasagem no tempo. A combinação do paralelismo espacial com o paralelismo temporal (paralelismo combinado) permite a execução sobreposta de vários conjuntos de tarefas que, por sua vez, são executadas com defasagem no tempo.

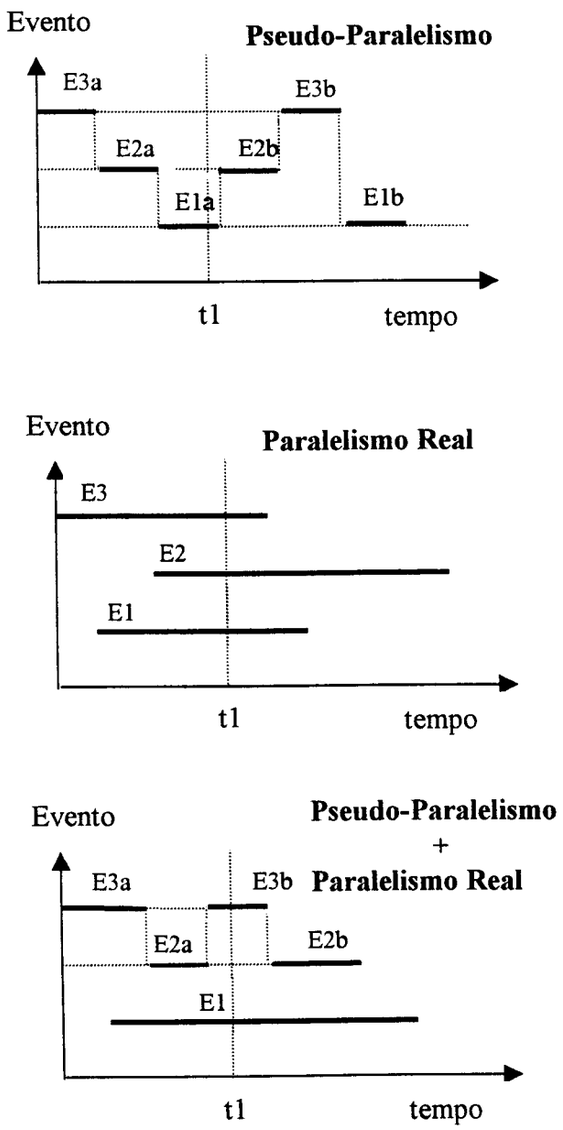

Figura 2.1 - Tipos de Paralelismo. No psedo-paralelismo a execução dos eventos compete pela utilização de recursos. No paralelismo real os eventos são executados em paralelo, utilizando múltiplos recursos. O paralelismo misto permite uma combinação desses dois tipos de paralelismo.

O termo paralelismo é normalmente utilizado para designar paralelismo real. Já o termo concorrência é empregado em casos nos quais se tem mais de um processo iniciado e não ainda terminado. Isso não implica, necessariamente, na existência de

mais de um elemento de processamento. Desta forma, o paralelismo é um caso especial de concorrência, no qual existem processos sendo executados utilizando múltiplos elementos de processamento.

2.2.2 Granulosidade

A granulosidade de uma aplicação paralela corresponde ao tamanho das unidades de trabalho submetidas aos elementos de processamento [Hwang 1993]. Basicamente, a granulosidade pode ser classificada em grossa, média ou fina.

A granulosidade grossa está relacionada com paralelismo ao nível de processos e programa (pequeno número de elementos de processamento complexos), enquanto a granulosidade fina corresponde ao paralelismo ao nível de instruções ou operações (grande número de elementos de processamento simples) [Navaux 1989]. A granulosidade média é uma classificação intermediária entre fina e grossa.

2.2.3 *Speedup* (*Sp*)

O principal objetivo na utilização da computação paralela é a diminuição do tempo de execução dos processos envolvidos. Diferentes métricas podem ser utilizadas para determinar se a utilização do processamento paralelo está sendo vantajosa e quantificar o desempenho alcançado. O *speedup* é uma das métricas mais utilizadas para atingir esse objetivo, sendo definido como o aumento de velocidade observado em um computador paralelo com p elementos de processamento, em relação a um computador seqüencial, como apresentado pela Eq. (2.1) [Hwang; Xu 1998].

$$Sp = Ts/Tp \qquad (2.1)$$

Onde:

Sp = *speedup* observado em um computador com *p* elementos de processamento;

Ts = tempo de execução do programa em um único elemento de processamento;

Tp = tempo de execução do mesmo programa em um computador paralelo, com *p* elementos de processamento.

Nos casos em que se atinge uma distribuição perfeita dos processos a serem executados em paralelo, obtém-se o valor máximo de *speedup*, ou seja, $Sp = p$ (nesse caso, $Tp = Ts/p$). Na prática, diversos fatores influenciam no valor do *speedup* alcançado, principalmente no sentido de torná-lo menor. Alguns fatores que diminuem o *speedup* são: sobrecarga de comunicação e de sincronismo, balanceamento de carga e algoritmos ineficientes para explorar o paralelismo do problema a ser solucionado.

Em alguns casos, o *speedup* obtido pode ser maior que o previsto, ou seja, $Sp > p$. Esse fenômeno, denominado anomalia de *speedup* ou *speedup* super linear, pode ser causado pelas características do algoritmo paralelo ou pelas características da plataforma de *hardware* (utilização da memória *cache* dos elementos de processamento) [Foster 1995, Quinn 1987].

2.2.4 Eficiência (*E*)

A eficiência representa a porcentagem dos elementos de processamento que estão sendo efetivamente utilizados [Ayani 1993, Hwang; Xu 1998] (Eq. (2.2)).

$$E = speedup/p * 100 \quad (2.2)$$

Onde:

p = número de elementos de processamento.

2.3 Arquiteturas Paralelas

Tomando como base as características de uma arquitetura paralela é possível classificá-la, utilizando-se diferentes enfoques. A classificação de Flynn, proposta na década de 60, é a mais conhecida, embora apresente alguma dificuldade em abranger todas as arquiteturas atuais. A classificação de Duncan é mais abrangente, porém menos conhecida. Essas classificações são descritas nas seções 2.3.1 e 2.3.2.

2.3.1 Classificação de Flynn

A classificação de Flynn é baseada na relação entre o fluxo de instruções e o fluxo de dados. Um fluxo de instruções refere-se a uma seqüência de instruções executadas em um elemento de processamento, sobre um fluxo de dados aos quais essas instruções estão relacionadas. Nessa classificação são definidas quatro classes para arquiteturas de computadores [Duncan 1990, Hwang; Briggs 1984, Navaux 1989]:

- ***SISD*** (*Single Instruction Stream, Single Data Stream*) fluxo único de instruções e dados (figura 2.2). São as máquinas baseadas no princípio de

von Neumman. As instruções são executadas em seqüência, mas podem ser sobrepostas nos seus estágios de execução (*pipeline*);

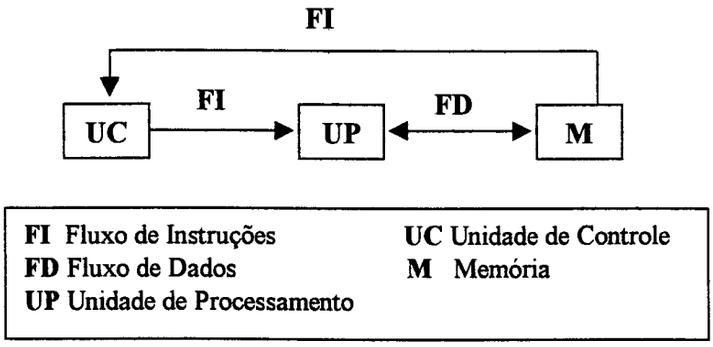

Figura 2.2 - Modelo Computacional *SISD*. Arquitetura constituída por um fluxo único de instruções e dados. As instruções são executadas seqüencialmente.

- ***SIMD*** (*Single Instruction Stream, Multiple Data Stream*): fluxo de instruções único com vários fluxos de dados (figura 2.3). Envolve múltiplos elementos de processamento que executam, simultaneamente, a mesma seqüência de instruções sobre dados diferentes;

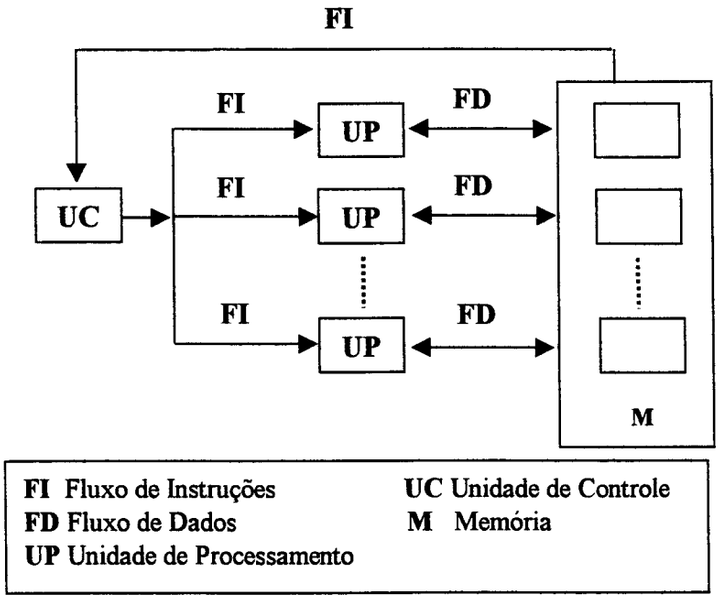

Figura 2.3 - Modelo Computacional *SIMD*. Arquitetura constituída por um fluxo de instruções único, com vários fluxos de dados. Nesse tipo de arquitetura, a mesma seqüência de instruções é executada simultaneamente sobre dados distintos.

- ***MISD*** (*Multiple Instruction Stream, Single Data Stream*): múltiplo fluxo de instruções e um único fluxo de dados. Corresponderia a determinada máquina que, com vários elementos de processamento, executa instruções distintas sobre um mesmo conjunto de dados (figura 2.4). Essa estrutura não existe na prática, sendo considerada sem validade por alguns autores (outros autores consideram o *pipeline* como representante dessa categoria);

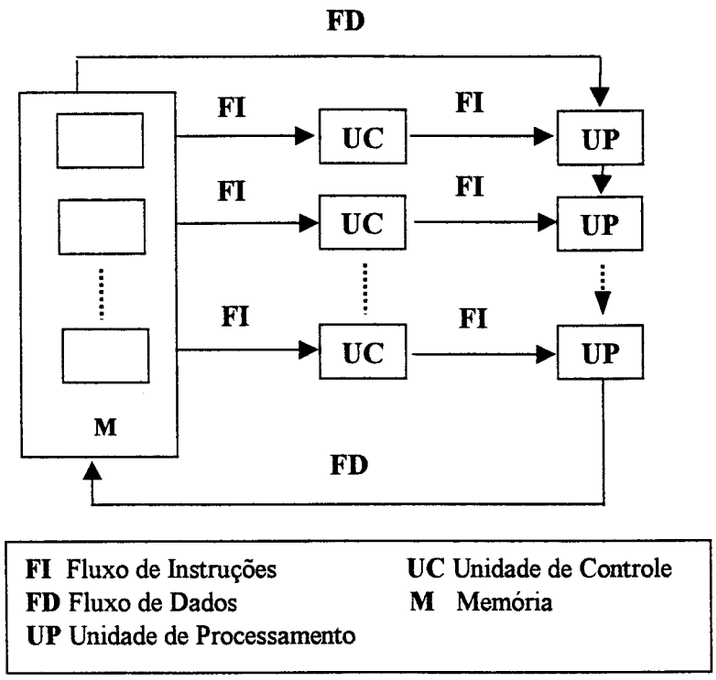

Figura 2.4 - Modelo Computacional *MISD*. É constituído por diversos fluxos de instruções e um único fluxo de dados. Corresponderia a determinada máquina que executa instruções distintas sobre um mesmo conjunto de dados.

- ***MIMD*** (*Multiple Instruction Stream, Multiple Data Stream*): múltiplos fluxos de instruções e de dados (figura 2.5). Envolve múltiplos elementos de processamento autônomos, os quais executam diferentes instruções sobre conjuntos de dados distintos.

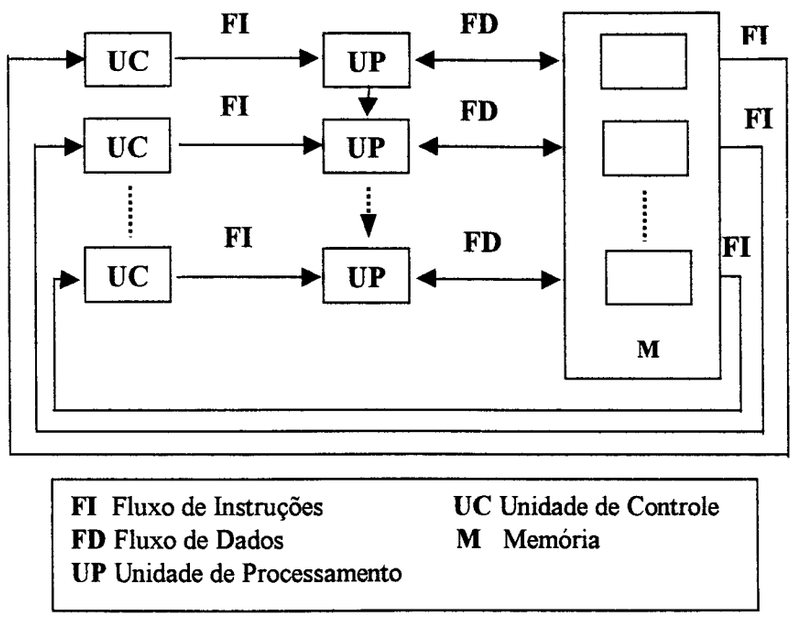

Figura 2.5 - Modelo Computacional *MIMD*. Engloba as arquiteturas constituídas por múltiplos fluxos de instruções e de dados.

A classificação de Flynn foi considerada insuficiente por não abranger algumas arquiteturas modernas, como as máquinas de fluxo de dados e os processadores vetoriais, além de não apresentar uma hierarquia (a classificação *MIMD* engloba quase todas as arquiteturas paralelas sem apresentar subníveis) [Santana et al. 1997]. Duncan [Duncan 1990] propôs uma nova classificação, mais complexa e abrangente, descrita na seção 2.3.2.

2.3.2 Classificação de Duncan

A classificação de Duncan está dividida em dois grupos principais, arquiteturas síncronas e arquiteturas assíncronas [Duncan 1990]. Essa classificação é apresentada na figura 2.6 e detalhada nas seções 2.3.2.1 e 2.3.2.2.

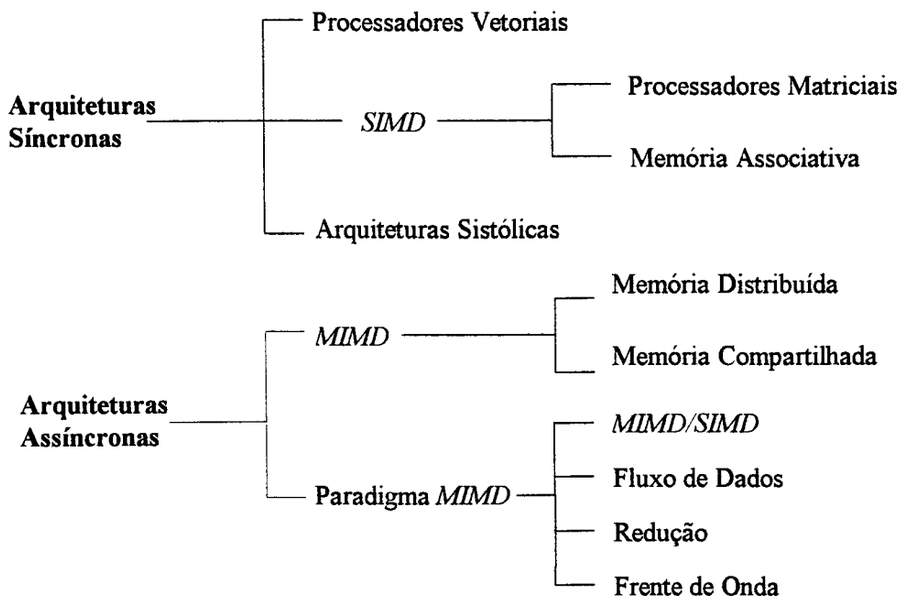

Figura 2.6 - Classificação de Duncan para Arquiteturas Paralelas. Considerada mais abrangente que a classificação de Flynn, permite a inclusão de diversos tipos de arquiteturas.

2.3.2.1 Arquiteturas Síncronas

São arquiteturas paralelas que coordenam as operações por meio de relógios globais, unidade de controle central ou controladores de unidades vetoriais. Essas arquiteturas são divididas em três classes:

- **Processadores Vetoriais**: são caracterizados por unidades funcionais que implementam operações sobre vetores;
- **Arquiteturas *SIMD***: apresentam uma única unidade de controle e múltiplos elementos de processamento. São divididas em duas categorias:
  - **Processadores Matriciais**: desenvolvidos para aplicações específicas e possibilitam operações sobre matrizes de dados;

- **Memória Associativa**: arquiteturas que utilizam comparações lógicas para recuperar, em paralelo, dados armazenados na memória, de acordo com seu conteúdo;

- **Arquiteturas Sistólicas**: foram propostas para resolver problemas que exigem computação intensa, com grande quantidade de operações de E/S (Entrada/Saída). São constituídas por vários elementos de processamento organizados de forma a constituir um *pipeline*, que formam uma cadeia, na qual apenas os elementos de processamento localizados nos limites dessa estrutura possuem comunicação com a memória [Santana et al. 1997].

2.3.2.2 Arquiteturas Assíncronas

As arquiteturas assíncronas são caracterizadas por um controle de *hardware* descentralizado, o que possibilita aos múltiplos elementos de processamento serem independentes. Esses elementos de processamento trabalham com conjunto de dados distintos. As arquiteturas assíncronas são classificadas em:

- **Arquiteturas *MIMD*:** são constituídas por múltiplos elementos de processamento independentes, que executam diferentes conjuntos de instruções, sobre conjuntos de dados distintos. Esse tipo de arquitetura oferece grande flexibilidade para a construção de algoritmos paralelos [Blech 1994]. As arquiteturas *MIMD* podem ser divididas em duas categorias, dependendo do grau de interação entre os elementos de processamento:

- **Memória Compartilhada (multiprocessadores)**: caracterizam-se pela existência de uma memória global e única, a qual é utilizada por todos os elementos de processamento. A comunicação e a sincronização entre os diversos elementos é realizada por intermédio do compartilhamento de posições dessa memória [Santana et al. 1997, Tanenbaum 1995];
- **Memória Distribuída (multicomputadores)**: cada elemento de processamento possui sua própria memória local, e toda a comunicação e a sincronização são realizadas por meio da troca de mensagens [Tanenbaum 1995];

- **Paradigma *MIMD*:** engloba as arquiteturas assíncronas que, apesar de apresentarem as características de uma arquitetura *MIMD*, possuem também características próprias, que dificultam a sua classificação como puramente *MIMD*:
  - ***MIMD/SIMD* (híbridas)**: são caracterizadas pelo fato de algumas partes da arquitetura *MIMD* estarem sob o controle *SIMD*;
  - **Fluxo de Dados**: nessas arquiteturas uma instrução é executada assim que os dados necessários (operandos) se tornam disponíveis. Desta forma, a seqüência de instruções é baseada apenas na dependência entre os dados;
  - **Redução (dirigidas à demanda):** nesse tipo de arquitetura uma instrução é habilitada para execução quando seu resultado

é solicitado como operando de outra instrução, também habilitada para execução;

- **Frente de Onda:** combinam uma estrutura sistólica de elementos de processamento com o paradigma de execução de fluxo de dados.

Apesar do desenvolvimento de máquinas complexas e com grande poder computacional, a utilização do processamento paralelo foi restrita durante muitos anos. Esse fato é justificado pelo lento desenvolvimento de ferramentas de *software*, dificultando e afastando o usuário comum desse tipo de processamento. O desenvolvimento de ferramentas como compiladores paralelizadores e bibliotecas possibilitou a mudança desse cenário. A seção 2.4 apresenta os paradigmas para o desenvolvimento de aplicações paralelas.

2.4 Paradigmas de Programação Paralela

[Almasi; Gottlieb 1994] definem os seguintes tipos de ferramentas para o desenvolvimento de programas paralelos: compiladores paralelizadores, extensões paralelas para linguagens seriais (bibliotecas), compiladores especiais e linguagens concorrentes.

- **Compiladores paralelizadores**: geram automaticamente versões paralelas de programas não paralelos. Esses compiladores exigem o mínimo de trabalho do usuário, mas o desempenho obtido normalmente é modesto. Além disso, nem sempre esses compiladores estão disponíveis, principalmente para sistemas de memória distribuída. [Blech 1994,

Hwang; Xu 1998]. Como exemplo, pode-se citar *Parafrase* [Almasi; Gottlieb 1994];

- **Extensões paralelas**: são bibliotecas cujo conjunto de instruções, aliado com as primitivas da linguagem hospedeira, permite o desenvolvimento de aplicações paralelas. Possuem como vantagem o fato de não exigirem do usuário o aprendizado de uma nova linguagem. Normalmente apresentam um desempenho melhor quando comparadas com os compiladores paralelizadores. Como exemplo tem-se *PVM* (*Parallel Virtual Machine*) e *MPI* (*Message Passing Interface*) (para arquiteturas de memória distribuída);
- **Compiladores especiais**: permitem a extensão de linguagens seqüenciais com ferramentas para o desenvolvimento de aplicações paralelas, como processos e monitores. Como exemplo pode-se citar Pascal Concorrente (para arquiteturas de memória compartilhada);
- **Linguagens concorrentes**: são linguagens criadas especificamente para esse fim. Apresentam a desvantagem de exigir do usuário o aprendizado de uma nova linguagem de programação. Por outro lado, podem oferecer melhor desempenho para as aplicações. Como exemplo podem ser citadas as linguagens *Occam* e *Ada*.

A escolha de qual ferramenta utilizar deve ser feita considerando diferentes aspectos, como o tipo de aplicação, o perfil do usuário que utilizará a ferramenta e a arquitetura que executará os códigos gerados. Dois aspectos são fundamentais na

escolha: o tempo de trabalho do usuário e o desempenho obtido [Almasi; Gottlieb 1994].

Definida a ferramenta a ser usada, deve-se determinar o paradigma a ser utilizado para o desenvolvimento da aplicação paralela, ou seja, especificar o modelo de programação que descreve o método por meio do qual os elementos de processamento são coordenados na execução de suas tarefas:

- **Troca de mensagens**: os diversos processos comunicam e sincronizam enviando e recebendo mensagens;
- **Memória compartilhada**: múltiplos elementos de processamento compartilham uma memória única.

O paradigma de passagem de mensagens para o desenvolvimento de aplicações paralelas pode ser subdividido em duas classes: *SPMD* e *MPMD* [Geist et al. 1994a]:

- ***SPMD*** (*Single Program, Multiple Data*): nesse modelo um único programa executa em mais de um elemento de processamento, ou seja, todos os elementos de processamento executam o mesmo código, porém sobre um conjunto de dados distinto;
- ***MPMD*** (*Multiple Program, Multiple Data*): neste caso programas diferentes executam em elementos de processamentos distintos (cada elemento de processamento com seu próprio conjunto de dados e código).

O paradigma de troca de mensagens oferece uma forma simples e principalmente adaptável em qualquer plataforma para o desenvolvimento de aplicações paralelas. Desta forma, ressalta-se a importância da programação por meio

da troca de mensagens e dos ambientes de passagem de mensagens, que se destinam a fornecer maneiras de se utilizar sistemas distribuídos como ambientes de programação paralela, e assim fornecer plataformas flexíveis e de menor custo para a construção e execução de algoritmos paralelos [Souza 1996]. A programação via troca de mensagens e os ambientes de passagem de mensagens são discutidos na seção 2.5.

2.5 Programação Paralela Utilizando Ambientes de Passagem de Mensagens

O modelo computacional *MIMD* tem obtido destaque no desenvolvimento de aplicações paralelas devido a sua flexibilidade, além da sua potencialidade na execução de programas paralelos de granulosidade média e grossa. Entre as plataformas *MIMD* destacam-se as plataformas paralelas de memória distribuída, que podem ser computadores paralelos ou máquinas paralelas virtuais (sistema distribuído ou conjunto de máquinas interligadas funcionando como máquina paralela virtual) [Blech 1994, Mcbryan 1994, Zaluska 1991].

Na computação paralela é fundamental a cooperação (comunicação e sincronismo) entre os elementos de processamento que trabalham em conjunto na execução de uma aplicação. Com memória compartilhada, a comunicação e a sincronização são realizadas por meio de espaços de memória compartilhados entre os diversos processos paralelos. Utilizando memória distribuída, devem ser definidas primitivas explícitas que possibilitem que os processos se comuniquem e se sincronizem [Almasi; Gottlieb 1994, Andrews; Schneider 1983, Quinn 1987]. O paradigma de troca de mensagens define então um conjunto de primitivas que permite a comunicação e a sincronização entre os processos por via da troca de mensagens.

2.5.1 Computação Paralela em Sistemas Distribuídos

Em arquiteturas paralelas todos os elementos de processamento são iguais em termos de capacidade, recursos, *software* e velocidade de comunicação. Porém, em um sistema distribuído, podem ser encontradas dificuldades no desenvolvimento de programas paralelos em função de diferentes tipos de heterogeneidade [Geist et al. 1994a]:

- **Arquitetura**: o conjunto de computadores disponíveis pode incluir uma grande variedade de tipos de arquiteturas, como por exemplo computadores pessoais, estações de trabalho de alto desempenho, multiprocessadores, computadores vetoriais e arquiteturas maciçamente paralelas;
- **Formato de dados**: computadores diferentes podem apresentar formatos de dados distintos, ou seja, incompatíveis, o que exige algum tipo de padronização;
- **Potência computacional**: em um ambiente virtual composto por diferentes tipos de arquiteturas deve-se garantir que nenhuma das máquinas fique ociosa, o que pode levar a um baixo desempenho;
- **Carga de trabalho na rede e em cada máquina**: com muitos usuários a carga de trabalho imposta à rede e até mesmo às máquinas pode degradar o desempenho das aplicações paralelas.

Apesar dessas dificuldades, a computação paralela em sistemas distribuídos oferece diversas vantagens:

- Não existe a necessidade de grandes investimentos adicionais, uma vez que o *hardware* já existente pode ser utilizado;

- Alto desempenho, pois cada tarefa pode ser atribuída para a arquitetura mais apropriada;
- Os recursos da máquina virtual podem aumentar gradativamente, o que permite maior facilidade na assimilação de novas tecnologias.

Além das vantagens citadas, um requisito fundamental para a computação distribuída são as redes de comunicação de alta velocidade, que têm apresentado um grande avanço com o desenvolvimento de novas tecnologias visando maior *throughput* e baixa latência. Podem ser citadas, entre outras, as tecnologias *Ethernet*, *FDDI* (*Fiber Distributed Data Interface*) e *ATM* (*Asynchronous Transfer Mode*). Todas essas características têm incentivado a utilização dos chamados ambientes de passagens de mensagens.

Diversos grupos de pesquisa têm desenvolvido ambientes para passagem de mensagens, visando otimizar a utilização de recursos, reduzir os custos e o tempo de desenvolvimento e possibilitar implementações mais eficientes das aplicações, utilizando a computação distribuída. Uma vantagem adicional dos ambientes de passagem de mensagens é permitir a migração de programas paralelos entre diferentes tipos de plataformas. A seção 2.5.2 detalha o paradigma de passagem de mensagens e os ambientes *PVM* e *PVMe* são discutidos nas seções 2.6 e 2.7.

2.5.2 Ambientes de Passagem de Mensagens

Um ambiente de passagem de mensagens consiste basicamente em uma biblioteca (extensão funcional) de comunicação que, juntamente com os comandos oferecidos por uma linguagem seqüencial hospedeira, como C ou Fortran, permite o

desenvolvimento de aplicações paralelas. Quealy define passagem de mensagens como "um método pelo qual os processos se comunicam quando não compartilham memória". O paradigma de passagem de mensagens pode ser visualizado na figura 2.7 [Quealy 1994].

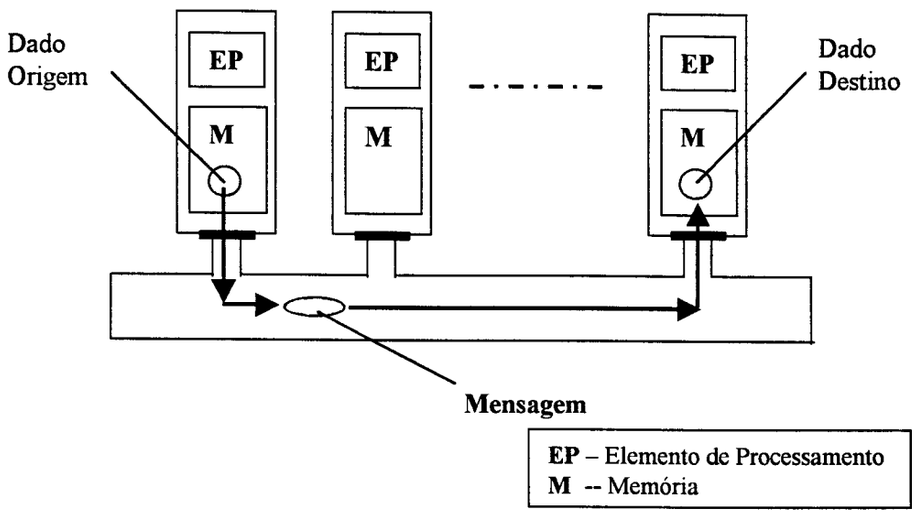

Figura 2.7 - Paradigma de Passagem de Mensagem. Nesse paradigma os processos paralelos, que não compartilham memória, comunicam entre si por intermédio da troca de mensagens.

Inicialmente, os ambientes para passagem de mensagens foram criados pelos próprios fabricantes para os sistemas *MPP's* (*Massively Parallel Processor*). Porém, com a ausência de padrões, aplicações escritas para uma determinada arquitetura não eram portáteis para outras arquiteturas. Os ambientes desenvolvidos, apesar de serem logicamente similares, diferiam na maneira como os serviços principais (sincronização e passagem de dados) eram organizados. Como resultado, obteve-se um grande número de sistemas incompatíveis, dentre os quais podem ser citados *Meiko CS*, *IBM EUI* e *nCUBE PSE* [Geist et al. 1994b].

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

A solução encontrada para o problema da incompatibilidade foi o desenvolvimento das chamadas plataformas de portabilidade [Mcbryan 1994]. Neste caso, um ambiente de programação é definido e implementado em várias plataformas paralelas (arquiteturas paralelas ou sistemas distribuídos), possibilitando que programas possam ser migrados de maneira fácil e direta. Essas plataformas de portabilidade também participaram ativamente do processo de migração da programação paralela para sistemas distribuídos em virtude da adequação natural desses tipos de plataformas para ambientes heterogêneos, formados pela interligação de diferentes arquiteturas em um ambiente único [Souza 1996].

Muitos ambientes de passagem de mensagens foram desenvolvidos nos últimos anos, tendo como característica básica a portabilidade: os ambientes são constituídos por um conjunto de primitivas que independe da plataforma de *hardware* a ser utilizada.

Porém, a falta de um padrão para o desenvolvimento dessas plataformas deu origem a problemas semelhantes aos ocorridos com os ambientes específicos para determinadas arquiteturas. Com o objetivo de padronizar as plataformas de portabilidade, iniciou-se um processo que incluiu representantes de diversas organizações, resultando no padrão *MPI* [Mcbryan 1994].

Alguns ambientes de passagem de mensagens são apresentados na tabela 2.1 [Hwang; Xu 1998].

Tabela 2.1 - Ambientes de Passagem de Mensagens.

AMBIENTE	Origem	Observação
CMMD	*Thinking Machines*	*Software* proprietário
Express	*ParaSoft Corporation*	*Software* proprietário
Fortran-M	*Argonne National Laboratory*	*Software* domínio público
MPI	*MPI Forum*	*Software* domínio público
Nx	*Intel*	*Software* proprietário
P4	*Argonne National Laboratory*	*Software* domínio público
PARMACS	*ANL/GMD*	*Software* domínio público
PVM	*Oak Ridge National Laboratory*	*Software* domínio público
PVMe	*IBM*	*Software* proprietário

CMMD é utilizado em sistemas *Thinking Machines CM-5*. *Express*, da *ParaSoft Corporation*, é constituído por uma coleção de ferramentas e utilitários, incluindo uma interface para passagem de mensagens, para programação paralela em arquiteturas *MIMD* com memória distribuída.

Fortran-M é uma extensão do *Fortran 77*, projetada para fornecer suporte aos paradigmas de passagem de mensagens e de memória compartilhada. Apenas o suporte para passagem de mensagens está disponível atualmente.

MPI foi desenvolvido com objetivo de padronizar os ambientes para passagem de mensagens em sistemas *MIMD* com memória distribuída.

Nx é um sistema desenvolvido para arquiteturas *Intel MPP's* (*Paragon*). *P4* e *Parmacs* (que herdou muitas das características do *P4*) são bibliotecas de macros e subrotinas desenvolvidas pelo *Argonne National Laboratory*, para a programação em uma variedade de máquinas paralelas. *P4* e *Parmacs* suportam os modelos de programação baseados em memória compartilhada (por meio de monitores) e distribuída (pela troca de mensagens).

PVM é um ambiente de passagem de mensagens desenvolvido pelo *Oak Ridge National Laboratory*, sendo considerado um padrão de fato entre os ambientes de passagem de mensagens. *PVMe* é um ambiente de passagem de mensagens desenvolvido pela *IBM*. As seções 2.6 e 2.7 descrevem as características básicas do *PVM* e do *PVMe*, utilizados para o desenvolvimento deste trabalho.

2.6 *PVM*

PVM constitui um conjunto integrado de ferramentas de *software* e bibliotecas que permite que uma rede de computadores heterogêneos possa ser utilizada como um único computador paralelo, constituindo um sistema flexível e de propósito geral. Desta forma, grandes problemas computacionais podem ser resolvidos utilizando os computadores que compõem a rede [Geist et al. 1994a, Geist el al. 1994b, Sunderan et al. 1994]. O termo máquina virtual é utilizado para referenciar um computador lógico com memória distribuída e *host* é utilizado para designar os computadores que constituem a máquina virtual.

A estrutura do *PVM* está baseada nos seguintes princípios [Geist et al. 1994b]:

- ***Host pool***: as aplicações são executadas em um conjunto de máquinas que pode ser configurado pelo próprio usuário. Podem fazer parte desse *pool*, máquinas com um único processador e multiprocessadores (com memória compartilhada ou distribuída). A configuração do *host pool* também pode ser alterada dinamicamente pelo usuário;
- **Acesso transparente ao *hardware***: as aplicações podem tratar o *hardware* como sendo uma coleção de elementos de processamento em uma máquina

paralela virtual, ou podem explorar as características individuais de cada arquitetura;

- **Computação baseada em tarefas**: a unidade de paralelismo no *PVM* é a tarefa (análoga a um processo *UNIX*) que alterna execução seqüencial entre computação e comunicação, o que permite a execução concorrente de tarefas em um *host*;
- **Modelo para passagem de mensagens**: conjuntos de tarefas, cada uma delas executando uma parte do trabalho de uma aplicação, comunicam-se e sincronizam-se por intermédio da troca de mensagens. O tamanho dessas mensagens é limitado apenas pela quantidade de memória disponível no *host*;
- **Suporte a heterogeneidade**: *PVM* suporta heterogeneidade em termos de arquitetura de computadores, redes de comunicação e aplicações. Ou seja, *PVM* permite que as tarefas utilizem a arquitetura que melhor se adapta à solução do problema. Quando dois computadores utilizam representações diferentes para dados, *PVM* manipula todas as conversões que possam ser necessárias.

O sistema *PVM* é composto por duas partes principais: o *daemon*, responsável pelo controle da máquina virtual, e a biblioteca, que engloba todas as primitivas oferecidas pelo ambiente:

- ***Daemon***: denominado *PVMD3* ou simplesmente *PVMD*, reside em todos os *hosts* e possibilita a formação da máquina virtual. Um exemplo de programa *daemon* é o *sendmail*, que manipula todas as mensagens (*mail*) em um sistema *UNIX*. Quando um usuário quer executar uma aplicação

PVM, ele deve primeiro iniciar o mesmo executando o *PVMD* especificando uma lista de *hosts* que vão constituir a máquina virtual. As aplicações no *PVM*, que podem ser iniciadas em qualquer *host* que compõe o sistema, fazem requisições e recebem serviços do *daemon*. O *daemon* não realiza processamento, mas atua como um roteador e controlador de mensagens, funcionando como um ponto de contato entre cada *host*. Diversos usuários podem configurar suas máquinas virtuais, sem interferir na máquina virtual de outro usuário [Geist et al. 1994a];

- **Biblioteca *PVM***: é uma biblioteca (*libpvm*) que contém as primitivas necessárias para a cooperação entre as tarefas, como troca de mensagens, criação de processos, sincronização de tarefas e alterações na máquina virtual. Os programas de aplicação devem ser ligados com essa biblioteca para utilizar o *PVM*.

A implementação do *PVM* está disponível atualmente para uma série de plataformas, como por exemplo *IBM PC*, *IBM/RS6000*, *Sun SparcStation*, *Silicon Graphics*, *Intel Paragon* e *Cray* [Geist et al. 1994b].

2.6.1 O Modelo Computacional no *PVM*

O modelo computacional do *PVM* considera que uma aplicação é constituída por diversas tarefas, cada uma responsável por executar uma parte da aplicação. Aplicações podem ser paralelizadas por meio dos paradigmas *MPMD* ou *SPMD*.

No paradigma *MPMD* cada tarefa tem uma função distinta (como entrada, saída ou processamento) e diferentes máquinas executam diferentes tarefas. Por

exemplo, um supercomputador vetorial pode executar parte de um problema adequado para vetorização, enquanto um multiprocessador pode resolver outra parte do problema que possa ser paralelizado, e uma estação gráfica pode ser utilizada para visualizar os dados gerados em tempo real.

No paradigma *SPMD* todas as tarefas são idênticas, porém trabalham com dados distintos, que são fracionados e distribuídos entre as mesmas. Os programas são executados sobre cada conjunto de dados e informações são trocadas entre tarefas até que o problema seja resolvido [Geist et al. 1994a, Sunderan et al. 1994].

O *PVM* suporta esses dois modelos de paralelismo e também uma combinação deles. A figura 2.8(a) apresenta uma visão do modelo computacional do *PVM* e a figura 2.8(b) apresenta uma visão arquitetural do mesmo [Sunderan et al. 1994].

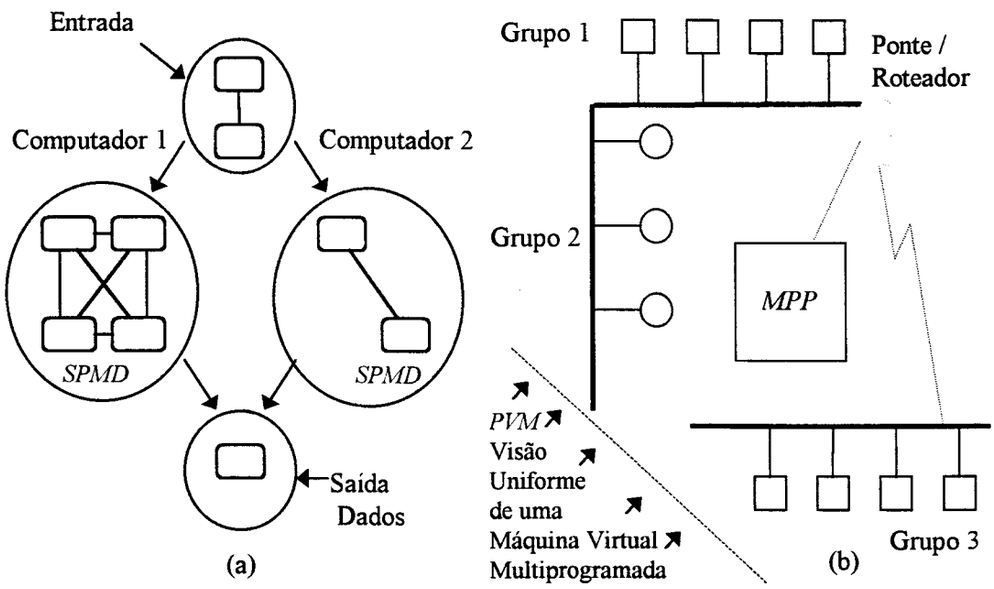

Figura 2.8 - Modelo Computacional e Visão Arquitetural do *PVM*. O modelo computacional do *PVM* possibilita que as aplicações sejam constituídas por diversas tarefas (2.8(a)). Essas aplicações podem ser paralelizadas por meio dos paradigmas *MPMD* ou *SPMD*. Diversos tipos de arquiteturas podem constituir a máquina paralela virtual (2.8(b)).

2.6.2 O Modelo de Comunicação no *PVM*

O modelo de comunicação do *PVM* assume que qualquer tarefa pode enviar mensagens para outra tarefa, e não há limites para o tamanho ou número de mensagens enviadas (apenas a quantidade de memória disponível em cada *host*). Três passos são necessários para se enviar uma mensagem no *PVM*. Primeiro é necessário criar um *buffer* (dinamicamente) para armazenar a mensagem. Em seguida essa mensagem deve ser empacotada e finalmente enviada a outro processo. O receptor pode aceitar qualquer mensagem, qualquer mensagem de um determinado *host*, qualquer mensagem com um determinado identificador, ou somente as mensagens com um determinado identificador e de determinado *host*.

A comunicação entre as tarefas no *PVM* pode ser realizada nos seguintes modos [Geist et al. 1994a, Sunderan et al. 1994]:

- *send* bloqueante assíncrono;
- *receive* bloqueante assíncrono;
- *receive* não bloqueante.

Um *send* bloqueante retorna logo que o *buffer* de transmissão fica livre para reutilização, independente do estado do receptor. Um *receive* não bloqueante retorna imediatamente com a mensagem ou a informação de erro, após verificar o *buffer* do receptor. Um *receive* bloqueante só retorna quando a mensagem está disponível no *buffer* de recepção.

Além da comunicação ponto a ponto, o modelo suporta também *multcast* para um conjunto de tarefas e *broadcast* para um grupo de tarefas definido pelo usuário. O modelo garante que a ordem das mensagens é preservada entre qualquer par de entidades que estão se comunicando.

A troca de mensagens entre dois nós pertencentes a um multiprocessador é efetuada com as rotinas locais. Já as mensagens destinadas a um *host* externo são roteadas por meio dos *daemons*. Em sistemas com memória compartilhada a troca de dados pode ser implementada com um *pool* de *buffers* compartilhados e primitivas do tipo *lock*.

Sockets são utilizados para intercomunicação entre processos e cada *host* na máquina virtual pode se conectar diretamente a qualquer outro *host* por intermédio dos protocolos *TCP* (*Transmission Control Protocol*) e *UDP* (*User Datagrama Protocol*). O protocolo *TCP* é utilizado para a comunicação entre as tarefas e o *PVM*

daemon, devido à confiabilidade que esse protocolo possui. Já a comunicação entre os *daemons* é implementada utilizando o protocolo *UDP*.

O padrão *XDR* (*EXternal Data Representation*) é empregado quando existe comunicação em ambientes heterogêneos. Sua utilização é muitas vezes necessária e gera tempos adicionais na transmissão da mensagem, devido às sobrecargas impostas para a codificação da mensagem pelo processo emissor e sua decodificação pelo processo receptor.

2.6.3 Outras Características do *PVM*

O sistema *PVM* disponibiliza aos usuários uma série de facilidades para o gerenciamento de processos, tolerância a falhas e criação de grupos dinâmicos, descritas a seguir.

- **Controle de processos:** *PVM* fornece rotinas que possibilitam adicionar ou retirar *hosts* da máquina virtual, iniciar e encerrar tarefas, enviar sinais para outras tarefas, mapear as tarefas para os *hosts*, depurar programas, além de obter informações sobre a configuração da máquina virtual e tarefas ativas no sistema;
- **Tolerância a falhas:** *PVM* detecta automaticamente a falha de um *host*, retirando-o da máquina virtual. O controle e recuperação das tarefas "mortas" pela falha de um *host* são de responsabilidade do usuário;
- **Criação de grupos de processos dinâmicos:** os processos no *PVM* podem pertencer a muitos grupos (criados em tempo de execução). Esses podem ser alterados dinamicamente em qualquer ponto de execução e podem ser

sobrepostos. Rotinas permitem que as tarefas possam se juntar e deixar qualquer grupo, e também permitem obter informações sobre outros grupos, além de possibilitar o *broadcast* de mensagens e a sincronização entre os membros de um grupo;

- **Sinalização:** existem dois métodos para a sinalização de tarefas (notificar a ocorrência de um evento) no *PVM*:
  - utilizar o *signal* do sistema *UNIX*;
  - enviar uma mensagem com um identificador para outras tarefas.

Alguns exemplos de eventos que podem ocorrer no *PVM* são a falha, retirada ou inclusão de um *host* na máquina virtual, ou o término de uma tarefa.

2.7 *IBM PVMe*

O *PVMe* é um ambiente de passagem de mensagens desenvolvido pela *IBM* e otimizado para a utilização das características do *IBM SP2 (IBM RISC System/6000 Scalable POWERparallel System)*[1].

Esse ambiente de passagem de mensagens foi desenvolvido de forma a manter a compatibilidade com o *PVM* de domínio público, porém sua estrutura interna foi adaptada para a execução em plataformas homogêneas (*IBM SP2* e *cluster* de estações de trabalho *IBM RS/6000*), e também para explorar as características do

[1] *IBM SP2* constitui uma arquitetura paralela com memória distribuída, composta de nós (elementos de processamento com memória e discos próprios) interligados por um *switch* de alto desempenho (*High Performance Switch*).

switch[2] de alto desempenho do *IBM SP2*, o que permite que muitas operações sejam simplificadas.

O *PVMe* é constituído por dois componentes: o *daemon* (*pvmd3e*) e a biblioteca de comunicação. O *daemon* é responsável pelo controle, sincronização e gerenciamento de *hosts* e grupos de tarefas. Ao contrário do *PVM*, no *PVMe* o *daemon* é executado em um único elemento de processamento, o que possibilita simplificar e agilizar o gerenciamento de tarefas. A biblioteca disponibiliza todas as primitivas necessárias para a criação, comunicação e sincronização entre tarefas [*IBM Corporation* 1996].

As aplicações em *PVMe* podem ser desenvolvidas utilizando os paradigmas de programação *MPMD* e *SPMD*. A comunicação entre os processos paralelos ocorre sempre sem a participação do *daemon*. O *PVMe* utiliza o *software* de comunicação *MPCI* (*Message Passing Client Interface*), que executa sobre o *switch* de alto desempenho. O *MPCI* utiliza dois protocolos de comunicação: *IP* (*Internet Protocol*) e *US* (*User Space*) [*IBM Corporation* 1996].

O protocolo *US* foi desenvolvido exclusivamente para o *switch* de alto desempenho, permitindo o uso exclusivo do *switch* por uma determinada aplicação e possibilitando maior desempenho dessa aplicação. O protocolo *IP* suporta múltiplos processos por elemento de processamento, porém não oferece desempenho como o protocolo *US*.

[2] O *switch* de alto desempenho é uma rede de comunicação bidirecional, com interconexão multi-estágio, que conecta todos os elementos de processamento do *IBM SP2*, permitindo que possam comunicar entre si simultaneamente (*IBM Corporation* 1996).

2.8 Considerações Finais

Os últimos anos têm apresentado um avanço tecnológico significativo, o qual tem possibilitado tanto o desenvolvimento de arquiteturas com maior poder computacional quanto o desenvolvimento de novas tecnologias de redes de comunicação, resultando no surgimento de máquinas para processamento de alto desempenho. Como conseqüência um grande aumento na aceitação e utilização da computação paralela, voltado para a computação científica de alto desempenho e também para as aplicações de propósito geral, vem sendo observado.

Essa aceitação tem sido favorecida principalmente pelo desenvolvimento dos ambientes com processamento maciçamente paralelo (*MPP's*) e da computação distribuída. A maioria dos sistemas *MPP's* utiliza memória distribuída (*MIMD*) e dados são trocados entre tarefas que cooperam entre si. Um ponto comum entre computação distribuída e *MPP's* é a noção de troca de mensagens. Para a maioria desses sistemas, ambientes para passagem de mensagens têm sido desenvolvidos. Com a interligação de diversos computadores por meio de uma rede de comunicação de alta velocidade e confiabilidade, é possível obter grande poder computacional que pode ser utilizado para diversas aplicações paralelas. Os fatores citados levaram então ao desenvolvimento dos chamados ambientes de portabilidade para passagem de mensagens.

As plataformas de portabilidade têm-se mostrado eficientes para a implementação de programas paralelos utilizando troca de mensagens, e o principal atrativo dessas plataformas é a possibilidade de migrar programas diretamente entre ambientes heterogêneos, incluindo arquiteturas paralelas e sistemas distribuídos.

Dentre as plataformas de portabilidade destaca-se o *PVM* por sua generalidade e aplicabilidade em uma grande gama de aplicações, sendo inclusive considerado por muitos autores como um padrão "de fato".

É importante ressaltar que *PVM* e outros sistemas similares não são simplesmente uma estrutura de *software* para computação paralela/concorrente: *PVM* constitui uma metodologia integrada para processamento paralelo, distribuído e concorrente, e mais importante, é uma interface para o desenvolvimento de aplicações portáteis.

CAPÍTULO 3

PROTOCOLOS DE SINCRONIZAÇÃO EM SIMULAÇÃO DISTRIBUÍDA

Este capítulo apresenta os conceitos básicos sobre simulação discreta, simulação distribuída e os principais protocolos de sincronização, utilizados no desenvolvimento de programas de simulação distribuída. O protocolo CMB (Chandy-Misra-Bryant) [Misra 1986], objetivo de estudo deste trabalho, é apresentado em detalhes.

3.1 Introdução

Uma das áreas da computação que tem se tornado essencial com a grande evolução das arquiteturas de computadores é a avaliação de desempenho. Diversas ferramentas têm sido utilizadas para esse fim, como protótipos, *benchmarks* e coleta de dados. A escolha de qual ferramenta utilizar depende fundamentalmente do objetivo do estudo em andamento e do tipo de avaliação desejada.

Para o caso em que se deseja avaliar um projeto e o sistema ainda não está disponível, as técnicas mencionadas no parágrafo anterior tornam-se inviáveis. Neste caso, a utilização de modelos de simulação pode auxiliar no entendimento do projeto, na sua avaliação e na avaliação do desempenho que será obtido [Swain 1995].

Uma outra situação em que a simulação pode ser utilizada com grandes vantagens é na avaliação de modificações em arquiteturas existentes. Essa técnica é capaz de responder de forma tanto quantitativa quanto qualitativa às perguntas do tipo "e se?" feitas pelos projetistas de sistemas. Como um exemplo dessa situação pode-se citar a avaliação da influência de um processador duas vezes mais rápido no desempenho de uma máquina. Desta forma, conclui-se que a simulação é uma ferramenta flexível e pode ser utilizada em diferentes situações. Outras vantagens que podem ser citadas sobre simulação são baixo custo, popularidade e possibilidade de analisar alterações de projeto com facilidade [MacDougall 1987, Soares 1992].

Apesar das vantagens na utilização da simulação seqüencial, algumas características restringem a sua aplicação. A execução de uma simulação baseia-se em variáveis aleatórias. Desta forma, para que os resultados de uma simulação sejam válidos, deve-se executá-la por um longo período, garantindo assim que essa aleatoriedade não está influindo nos resultados finais. Quando se deseja simular um sistema em um nível de detalhes considerável, o tempo de simulação pode tornar-se inviável.

Uma solução para diminuir esse tempo é dividir a simulação em diversos processos e executá-los em paralelo, adotando assim uma simulação distribuída, simulação paralela ou simulação concorrente [Bagrodia et al. 1998].

Com o advento da simulação distribuída, torna-se necessária a utilização de protocolos para garantir a coerência entre os processos lógicos e a execução correta da simulação. A seção 3.2 apresenta conceitos básicos da simulação seqüencial e as demais seções apresentam as particularidades da simulação distribuída, os protocolos utilizados para a sincronização entre os processos, uma comparação entre esses protocolos e as principais linguagens e extensões utilizadas.

3.2 Conceitos Básicos Sobre Simulação Discreta

Simulação é a arte e a ciência de criar a representação de um sistema com o objetivo de realizar experimentos e avaliar seu desempenho [Gogf; Mott 1995]. A simulação é uma experiência ou ensaio realizado com o auxílio de modelos. A partir dessa definição, entende-se por simulação como sendo uma representação das características do comportamento de um sistema físico real ou teórico.

A simulação é utilizada com freqüência em uma grande variedade de sistemas, seja nas ciências físicas, sociais, biológicas e econômicas. Quatro situações principais justificam a utilização da simulação: como uma ferramenta para estudar e para definir um problema, para a detecção de elementos críticos, para a síntese e avaliação de soluções propostas e para o planejamento de desenvolvimentos futuros [Gogf; Mott 1995, Maryansky 1980].

O estado de um sistema é definido por um conjunto de variáveis que contém todas as informações necessárias para descrever o sistema em qualquer tempo. A base para a classificação dos modelos são as alterações que ocorrem nas variáveis de estado do sistema e, em um grande número de simulações, o tempo é a principal variável independente. As demais variáveis incluídas na simulação são funções do

tempo e portanto, variáveis dependentes. Os modelos de um sistema são classificados em discretos e contínuos, de acordo com o comportamento das variáveis independentes [Soares 1992].

- **Modelos discretos**: representam sistemas nos quais as mudanças em seu estado ocorrem em pontos específicos e descontínuos do tempo simulado, alterando o estado do sistema instantaneamente;
- **Modelos contínuos**: representam sistemas nos quais as mudanças em seu estado ocorrem continuamente ao longo do tempo simulado.

A seguir são descritas as principais características de um sistema discreto, uma vez que este trabalho analisa o comportamento da simulação distribuída em sistemas computacionais, representados por esse tipo de modelo.

3.2.1 Composição de Sistemas Discretos

O primeiro passo de uma simulação é o desenvolvimento de um modelo que descreva o comportamento dinâmico do sistema a ser simulado. Um modelo constitui uma simplificação do sistema, podendo ser representado de acordo com vários níveis de detalhe (cada nível apresenta um grau de abstração) [MacDougall 1987].

A composição de um sistema pode ser definida em termos de entidades denominadas atividades, processos e eventos [MacDougall 1987, Tanir; Suleyman 1994]:

- **Atividade:** representa a menor unidade de trabalho no sistema, porém depende da visão particular que se tenha do mesmo. Uma atividade pode

ser a execução de um passo de uma instrução ou a execução de uma tarefa em uma simulação macroscópica de um sistema computacional;

- **Processo:** um conjunto de atividades logicamente relacionadas constitui um processo. Uma operação de escrita em disco, constituída de posicionamento, atraso rotacional e atividades de transferência de dados, pode ser vista como um processo;
- **Evento:** representa a mudança de estado de alguma entidade do sistema. O término de uma atividade é um evento cuja ocorrência pode iniciar outras atividades subseqüentes. Tais eventos controlam a seqüência de atividades dentro de um processo.

Um evento ocorre em um ponto isolado no tempo, no qual decisões devem ser tomadas para iniciar ou encerrar uma atividade. Um processo é então uma seqüência ordenada de eventos, podendo englobar várias atividades [Soares 1992].

Um modelo de simulação discreta pode ser visto como uma coleção de processos interativos, cada processo composto por diversas atividades e com as interações controladas e coordenadas pela ocorrência de eventos [MacDougall 1987].

A relação entre esses conceitos pode ser melhor compreendida pela figura 3.1 [Soares 1992].

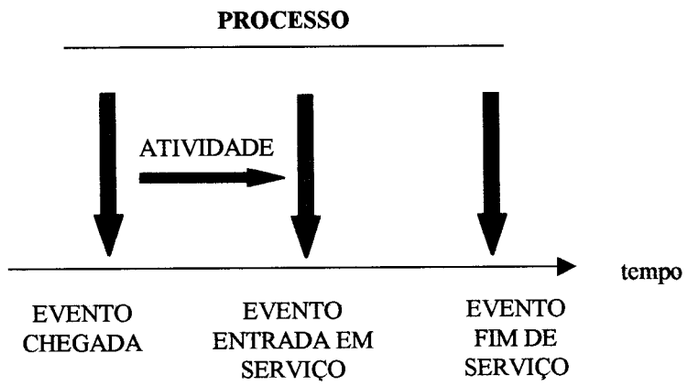

Figura 3.1 - Relação entre Evento, Processo e Atividade. O processo é composto por diversas atividades, e todas as interações são controladas e coordenadas pela ocorrência de eventos.

Em simulação discreta, atividades, processos e eventos são construções utilizadas para descrever o seu comportamento. Desta forma, a elaboração de um modelo para simulação discreta pode ser realizada de três modos, de acordo com a maneira como o sistema é visto:

- **Simulação orientada a eventos**: a simulação do sistema é produzida pela execução da lógica associada a cada evento, em uma seqüência ordenada no tempo;
- **Simulação orientada a atividades**: conforme o tempo simulado avança, as condições para o início ou fim de uma atividade são verificadas. A cada avanço do tempo, todo o conjunto de atividades é examinado para garantir que cada atividade seja levada em consideração;
- **Simulação orientada a processos**: nesta abordagem o processo é utilizado para definir a estrutura e a atividade de um componente do

sistema sendo modelado, existindo então uma grande correspondência entre o sistema real e o modelo.

De acordo com MacDougall [MacDougall 1987] linguagens com orientação a processo são adequadas para a implementação de modelos complexos, uma vez que os processos retratam diretamente as operações do sistema. Já linguagens orientadas a evento e atividades são adequadas para modelos com menor complexidade, uma vez que esse tipo de representação tende a impor uma visão global do sistema a ser modelado.

Na seção 3.3 são discutidos aspectos relacionados com a simulação orientada a eventos. Essa abordagem foi escolhida para o desenvolvimento do trabalho visando, no futuro, a integração da extensão para simulação distribuída ao ASiA (Ambiente de Simulação Automático) [Santana et al. 1996a, Santana et al. 1996b], em desenvolvimento pelo Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP (Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo).

3.3 Simulação Distribuída

No paradigma de simulação discreta três estruturas de dados são utilizadas para a implementação das primitivas básicas no desenvolvimento do programa de simulação orientado a eventos [Misra 1986, Nicol 1996]:

- as variáveis que descrevem o estado do sistema;
- uma lista de eventos, denominada lista de eventos futuros, que contém todos os eventos pendentes para execução;

- um relógio global que controla o progresso da simulação.

Cada evento a ser executado possui uma marca de tempo e determina alguma mudança no estado do sistema que está sendo simulado. A marca de tempo determina quando essa mudança deve ocorrer. A lista de eventos futuros é ordenada ascendentemente pelo tempo de ocorrência dos eventos.

O programa de simulação repetidamente remove o evento com a menor marca de tempo do início da lista para executá-lo. A ativação desses eventos é controlada pelo relógio global, que determina o tempo de simulação. Quando o programa completa todo o processamento em um determinado instante do tempo, o evento do início da lista (com o menor tempo associado) é retirado, o relógio é então avançado para o tempo de ocorrência do mesmo, e o programa de simulação inicia a execução do evento [Fujimoto 1990, MacDougall 1987]. Esse mecanismo garante que os eventos no sistema físico sejam simulados em ordem cronológica no tempo de simulação.

Como discutido na seção 3.1, um problema da simulação está relacionado com o grande esforço computacional necessário para a obtenção de resultados realistas. Às vezes o tempo de processamento chega a ser inviável, conforme a complexidade e a magnitude do problema a ser solucionado. Além do tempo de simulação excessivo, muitos modelos de simulação necessitam de grande quantidade de memória que pode não estar disponível em um único elemento de processamento [Bagrodia et al. 1998].

Para esses casos a utilização da simulação distribuída vem, gradativamente, substituindo a simulação seqüencial, permitindo o uso de máquinas paralelas com vários processadores e alta capacidade de comunicação [Fujimoto 1990, Misra 1986, Overeinder et al. 1991].

No entanto, a natureza do mecanismo da lista de eventos futuros imposta pela simulação seqüencial, na qual a cada ciclo da simulação apenas um item é removido da lista de eventos, não permite a execução concorrente do algoritmo em máquinas com mais de um elemento de processamento (já que a lista de eventos não pode ser efetivamente particionada). A existência de uma lista de eventos ordenados cronologicamente impõe uma simulação seqüencial na qual apenas um evento é simulado por vez.

Assim, para possibilitar a utilização de diversos elementos de processamento na execução de uma simulação é necessário adaptar a simulação seqüencial para essa nova situação, uma vez que as estruturas de dados utilizadas (centralizadas) não podem ser compartilhadas na simulação distribuída [MacDougall 1987].

3.4 O Problema Fundamental em Simulação Distribuída

Na simulação distribuída o sistema a ser simulado é decomposto em um conjunto de n sub-sistemas ou n processos lógicos PL_0, PL_1, ..., PL_n, cada um representando um processo do sistema real.

Cada processo lógico possui um relógio que indica o progresso da simulação. Quando um evento é processado, o relógio do processo é automaticamente avançado para a marca de tempo desse evento. Se um PL_i pode enviar uma mensagem para PL_j durante a simulação, então existe um canal de comunicação entre esses dois processos. Todas as interações entre os processos físicos são modeladas por mensagens (eventos com marca de tempo) trocadas entre os processos lógicos correspondentes, obedecendo a ordem seqüencial imposta pelo tempo no sistema real. As mensagens são trocadas entre os processos por intermédio dos canais de

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

comunicação. Não existem objetos compartilhados na simulação distribuída, e cada processo tem suas próprias estruturas de dados, como por exemplo a lista de eventos futuros [Nicol 1996].

Na simulação, se dois eventos são escalonados para execução em um mesmo processo, o evento com a menor marca de tempo deve ser executado primeiro. Se a execução de um evento em um processo resulta no escalonamento de outro evento em um processo diferente, então o evento local deve ser executado antes do evento remoto [Lin; Lazowska 1991].

Todo o processamento dos eventos deve então ser análogo ao comportamento do sistema real, e os relacionamentos de causa e efeito dos eventos nunca devem ser violados. O relacionamento de causa e efeito determina que, se em um sistema real sendo simulado, o evento E_1 ocorre antes do evento E_2, na simulação o mesmo deve ocorrer, ou seja, E_1 deve ser tratado antes de E_2.

Uma simulação constituída de processos lógicos que interagem por via da troca de mensagens (com marca de tempo) obedece ao relacionamento de causa e efeito se e somente se cada processo lógico executa os eventos em ordem crescente da marca de tempo [Overeinder et al. 1991].

Considere como exemplo um sistema de redes de filas, onde E_1 poderia representar a chegada de um cliente na fila de um recurso no tempo t_1 e E_2 a chegada de outro cliente ao mesmo recurso no tempo t_2. Suponha que a fila de atendimento seja do tipo *FIFO* (*First-In, First-Out*) e que t_1 seja menor que t_2. Se o evento E_2 for simulado antes do evento E_1 (o futuro afetando o passado) tem-se a ocorrência de um erro, violando-se a relação de causa e efeito [Fujimoto 1990].

Considere outro exemplo: o processo PL_1 executa o evento E_1 com marca de tempo 10 e o processo PL_2 executa o evento E_2 com marca de tempo 20 (figura 3.2(a)). Se E_1 escalona um novo evento E_3 para execução em PL_2 com marca de tempo menor do que 20, E_3 pode afetar o evento E_2, o que implica na necessidade de execução seqüencial dos três eventos (figura 3.2(b)) [Overeinder et al. 1991].

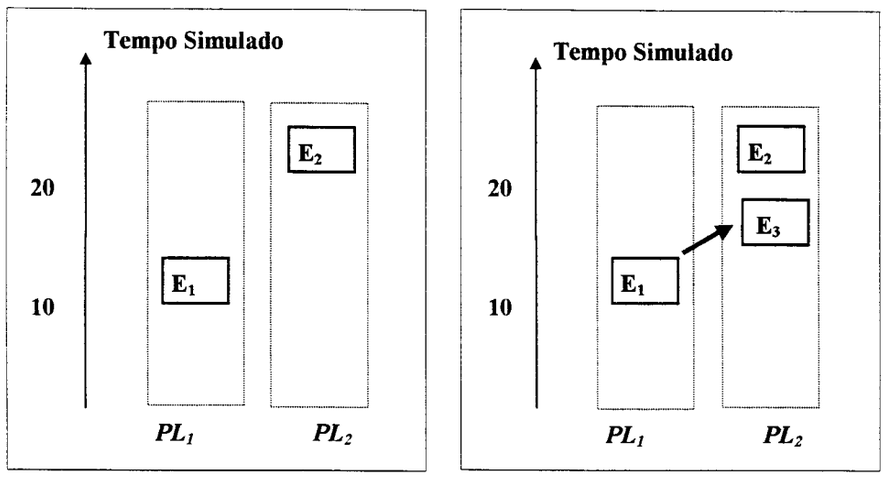

Figura 3.2 - Ocorrência de Erros de Causa e Efeito. A relação de causa e efeito determina que se em um sistema real sendo simulado o evento E_1 ocorre antes do evento E_2, na simulação o mesmo deve ocorrer: E_1 deve ser tratado antes de E_2 (a). Em (b) o escalonamento do evento E_3 pode implicar na ocorrência desse tipo de erro.

Para solucionar o problema do sincronismo entre os processos na simulação distribuída, diversos protocolos têm sido desenvolvidos. A seção 3.5 apresenta a classificação dos protocolos para sincronização, bem como uma revisão dos principais protocolos utilizados.

3.5 Protocolos para Sincronização em Simulação Distribuída

A simulação distribuída pode ser classificada em dois grupos principais: os protocolos síncronos e os protocolos assíncronos (figura 3.3) [Bagrodia et al. 1998].

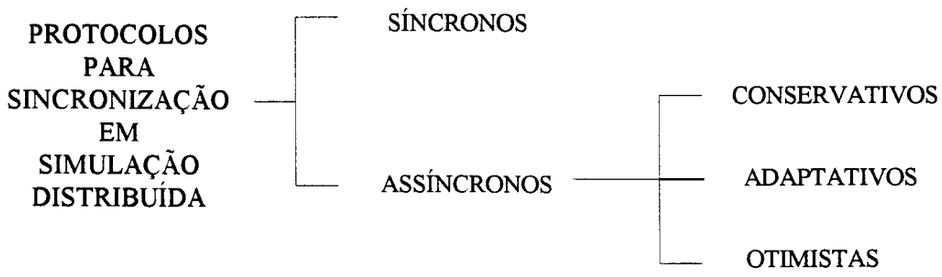

Figura 3.3 - Classificação dos Protocolos em Simulação Distribuída. Os protocolos são classificados em síncronos e assíncronos, de acordo com a existência ou não de mecanismos para o controle da simulação. Os protocolos assíncronos dividem-se em conservativos (que evitam a ocorrência de erros de causa e efeito), otimistas (que permitem a ocorrência de erros de causa e efeito e utilizam mecanismos de recuperação) e adaptativos (podem mudar dinamicamente seu comportamento).

Alguns autores citam ainda outra classe de protocolos, denominados protocolos mistos. Nesse tipo de protocolo podem existir submodelos executando em modo conservativo ou modo otimista [Bagrodia et al. 1998].

Nos protocolos síncronos são incluídos mecanismos globais para o controle da simulação e todos os processos compartilham o mesmo relógio global, processando todos os eventos que ocorrem em um determinado tempo da simulação. Devido aos mecanismos de sincronização utilizados, esses algoritmos são mais adequados para arquiteturas com memória compartilhada [Fujimoto 1990]. Exemplos desses protocolos, como Janela de Tempo Conservativo e Evento Condicional, podem ser

encontrados em [Ayani; Rajaei 1992, Ayani; Rajaei 1994, Chandy; Sherman 1989, Lubachevsky 1989, Nicol, Fujimoto 1993].

Nos protocolos assíncronos cada processo sincroniza somente com os processos com os quais se comunica, e cada processo possui seu relógio local. Os eventos são processados pela ordem crescente do tempo de ocorrência, e os processos podem ter valores de relógios diferentes em um determinado ponto da simulação [Alonso et al. 1995a, Ferscha; Tripathi 1994].

Os protocolos assíncronos estão classificados em dois grupos: protocolos conservativos e protocolos otimistas [Fujimoto 1990, Jefferson 1985, Hamilton Jr. Et al. 1997, Misra 1986]. Os protocolos conservativos evitam a possibilidade da ocorrência de erros de causa e efeito, ou seja, determinam quando é seguro processar um evento. Já os protocolos otimistas utilizam uma estratégia de detecção e recuperação na qual erros de causa e efeito são detectados, e um mecanismo de *rollback* é utilizado para a recuperação [Fujimoto 1990, Nicol 1996].

Entre os protocolos conservativos e otimistas estão os protocolos adaptativos, ou seja, aqueles que têm a capacidade de modificar seu comportamento dinamicamente, em respostas às mudanças no estado da simulação [Ferscha 1995, Srinivasan; Reynolds Jr. 1995, Tay et al. 1997]. Ayani [Ayani 1993] descreve essa nova classe de protocolos, que incorpora as características dos protocolos conservativos e otimistas.

Historicamente, os primeiros protocolos para simulação distribuída foram baseados em abordagens conservativas. Esses protocolos foram desenvolvidos independentemente por Chandy e Misra [Chandy; Misra 1979] e Bryant [Bryant 1977].

Nas simulações conservativas, um processo lógico só trata um evento se puder garantir que não chegará um outro com marca de tempo menor do que a do evento a ser tratado [Srinivasan; Reynolds Jr. 1995].

O problema básico desses protocolos consiste então em determinar quando é seguro processar um evento, evitando rigorosamente que ocorra em algum momento um erro de causa e efeito. Os processos que não contêm eventos seguros ficam bloqueados, o que pode levar a situações de *deadlock* se não forem tomadas as precauções necessárias [Fujimoto 1990].

O tratamento do *deadlock* é um dos principais enfoques dos protocolos conservativos, os quais diferem basicamente quanto à forma desse tratamento. Alguns protocolos previnem o *deadlock*, enquanto outros permitem que a simulação entre em *deadlock* para depois detectá-lo e utilizam algum mecanismo para se recuperar.

A seguir são descritas as características de alguns protocolos conservativos, uma vez que este trabalho analisa o comportamento do protocolo conservativo *CMB*. A técnica de *lookahead*, definida como sendo a capacidade de um processo prever o futuro, é mostrada na seção 3.5.1. Na seção 3.5.2 são apresentados os protocolos conservativos clássicos: Prevenção do *Deadlock* ou *CMB* [Chandy; Misra 1979, Bryant 1977], Transmissão de Mensagens Nulas sob Demanda [Chandy; Misra 1981, Misra 1986] e Detecção e Recuperação do *Deadlock* [Chandy; Misra 1981, Fujimoto 1990]. A seção 3.5.3 faz uma revisão sobre protocolos otimistas.

3.5.1 *Lookahead*

Misra [Misra 1986] define *lookahead* como sendo o intervalo de tempo em que um processo pode inferir o futuro com absoluta certeza. Já Fujimoto [Fujimoto 1990] define *lookahead* como a capacidade do processo em prever o que acontecerá, ou mais importante, o que não acontecerá, no tempo de simulação futuro. Se um processo no tempo de simulação T pode prever com segurança todos os eventos que ele poderá gerar até o tempo de simulação $T+L$, pode-se dizer que o processo tem *lookahead* L.

Lookahead não é um protocolo de simulação distribuída, mas uma técnica fundamental da qual os protocolos conservativos fazem uso. Diferentes modelos de simulação apresentam capacidade de *lookahead* distintos [Turner 1998].

Incrementos mínimos diferentes de zero nas marcas de tempo são a forma mais evidente de *lookahead*. Um incremento mínimo na marca de tempo com o valor L representa um *lookahead* de L, pois o processo pode garantir que nenhuma mensagem nova de evento será criada com marca de tempo menor que $T+L$. Porém, essa técnica só pode ser aplicada em situações onde se utiliza estratégia *FIFO* nas filas e onde não ocorram preempções. [Fujimoto 1990].

Diversos trabalhos apresentam algoritmos e estudos sobre o impacto do *lookahed* no desempenho de simulações distribuídas que utilizam protocolos conservativos [Cota; Sargent 1989, Fujimoto; Nicol 1992, Meyer; Bagrodia 1999, Naroska; Schwiegelshohn 1999, Nicol 1988, Nicol 1996, Peterson; Chamberlain 1993, Preiss; Loucks 1989, Preiss; Loucks 1990].

3.5.2 Protocolos Conservativos

Nesta seção são descritas as características do protocolo conservativo *CMB* e variações desse protocolo.

3.5.2.1 O Protocolo *CMB* (Prevenção do *Deadlock* utilizando Mensagens Nulas)

Este protocolo exige que se especifique estaticamente os canais que indicam a comunicação entre os processos. Para determinar quando é seguro processar um evento, é necessário que a seqüência das marcas de tempo das mensagens enviadas por um canal seja crescente.

As mensagens que chegam em cada canal de entrada são armazenadas em ordem *FIFO*, que também é a ordem das marcas de tempo, pela restrição acima. Cada canal possui um relógio que armazena o valor da marca de tempo da mensagem que está no início da fila (o relógio terá valor zero se nenhuma mensagem tiver sido recebida pelo canal, ou o valor da marca de tempo da última mensagem recebida, se a fila estiver vazia). O relógio do processo, *LVT (Local Virtual Time)*, é sempre o menor valor entre os relógios dos seus canais de entrada [Ferscha; Tripathi 1994, Misra 1986].

A figura 3.4 [Fujimoto 1990] apresenta como exemplo o Processo Proc, cujos canais de entrada *L1*, *L2* e *L3* apresentam, respectivamente, relógio igual a 11, 2 e 4. O menor valor do tempo de recebimento de mensagens é igual a 2 (canal *L2*). Portanto o *LVT* do processo Proc é igual a 2 e o processo pode seguramente prosseguir sua execução até esse tempo.

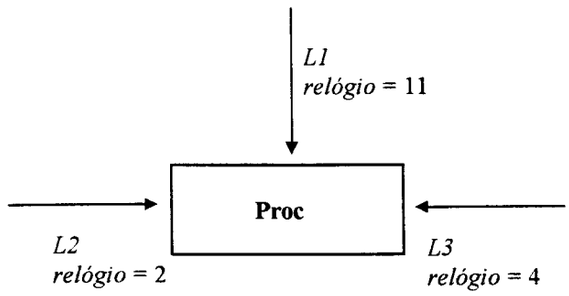

Figura 3.4 - *LVT* de um Processo em uma Simulação Distribuída Conservativa. O *LVT* (relógio do processo) assume sempre o menor valor entre os relógios dos seus canais de entrada, representados pelas marcas de tempo das mensagens que chegam pelos respectivos canais. Na figura o menor valor do tempo de recebimento de mensagens é igual a 2 (canal *L2*). Portanto, o *LVT* do processo Proc é igual a 2.

O valor do relógio da simulação (*T*) é o menor valor de relógio de todos os processos lógicos. Pode-se declarar que em um determinado ponto da simulação, o sistema físico tenha sido simulado até o tempo indicado por *T*, mesmo que algum processo lógico tenha simulado seu processo físico correspondente até um tempo maior que *T* [Morselli 1995].

Um problema crítico neste protocolo é demonstrado na figura 3.5 [Fujimoto 1990]. Cada processo fica indefinidamente esperando uma mensagem no canal de entrada que contém o menor valor de relógio, uma vez que a fila correspondente está vazia. Todos os três processos estão bloqueados, embora existam mensagens em outras filas de entrada de cada processo, aguardando para serem processadas.

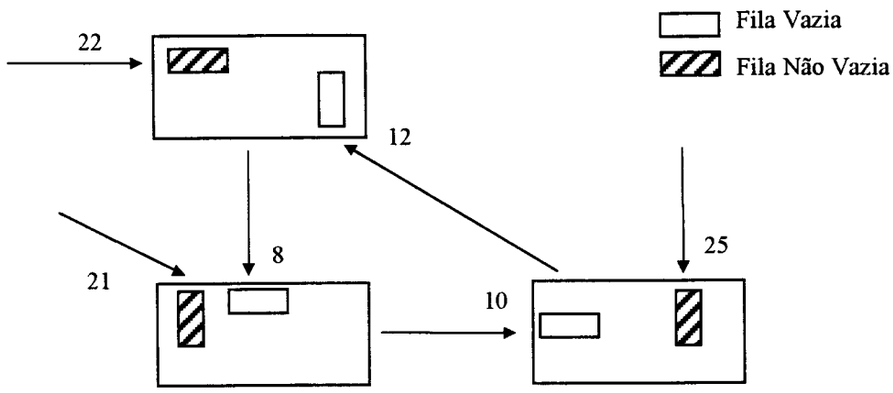

Figura 3.5 - Situação de *Deadlock* em uma Simulação Distribuída. De acordo com o protocolo, cada processo pode avançar seu relógio até o menor dos tempos dos canais de entrada e, conseqüentemente, a simulação pode progredir até esse ponto. No exemplo todos os processos estão indefinidamente esperando para receber uma mensagem no canal de entrada que contém o menor valor de relógio, apesar de outras filas conterem mensagens. Neste caso, todos os processos estão bloqueados e a simulação entra em *deadlock*.

Para evitar que a simulação entre em *deadlock*, foi desenvolvido um protocolo que utiliza mensagens nulas [Misra 1986]. Esse protocolo é descrito a seguir.

Prevenção do *Deadlock* utilizando Mensagens Nulas

Para que a simulação não entre em *deadlock*, são utilizadas mensagens nulas, enviadas para os processos com os quais o processo em execução mantém comunicação. Essas mensagens são utilizadas apenas com o propósito de sincronização, e não correspondem a atividades no sistema físico. Uma mensagem nula com marca de tempo t, enviada por um PL_i para um PL_j indica que PL_i não enviará mensagens a PL_j entre o valor corrente do relógio e o tempo t.

Como citado anteriormente, LVT_i indica o valor do relógio de um processo PL_i. Quando esse processo PL_i recebe uma mensagem, ele próprio atualiza seu LVT_i. Se o LVT_i é alterado, PL_i avança a simulação do processo físico correspondente até o valor de LVT_i.

Nesse instante, PL_i prediz para cada canal de saída uma seqüência de mensagens que o processo físico possa ter enviado, gerando uma seqüência de mensagens para transmitir para PL_j, para cada j no qual exista um canal de saída.

Suponha que PL_i possa futuramente predizer que após a transmissão dessa seqüência de mensagens o processo físico correspondente (i) não irá enviar mensagens ao processo físico PL_j até o tempo t_j. Nesse caso, PL_i envia a mensagem (t_j, nula) a PL_j depois de ter enviado a seqüência correta de mensagens. Uma vez que PL_i conhece o estado do seu processo físico correspondente até o tempo LVT_i, pode predizer todas as mensagens que estão sendo enviadas, e também detectar a ausência de mensagens no mínimo até o tempo LVT_i.

As mensagens nulas são tratadas como qualquer mensagem quando recebidas por um processo: a recepção de uma mensagem nula faz com que o processo atualize seu LVT e (possivelmente) envie novas mensagens [Misra 1986].

3.5.2.2 Transmissão de Mensagens Nulas Sob Demanda

Uma desvantagem do protocolo *CMB* é a utilização das mensagens nulas. Uma variação desse protocolo consiste em enviar mensagens nulas sob demanda [Chandy; Misra 1981] e não após o processamento de cada evento. A freqüência dessa demanda pode ser dada por um *timeout* ou também quando o menor relógio de

todos os canais for o de uma fila vazia, indicando que o processo está bloqueado. Quando isso ocorre, a próxima mensagem é requisitada (pode ser uma mensagem nula ou não) do processo que envia para esse canal. O processo continua sua execução quando a resposta a esse pedido for recebida.

Supondo que um processo PL_1 requisita a um processo PL_2 que aumente o valor do relógio do canal que conecta os dois processos. Esse avanço é possível se o valor do relógio desse canal for menor que o valor atual do *LVT* de PL_2 (mínimo entre os relógios dos canais de entrada de PL_2).

Neste caso, PL_2 envia uma mensagem avançando o *relógio* do canal (PL_2 PL_1). A figura 3.6 ilustra o exemplo. O processo PL_2 pode enviar uma mensagem nula ao processo PL_1 incrementando o relógio do canal (PL_2 PL_1) para 12. Com isso, o processo PL_1 pode computar o novo valor do seu *LVT* (igual a 9) e prosseguir sua execução.

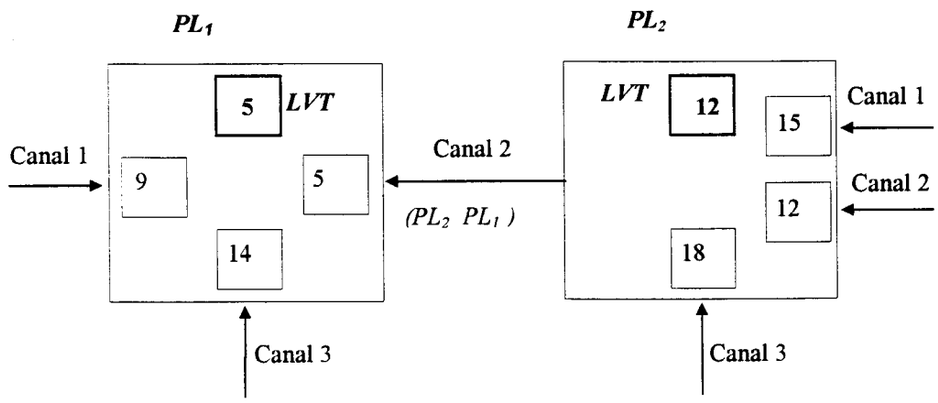

Figura 3.6 - Transmissão de Mensagens Nulas Sob Demanda. Exemplo de solicitação de avanço do relógio. O processo PL_2 pode enviar uma mensagem ao processo PL_1 incrementando o relógio do canal $(PL_2\ PL_1)$ para 12. Assim, o processo PL_1 pode computar o novo valor do seu *LVT* (igual a 9) e prosseguir sua execução.

Porém, se o valor do relógio do canal $(PL_2\ PL_1)$ for igual ao valor do *LVT* de PL_2, nenhum avanço será possível (figura 3.7). PL_2 tem que avançar seu próprio *LVT* primeiro, fazendo o mesmo tipo de pedido a todos os processos que comunicam com PL_2, cujos relógios do canal sejam iguais ao *LVT* de PL_2. É o caso do canal de entrada 2 do processo PL_2. Neste caso, o pedido será efetivamente propagado por PL_2.

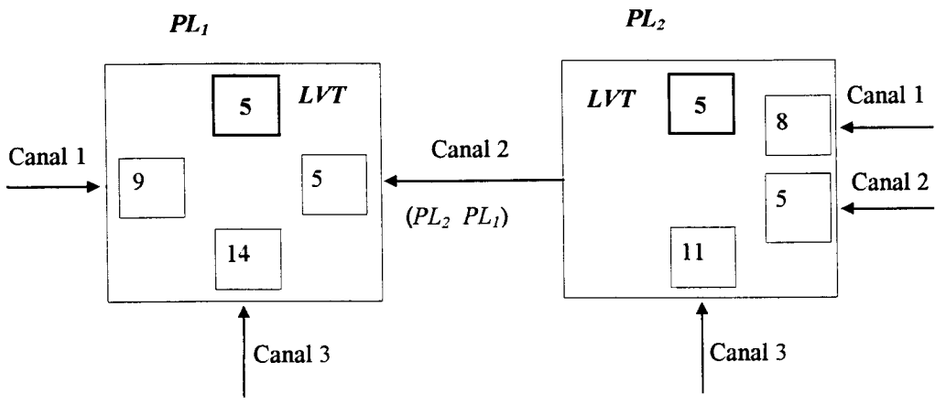

Figura 3.7 - Transmissão de Mensagens Nulas Sob Demanda. Nesta situação o avanço do relógio não é possível, pois o valor do relógio do canal (PL_2 PL_1) é igual ao valor do *LVT* de $PL_2(5)$.

Somente quando o valor do *LVT* de PL_2 for maior que o valor do relógio do canal (PL_2 PL_1), PL_2 pode enviar uma mensagem para PL_1, incrementando o valor do relógio do canal. As propagações dos pedidos de avanço do relógio podem formar um ciclo, gerando uma situação de *deadlock*.

Segundo Misra [Misra 1986] a resolução do *deadlock* é difícil e pode ser efetuada determinando-se o mínimo dos próximos tempos de eventos. Para cada processo no conjunto de processos em *deadlock*, o tempo em que uma mensagem será enviada (contando que nenhuma mensagem posterior seja recebida até então) é determinado e o valor do relógio do processo com o menor relógio é avançado para esse tempo.

3.5.2.3 Detecção e Recuperação do *Deadlock*

Em situações nas quais a ocorrência de *deadlock* é rara têm-se uma sobrecarga de envio de mensagens nulas. Chandy e Misra desenvolveram um protocolo

alternativo que elimina o uso de mensagens nulas [Chandy; Misra 1981]. Nesse protocolo a simulação é executada até que entre em *deadlock*. Um mecanismo detecta quando isso ocorre, e outro mecanismo é utilizado para a recuperação do *deadlock*, para que a simulação prossiga sua execução.

Esse protocolo utiliza um marcador que circula continuamente pela rede de processos, devendo percorrer todos os canais da rede, em algum tempo durante um ciclo. Esse marcador é apenas um tipo especial de mensagem. Quando um processo recebe o marcador, deve enviá-lo para sua rota designada depois de um certo tempo em que o processo se tornar ocioso (sem mensagens para enviar) [Misra 1986, Fujimoto 1990].

Cada processo possui um *flag* de 1 *bit* que indica se o mesmo recebeu ou enviou uma mensagem desde a última vez que recebeu o marcador. Um processo é branco se não recebeu nem enviou mensagens desde a última passagem do marcador; caso contrário, o processo é preto. Inicialmente todos os processos são pretos. A situação de *deadlock* é detectada quando os N últimos processos visitados pelo marcador estejam brancos (N é o número de canais na rede). Contudo, o sucesso desse protocolo é garantido apenas quando, entre dois processos quaisquer, a ordem de recebimento das mensagens (inclusive o marcador) for a mesma ordem com que foram enviados.

O marcador, além de armazenar o número de processos brancos que visitou desde o último processo preto encontrado, contém também informações sobre os próximos tempos de eventos dos processos brancos que visitou. Quando o marcador detecta o *deadlock* (ele possui a indicação do próximo tempo de evento e do processo em que ocorre o *deadlock*), pode reiniciar o processo [Misra 1986].

3.5.2.4 Considerações Sobre os Protocolos Conservativos

No protocolo *CMB* o número de mensagens nulas enviadas pode gerar uma sobrecarga de comunicação, o que pode prejudicar o desempenho da simulação.

Com a transmissão de mensagens nulas sob demanda tem-se uma redução do tráfego de mensagens nulas, já que essas são enviadas apenas quando necessário. Porém, ocorre um atraso maior para receber as mensagens nulas, já que, no mínimo, duas transmissões de mensagens são necessárias: uma mensagem requisitando uma mensagem de avanço de relógio e a própria resposta a esse pedido. Além disso, esse protocolo pode levar a situações de *deadlock*, exigindo um mecanismo complexo para ser solucionado.

Com o protocolo de detecção e recuperação do *deadlock* tem-se uma sobrecarga de comunicação, gerada pela transmissão do marcador, além da necessidade de um mecanismo para a resolução do *deadlock*.

3.5.3 Protocolos Otimistas

Os métodos otimistas detectam e recuperam erros de causa e efeito, não os evitam. Em contraste com os protocolos conservativos, as estratégias otimistas não necessitam determinar quando é seguro executar um evento. Ao invés disso determinam quando ocorreu um erro, e utilizam um procedimento para retornar a simulação para um estado consistente. *Time Warp* é o protocolo otimista mais conhecido, sendo apresentado na seção 3.5.3.1.

3.5.3.1 *Time Warp*

No protocolo *Time Warp* todos os eventos são considerados seguros e são executados. Quando um erro de causa e efeito é detectado (quando uma mensagem de evento é recebida contendo marca de tempo menor do que o relógio do processo) um mecanismo de recuperação, denominado *rollback*, deve ser executado.

Esse protocolo é constituído por dois mecanismos básicos, o mecanismo de controle local (responsável por garantir que os eventos sejam executados e as mensagens recebidas em ordem correta) e o mecanismo de controle global (destinado ao gerenciamento de memória e cálculo do *GVT* (*Global Virtual Time*)) [Jefferson 1985].

Um processo lógico na abordagem *Time Warp* é similar a um processo na abordagem *CMB*, contendo basicamente as seguintes estruturas:

- ***LVT***: determina o relógio do processo;
- **Lista de entrada de mensagens**: armazena as mensagens recebidas pelo processo. Mesmo depois de executada, uma mensagem continua a fazer parte da lista. Essas mensagens são armazenadas em ordem crescente da marca de tempo e são denominadas mensagens positivas [Schof 1998];
- **Lista de saída de mensagens**: contém uma cópia de todas as mensagens enviadas pelo processo. Nessa lista todas as mensagens têm marca de tempo menor ou igual a *LVT*, e são denominadas mensagens negativas (antimensagens). Essa lista é utilizada em caso de *rollback* [Schof 1998];

- **Lista de estados**: armazena cópia dos estados do processo. Para viabilizar a execução do *rollback*, o estado do processo é salvo periodicamente [Soliman 1998].

A execução da simulação consiste na recepção de mensagens e execução dos eventos por parte dos processos lógicos (mecanismo local), em ordem crescente da marca de tempo, sendo interrompida com a chegada de uma mensagem com marca de tempo menor que a marca de tempo de uma mensagem já processada. Quando essa situação ocorre, um erro de causa e efeito é detectado, e um *rollback* deve ser efetuado para retornar a um estado anterior à violação [Fujimoto 1990].

A recuperação de um erro exige que o estado do processo seja periodicamente salvo na lista de estados. Quando uma mensagem é enviada, é inserida na lista de entrada do processo receptor e uma cópia da mesma (antimensagem) é inserida na lista de saída de mensagens do processo transmissor.

A operação de "enviar de volta" uma mensagem anteriormente enviada é executada por meio do envio da antimensagem, que cancela a mensagem original quando essa atinge o destino (mensagens positivas e negativas na mesma lista são eliminadas). Se um processo recebe uma antimensagem que corresponde a uma mensagem que já executou, deve retornar para desfazer a execução fora de ordem. A repetição desse procedimento de forma recursiva permite que todos os efeitos das computações erradas sejam cancelados [Fujimoto 1990].

O mecanismo de controle global é responsável pelo controle do *GVT*. Após o cálculo do *GVT*, as informações registradas com marca de tempo anteriores ao seu valor podem ser eliminadas. Ou seja, todas as informações que não são utilizadas a

tempos (denominadas fósseis) podem ser removidas das listas onde estão armazenadas.

3.5.3.2 Estratégias de Cancelamento

Quando um *rollback* ocorre pode existir a necessidade de cancelar algumas mensagens que foram enviadas para outros processos. Para efetuar essa operação podem ser utilizados diferentes tipos de estratégias de cancelamento:

- **Cancelamento Agressivo**: quando um processo efetua *rollback* para o tempo *T*, antimensagens são enviadas imediatamente, para todas as mensagens anteriormente enviadas com marca de tempo maior do que *T* [Fujimoto 1990];
- **Cancelamento Preguiçoso**: nessa estratégia os processos não enviam as antimensagens imediatamente após o *rollback*. Ao invés disso, esperam para verificar se a nova execução da computação gera as mesmas mensagens. Neste caso, não existe a necessidade de efetuar o cancelamento [Fujimoto 1990];
- **Cancelamento Dinâmico**: utiliza uma técnica na qual cada processo decide qual estratégia de cancelamento, agressivo ou preguiçoso, vai utilizar [Wilsey 1997].

3.5.3.3 Técnicas para Gerenciamento de Memória

O protocolo *Time Warp* requer uma quantidade maior de memória para executar uma simulação, em relação à simulação seqüencial correspondente. Na simulação seqüencial é necessário manipular apenas os eventos pendentes (aqueles

que estão na lista de eventos futuros). No *Time Warp* o mecanismo de *rollback* exige que os processos salvem seus estados periodicamente, o que introduz sobrecarga no gerenciamento de memória.

Por esse motivo, diversos mecanismos para gerenciamento de memória têm sido propostos, como por exemplo: Redução dos Estados Salvos [Das; Fujimoto 1993], Otimismo Limitado [Madiseti et al. 1992, Prakash; Subramarian 1991, Reiher; Jefferson. 1989], Coleta de Fósseis e protocolo de *Gafni* [Das; Fujimoto 1993], Retorno de mensagens e *Cancellback* [Akyildiz et al. 1992].

3.6 Comparação entre Protocolos Conservativos e Otimistas

A questão da superioridade dos protocolos conservativos ou otimistas tem sido freqüentemente levantada, e regras gerais para utilização de um ou outro protocolo em simulação distribuída são difíceis de serem formuladas. Até mesmo implementações completas de um mesmo protocolo podem dificultar a comparação quando diferentes estratégias de implementação ou diferentes plataformas são utilizadas. Otimizações específicas de alguns protocolos (por exemplo, o cancelamento preguiçoso no *Time Warp*) impedem uma melhor comparação entre os mesmos. Decidir qual protocolo utilizar não é uma tarefa fácil, pois diferentes fatores devem ser considerados, tais como a plataforma de *hardware* disponível e as particularidades do modelo a ser simulado.

Nos protocolos conservativos um problema crítico encontrado é a necessidade do programador conhecer os detalhes do mecanismo de sincronização para obter um bom desempenho. Além disso, o paralelismo da simulação pode não ser totalmente explorado. Se existe a possibilidade, mesmo rara, de um evento E_a afetar o evento

E_b, esses eventos são executados seqüencialmente. Porém, dependendo de uma série de fatores, pode-se obter bom desempenho.

Um problema sério com o protocolo *Time Warp* é a necessidade de salvar periodicamente o estado de cada processo lógico. A sobrecarga gerada por essa operação pode degradar o desempenho do protocolo. Além disso, os algoritmos otimistas tendem a utilizar mais memória do que os conservativos [Fujimoto 1990]. Essa classe de protocolos pode oferecer grande potencial como protocolos de simulação de propósito geral, segundo Overeinder [Overeinder et al. 1991].

A tabela 3.1 apresenta uma comparação das principais características dos protocolos *CMB* e *Time Warp* [Ferscha; Tripathi 1994].

Tabela 3.1 - Comparação das Características dos Protocolos Conservativos e Otimistas.

Estratégia	Protocolos Conservativos (*CMB*)	Protocolos Otimistas (*Time Warp*)
Princípio Operacional	• evita a violação do princípio de causa e efeito • somente eventos "seguros" são processados	• violações ao princípio de causa e efeito ocorrem, mas um mecanismo de recuperação é acionado quando detectado, imediatamente ou no futuro • todos os eventos são processados
Sincronização	• mecanismo de sincronização: bloquear processos • propenso a situações de *deadlock* • protocolos baseados em mensagens nulas: sobrecarga de comunicação • protocolos de detecção e recuperação: gerenciador de *deadlock* centralizado	• mecanismo de sincronização: *rollback* • mecanismos de aniquilação: sobrecarga de comunicação • cascatas de *rollbacks*: sobrecarrega utilização da memória e degrada desempenho
Paralelismo	• paralelismo do modelo não pode ser completamente explorado • comportamento excessivamente pessimista se a ocorrência de erros de causa e efeito for rara	• paralelismo do modelo é totalmente explorado • se a ocorrência de erros de causa e efeito for provável mas não freqüente, desempenho do protocolo aumenta
Lookahead	• necessário para tornar o protocolo operacional, essencial para seu desempenho	• não trabalha com *lookahead*
Global Virtual Time	• não requer cálculo de *GVT*	• *GVT* difícil de ser calculado, cálculo centralizado->gargalo; cálculo distribuído->sobrecarga de comunicação
Estados	• consumo de memória moderado • aceita modelos com espaços de estado grandes	• melhor desempenho quando o espaço de estados, necessário a ser armazenado, é pequeno
Memória	• consumo de memória moderado	• consumo de memória dinâmico • sobrecarga - salvar estados • cálculo freqüente do *GVT* • gerenciamento de memória complexo
Mensagens e Comunicação	• obrigatória a chegada das mensagens e processamento dos eventos em ordem cronológica da marca de tempo	• mensagens podem chegar fora de ordem cronológica - devem ser executadas pela ordem da marca de tempo
Implementação	• "simples" para implementar • estruturas de dados e controles simples	• complexo para implementar • estruturas de dados simples, porém manipulação complexa

3.7 O Estado da Arte em Simulação Distribuída

O desenvolvimento das arquiteturas maciçamente paralelas e a expansão do paradigma de passagem de mensagens disponibilizou ambientes propícios para o desenvolvimento de aplicações paralelas, especificamente a simulação distribuída (a disponibilidade de bibliotecas paralelas em máquinas com memória compartilhada também incentivou esse desenvolvimento) [Lim et al. 1998b].

O grande interesse pela simulação distribuída, entretanto, não coincide com a larga aplicação desse paradigma em aplicações do mundo real. A maior parte das pesquisas objetiva o desenvolvimento de novos algoritmos e a obtenção de melhor desempenho para as aplicações específicas. Duas razões principais explicam esse fenômeno. Primeiro, a utilização de máquinas com processamento paralelo, que são utilizadas principalmente pela comunidade científica, enquanto a maioria dos usuários tem acesso somente a máquinas monoprocessadoras. Segundo, o esforço de migrar uma simulação seqüencial para o paradigma distribuído requer uma quantidade substancial de esforço de programação, além de exigir experiência no desenvolvimento de aplicações paralelas [Low et al. 1999].

Para encorajar a adoção do paradigma de simulação distribuída, o processo de migração de um programa seqüencial para distribuído deve ser simplificado. O usuário deve concentrar seus esforços no processo de modelagem, e não se preocupar com as características do protocolo de sincronização utilizado. Com essa finalidade, diversas linguagens e bibliotecas (extensões funcionais) para simulação distribuída têm sido desenvolvidas nos últimos anos, como por exemplo, *Maise*, *YADDES* e *WARPED*. São apresentadas a seguir algumas das linguagens e extensões mais utilizadas no desenvolvimento da simulação distribuída.

3.7.1 Linguagens para Simulação Distribuída

A principal característica de uma linguagem para simulação distribuída é a transparência do modelo de programação disponibilizado ao usuário, ou seja, a linguagem deve esconder as características do protocolo de sincronização utilizado. Uma linguagem de simulação, normalmente, fornece um conjunto completo de primitivas que permite ao usuário projetar seus modelos de simulação, além do mesmo poder contar com o ambiente fornecido pela linguagem. Algumas linguagens para o desenvolvimento da simulação distribuída são apresentadas na tabela 3.2 [Low et al. 1999].

Tabela 3.2 - Linguagens para Simulação Distribuída.

LINGUAGEM	Organização	Protocolo
APOSTLE	*DRA Malvern*	Otimista
Maise/Moose/Parsec	*UCLA*	Conservativo/Otimista
ModSim	*CACI Products Company*	Otimista
YADDES	*University of Waterloo*	Conservativo/Otimista

APOSTLE é uma linguagem orientada a objetos, baseada no protocolo otimista *Time Warp*, em desenvolvimento desde 1994. A linguagem *Maise* implementa uma série de protocolos conservativos e otimistas [Preiss; Loucks 1989]. *Moose* é uma versão orientada a objetos de *Maise*. *Parsec* utiliza *Maise* na implementação de um ambiente para desenvolvimento de programas de simulação paralela. *ModSim* é uma linguagem orientada a objetos baseada em Modula 2. *YADDES* oferece suporte para diferentes protocolos de sincronização.

A tabela 3.3 apresenta uma comparação entre as principais características das linguagens citadas anteriormente [Bagrodia et al. 1998].

Tabela 3.3 - Comparação entre as Diferentes Linguagens para Simulação Distribuída.

	APOSTLE	*Maise/Parsec*	*Moose*	*ModSim*	*YADDES*
Protoloco	Otimista	Otimista Conservativo	Otimista Conservativo	Otimista	Otimista Conservativo
Orientação	Processo Evento	Processo Evento	Processo Evento	Processo Evento	Processo
Paradigma de Programação	Orientada Objeto	Estruturada	Orientada Objeto	Orientada Objeto	Especificação
Mapeamento	Manual	Manual	Manual	Automático	Manual
Visualização	Não	*MVPE*	*MVPE*	Não	Não
Depuração	Não	Sim	Sim	Não	Não
Estatísticas	Não	Sim	Sim	Sim	Sim

3.7.2 Bibliotecas para Simulação Distribuída

Uma biblioteca (ou extensão funcional) para simulação distribuída fornece apenas o conjunto de primitivas que devem ser utilizadas em conjunto com a linguagem de programação hospedeira (como C ou C++) e que possibilitam o desenvolvimento do programa de simulação. Algumas das bibliotecas utilizadas atualmente são apresentadas na tabela 3.4 [Low et al. 1999].

Tabela 3.4 - Bibliotecas para Simulação Distribuída.

Biblioteca	**Organização**	**Protocolo**
Cpsim	*Boyan Tech Inc.*	Conservativo
GTW	*Georgia Tech*	Otimista
Parasol	*Purdue University*	Otimista
PSK	*Royal Institute of Technology*	Otimista
Simkit	*University of Calgary*	Otimista
SpaDES	*National University of Singapore*	Otimista
SPEEDES	*Jet Propulsion Laboratory*	Otimista
TWOS	*Jet Propulsion Laboratory*	Otimista
WARPED	*University of Cincinnati*	Otimista

Dentre as bibliotecas apresentadas, *CPSim* é a única biblioteca disponível comercialmente. Essa biblioteca foi desenvolvida na linguagem C e comercializada a partir de 1995. *GTW* e *TWOS* também são desenvolvidas na linguagem C, e apresentam otimizações específicas para o protocolo *Time Warp*. *SPEEDS*

(*Synchronous Parallel Environment for Emulation and Discrete Event Simulation*) utiliza um protocolo síncrono para a sincronização entre os processos. *Parasol*, *PSK*, *Simkit*, *SPaDES* e *WARPED* são bibliotecas orientadas a objeto desenvolvidas em C++. Algumas dessas bibliotecas foram projetadas especificamente para o estudo de protocolos de sincronização, como *PSK* e *Parasol*.

A tabela 3.5 [Low et al. 1999] apresenta as principais características das bibliotecas para simulação distribuída.

Tabela 3.5 - Principais Características das Bibliotecas para Simulação Distribuída.

(a)

Característica	*CPSim*	*GTW*	*ParaSol*	*PSK*	*Simkit*
Protocolo	Conservativo	Otimista	Otimista	Otimista	Otimista
Orientação	Evento	Evento	Processo	Evento	Evento
Paradigma de Programação	Gabarito	Gabarito	Orientação objeto	Orientação objeto	Orientação objeto
Mapeamento	Manual Automático	Manual	Manual	Automático	Manual
Visualização	Não	Sim	Não	Sim	Não
Depuração	Sim	Sim	Sim	Sim	Sim
Estatísticas	Sim	Sim	Não	Não	Não

(b)

Característica	*SPaDES*	*SPEEDES*	*TWOS*	*WARPED*
Protocolo	Otimista	Otimista	Otimista	Otimista
Orientação	Processo	Evento	Evento	Evento
Paradigma de Programação	Orientação objeto	Orientação objeto	Gabarito	Orientação objeto
Mapeamento	Manual	Manual	Manual	Manual
Visualização	Não	Não	Sim	Não
Depuração	Não	Sim	Sim	Sim
Estatísticas	Não	Não	Sim	Não

Com o advento das máquinas maciçamente paralelas, o número de pesquisas e aplicações para os protocolos conservativos têm aumentado [Alonso et al. 1995a,

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Alonso et al. 1995b, Alonso et al. 1998, Preiss; Wan 1999, Xiao et al. 1999]. Diversos protocolos conservativos têm sido desenvolvidos, abrangendo alterações nos protocolos básicos ou trazendo novas propostas para a implementação da simulação distribuída, dentre eles:

- Alterações de protocolos existentes: [Dickens; Nicol 1993, Fujimoto; Nicol 1992, Galluscio et al. 1995, Hoeger; Jones 1996, Kacprzak 1999, Porras et al. 1997, Porras et al. 1998, Preiss; Loucks 1990, Preiss et al. 1991, Vries 1990];
- Novos protocolos ou protocolos específicos para uma aplicação: [Cai et al. 1997, Lim et al. 1998a, Nicol 1996, Nicol; Fujimoto 1993, Peytchev; Bargiela 1995, Pham; Fdida 1996, Pham et al. 1997, Pham et al. 1998, Poplawski; Nicol 1998, Sanches et al. 1996].

Em relação aos protocolos otimistas, diversas alterações (visando a otimização) e protocolos baseados no *Time Warp* original são descritos na literatura [Das 1996, Deelman; Szymanski 1997, D'Souza et al. 1997, Ferscha 1995, Kalantery 1997, Mascarenhas et al. 1997, Radharamanam; Abu-Ghazaleh 1998, Tay et al. 1997, Varghese et al. 1994, Wilsey 1997].

3.8 Considerações Finais

A simulação é uma técnica amplamente utilizada, seja como uma ferramenta para análise de desempenho, seja para o aperfeiçoamento e desenvolvimento de novos sistemas. Contudo, o desenvolvimento de um modelo de simulação consiste em uma tarefa complexa e até mesmo em uma arte, exigindo um bom conhecimento do

sistema real para que o modelo represente realisticamente as características essenciais do sistema em estudo. Além disso, o grande esforço computacional necessário para a execução de uma simulação realista pode tornar esse processo inviável, o que incentiva então a utilização da simulação distribuída.

A simulação distribuída oferece diversas vantagens sobre a simulação seqüencial: além do *speedup*, exige pouco controle global exercido por qualquer máquina e a simulação de um sistema pode ser adaptada com a estrutura do *hardware* disponível (com poucos processadores, diversos processos físicos podem ser simulados em um único processador).

O principal problema relacionado com a simulação distribuída consiste em lidar com a ocorrência de erros de causa e efeito, ou seja, impedir que o futuro possa afetar o passado. Os protocolos conservativos e otimistas solucionam esse problema a partir de diferentes abordagens.

Nos protocolos otimistas a complexidade dos mecanismos necessários para viabilizar o *rollback* e a quantidade de memória exigida podem comprometer o desempenho da simulação. Nos protocolos conservativos o programador deve estar familiarizado com os detalhes do protocolo para obter um bom desempenho.

Neste capítulo foram apresentados os principais conceitos relacionados aos protocolos conservativos e otimistas, além de uma análise comparativa entre essas abordagens e as principais linguagens e bibliotecas de simulação distribuída disponíveis atualmente.

Como o objetivo deste trabalho é analisar o comportamento do protocolo *CMB*, o estudo realizado possibilitou o desenvolvimento da extensão funcional *ParSMPL*, que utiliza o protocolo *CMB* para a sincronização entre os processos

paralelos. O capítulo quatro descreve essa ferramenta, apresentando as características, a estrutura dos processos lógicos e todos os aspectos relacionados com a implementação da extensão funcional.

CAPÍTULO 4

DESCRIÇÃO DA EXTENSÃO FUNCIONAL *ParSMPL*

Este capítulo descreve a extensão funcional ParSMPL, desenvolvida para viabilizar a avaliação de desempenho de sistemas computacionais distribuídos utilizando o protocolo de sincronização CMB em diferentes plataformas computacionais. As estruturas dos processos lógicos, o modelo de comunicação e as principais características da extensão são apresentados.

4.1 Introdução

Conforme discutido no capítulo 3, diversas variações para os protocolos conservativos e otimistas vêm sendo propostas na literatura. Um problema encontrado por um usuário no momento da escolha por uma das variações desses protocolos é a forma de avaliá-los. Normalmente, esses protocolos e suas variações

são propostos e avaliados para uma situação particular, o que não permite uma comparação global das possibilidades disponíveis.

Desta forma, responder a questão sobre qual protocolo ou qual variação deve ser utilizado em um problema específico não é uma tarefa trivial. Não existem na área de simulação distribuída *benchmarks* consagrados que auxiliem nessa tarefa. Diversas formas para efetuar uma avaliação de um protocolo considerando diferentes tipos de aplicação podem ser utilizadas [Spolon et al. 1999].

As seções 4.2 e 4.3 apresentam diversas abordagens para a avaliação de protocolos de sincronização para simulação distribuída. Diferentes ferramentas e métricas são apresentadas e avaliadas, e a opção pelo uso da prototipação (com o desenvolvimento de uma extensão funcional para simulação distribuída – *ParSMPL*) e coleta de dados é justificada.

A seção 4.4 apresenta uma visão geral da extensão funcional *ParSMPL* e as seções seguintes detalham as características, as estruturas de dados, o modelo de programação, as formas de validação e teste da extensão desenvolvida e um exemplo de utilização. São apresentadas também as características básicas do *SMPL*, uma extensão funcional para simulação orientada a eventos, e do *SMPLX*, uma extensão funcional para simulação orientada a eventos voltada para plataformas *UNIX*, desenvolvida neste trabalho.

4.2 Avaliação de Desempenho da Simulação Distribuída

Como na avaliação de desempenho de sistemas computacionais, a avaliação da simulação distribuída pode ser realizada por meio de técnicas de aferição e de técnicas

de modelagem. O estudo pode ser realizado por meio de experimentação no sistema (técnicas de aferição) ou pela solução de um modelo que represente esse sistema (técnicas de modelagem). A tabela 4.1 apresenta essas técnicas.

Tabela 4.1 – Mecanismos para Avaliação de Desempenho.

Técnicas de Aferição	Técnicas de Modelagem
Coleta de Dados	Solução Analítica
Benchmarks	Solução por Simulação
Protótipos	

A coleta de dados pode ser feita com o uso de monitores de *hardware* ou *software*. Um *benchmark* é um programa escrito em linguagem de alto nível, sendo considerado como representativo para uma dada classe de programas de aplicação. Um protótipo consiste em uma versão simplificada do sistema real [Spolon et al. 1999].

As técnicas de aferição por meio de *benchmarks* e coleta de dados são adequadas para sistemas já existentes. Deve-se ter o cuidado para que o *benchmark* escolhido seja representativo e que a coleta de dados não interfira na execução do sistema real. Já o protótipo pode ser utilizado quando o sistema em estudo ainda está em fase de projeto ou quando a experimentação com o sistema real não é viável. Porém essa abordagem pode apresentar um custo muito alto.

Para a construção de um modelo do sistema em estudo, pode-se utilizar por exemplo redes de fila, redes de Petri ou *statecharts*, e a solução desse modelo pode ser feita por meio de ferramentas analíticas ou de simulação. Os resultados analíticos são mais precisos, porém, com o aumento da complexidade, tem-se o aumento da dificuldade de resolução do modelo. A simulação é uma ferramenta mais versátil e

permite refletir alterações no modelo de forma mais simplificada do que no caso analítico [Spolon et al. 1999].

Resumindo, as técnicas de modelagem são mais adequadas ao estudo de um sistema que ainda não existe, ou em situações onde a aferição (por meio de *benchmarks* ou coleta de dados) pode interferir com os resultados obtidos. As técnicas de aferição apresentam resultados mais precisos quando comparadas às técnicas de modelagem e a escolha de qual técnica utilizar para a análise de uma simulação distribuída depende de uma série de fatores, cabendo destacar:

- **O tipo de informação requerida**: dependendo da complexidade e das características do modelo, a utilização de algumas técnicas, como a solução analítica ou a coleta de dados, pode ser inviável;
- **A precisão necessária**: técnicas de aferição oferecem resultados mais precisos quando comparados com aqueles obtidos por meio de modelagem, mas deve-se garantir que a experimentação não interfira no comportamento da simulação e nos resultados obtidos. Por outro lado, simular um protocolo de sincronização para simulação distribuída é uma forma de avaliação flexível e de baixo custo, que pode ser empregada quando a precisão não for fundamental para a análise;
- **Ferramentas disponíveis**: a literatura apresenta alguns tipos de ferramentas para a análise de desempenho da simulação distribuída, como o uso de protótipos [Balakrishnan et al. 1997, Ferscha; Johnson 1996] e uma estrutura básica para avaliação de desempenho [Cavitt et al. 1996]. Porém, essas abordagens são específicas e voltadas para determinado protocolo de sincronização.

Como a precisão dos resultados é um ponto essencial para avaliar o desempenho do protocolo conservativo *CMB*, neste trabalho foram utilizadas técnicas de aferição. Para viabilizar a realização de medidas em um sistema real, foi desenvolvida a extensão funcional *ParSMPL*.

Com a utilização do *ParSMPL* é possível desenvolver diversos modelos de simulação e utilizar *benchmarks* e coleta de dados. Os *benchmarks* foram utilizados para verificar o tempo de transmissão das mensagens nulas e completas nas diversas plataformas empregadas, e a coleta de dados foi aplicada na execução da simulação distribuída.

4.3 *Benchmark* para Avaliação de Desempenho

A avaliação de desempenho de uma simulação distribuída pode ser realizada por intermédio dos tempos de execução da aplicação, utilizando as métricas citadas no capítulo 2 (*speedup* e eficiência). Como a extensão *ParSMPL* utiliza o paradigma de troca de mensagens para a implementação dos processos paralelos, medidas que representem a comunicação entre esses processos são importantes para determinar o comportamento da simulação.

O tempo necessário para a troca de uma mensagem entre dois processos paralelos pode ser dividido nas seguintes etapas[1] [Dillon et al. 1995]:

- **Tempo de transmissão** ($T_{send}(n)$): representa o tempo gasto pelas funções da biblioteca de comunicação no processamento da mensagem (por exemplo, o acréscimo de cabeçalhos e cálculo de *checksum*);

[1] No decorrer do capítulo ***n*** indica, em *bytes*, o tamanho das mensagens enviadas e recebidas.

- **Tempo de transposição na rede** ($T_{net}(n)$): representa o tempo necessário para o processo de transposição na rede de comunicação (sobrecarga imposta pelos protocolos de comunicação, como por exemplo o protocolo *TCP/IP*);
- **Tempo de recepção** ($T_{recv}(n)$): representa o tempo gasto no processamento da mensagem pelo processo receptor (verificação de *checksum* e retirada de cabeçalhos).

Considerando as etapas citadas anteriormente, o tempo de transmissão de uma mensagem entre dois processos A e B ($T_{A\to B}$) pode ser representado pela Eq. (4.1):

$$T_{A\to B} = T_{send}(n) + T_{net}(n) + T_{recv}(n) \qquad (4.1)$$

Porém, a falta de sincronização entre as máquinas dificulta o cálculo do tempo de transmissão utilizando apenas uma operação de comunicação. [Dillon et al. 1995] sugerem calcular o tempo de transmissão de uma mensagem de um processo origem A para um processo destino B e o tempo da mesma operação, transmitindo a mensagem do processo B para o processo A. Essa medida é definida como tempo de *round-trip* (Eq. (4.2)):

$$T_{A\leftrightarrow B}(n) = T_{A\to B}(n) + T_{B\to A}(n) \qquad (4.2)$$

Se os computadores em que os processos A e B executam são homogêneos, o atraso de comunicação ponto-a-ponto é definido pela Eq.(4.3):

$$T_{A\to B} = \frac{T_{A\leftrightarrow B}(n)}{2} \qquad (4.3)$$

Para obter maior fidelidade nos tempos obtidos, é necessário minimizar a concorrência nas máquinas envolvidas. No caso ideal tem-se um processo executando em cada máquina (origem e destino), reduzindo-se assim a competição pelos recursos da máquina e do meio de comunicação e, conseqüentemente, as influências externas [Senger 1997].

Outra medida importante é o *throughput*, que representa a taxa na qual as primitivas de comunicação podem transmitir uma mensagem de tamanho **n** em um intervalo de tempo (normalmente representada em *Mbits/s)*. A quantidade de dados transmitida é limitada pela capacidade de transmissão (*bandwidth*) do meio de comunicação, mas na prática apresenta valores menores, devido às sobrecargas de *software* e o congestionamento da rede. O *throughput* ($\rho(n)$), medido em *Mbits/s*, é definido pela Eq. (4.4):

$$\rho(n) = \frac{8*n}{T_{A\rightarrow B}(n)*10^6} \qquad (4.4)$$

As medidas definidas anteriormente (atraso de comunicação ponto-a-ponto e *throughput*) podem ser determinadas por intermédio da utilização de *benchmarks*, como o *Ping-Pong*, que obtém o tempo de *round-trip* de uma mensagem. [Dillon et al. 1995]. No desenvolvimento do algoritmo *Ping-Pong*, o processo origem A envia mensagens para o processo destino B e espera uma resposta. Após a recepção da mensagem, o processo B retorna a mensagem para o processo A. O algoritmo é ilustrado na figura 4.1.

```
Processo A
        Para contador = 1 até 1000
                Determina tempo inicial
                envia(mensagem)
                recebe(mensagem)
                Determina tempo final
        Fim Para
        Calcula tempo médio

Processo B
        Para contador = 1 até 1000
                recebe(mensagem)
                envia(mensagem)
        Fim Para
```

Figura 4.1 - *Benchmark Ping-Pong*. No algoritmo o processo A (origem) envia mensagem para o processo B (destino) e aguarda o retorno da mensagem para determinar o tempo de *round-trip*.

4.4 A Extensão *ParSMPL*

ParSMPL é uma ferramenta de simulação distribuída que utiliza o protocolo conservativo *CMB* para a sincronização entre os processos paralelos. Essa ferramenta é baseada no *SMPL*, uma extensão funcional da linguagem C para simulação discreta orientada a eventos [MacDougall 1987]. Assim, *ParSMPL* é uma extensão funcional da linguagem de programação C para simulação distribuída [Ulson et al. 1997].

O principal atrativo na utilização da simulação distribuída é a redução do tempo de processamento que em muitos casos, com a simulação seqüencial, chega a ser inviável. Porém, o esforço computacional exigido com a simulação seqüencial muitas vezes é substituído pelo esforço dos usuários para o desenvolvimento da simulação distribuída.

Este fato ocorre porque, apesar dos atrativos que a simulação distribuída pode oferecer, a tarefa de desenvolvimento dos programas paralelos não é trivial. Essa atividade exige conhecimentos não apenas sobre o sistema a ser modelado, técnicas de modelagem e simulação distribuída, mas também conhecimento sobre o paradigma de programação paralela utilizado. Esses fatores explicam porque a simulação distribuída é ainda pouco difundida entre a comunidade de usuários de simulação. Além disso, a falta de ferramentas adequadas para o desenvolvimento de simulações utilizando esse paradigma vem desencorajando ainda mais os possíveis usuários da simulação distribuída.

Para usuários com pouca experiência em programação paralela e mesmo para facilitar o trabalho daqueles já experientes, a ferramenta *ParSMPL* oferece os recursos necessários para a elaboração de programas de simulação distribuída, com o objetivo de avaliar o desempenho de sistemas computacionais. Utilizando essa extensão, o usuário não precisa se preocupar com os aspectos de sincronização e comunicação entre processos e com as características do protocolo de sincronização utilizado pela simulação distribuída. Todos esses mecanismos estão inseridos no *ParSMPL* e são transparentes ao usuário.

ParSMPL utiliza os ambientes de passagem de mensagens *PVM*[2] [Geist et al. 1994a, Geist et al. 1994b] e *PVMe*[3] [IBM Corporation 1996] para implementar a comunicação e sincronização entre os processos. A extensão foi desenvolvida na linguagem C, em plataformas *IBM/RS6000* e *IBM SP2* utilizando o sistema operacional *AIX*. A portabilidade foi um fator importante considerado no

[2] *PVM* domínio público versão 3.3.10 – *Oak Ridge National Laboratory*
[3] *IBM PVMe for AIX – IBM Corporation*

desenvolvimento da extensão, visando a criação de uma ferramenta que pudesse ser facilmente migrada para outros tipos de plataformas, como por exemplo máquinas compatíveis com *IBM-PC* executando *Windows95* e diferentes versões do *UNIX* (como *Linux* e *FreeBSD)*.

A figura 4.2 apresenta a hierarquia do *software* que compõe o ambiente *ParSMPL*. O usuário desenvolve suas aplicações em linguagem de programação de alto nível, podendo interagir também com as primitivas da extensão para simulação, com o ambiente de passagem de mensagens e com o sistema operacional. Desta forma, usuários mais experientes têm maior flexibilidade no desenvolvimento de suas aplicações.

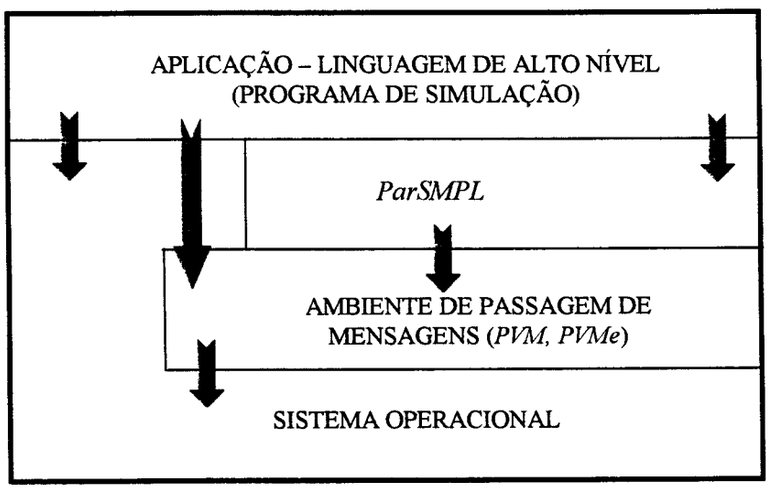

Figura 4.2 - Hierarquia dos Níveis de *Software* no *ParSMPL*. A figura apresenta o relacionamento entre as aplicações no ambiente *ParSMPL*. O usuário desenvolve a aplicação em linguagem de alto nível utilizando as primitivas da extensão, podendo interagir diretamente com o ambiente de passagem de mensagens e com o sistema operacional utilizado.

Duas etapas principais podem ser identificadas no processo de desenvolvimento do *ParSMPL*. Inicialmente, foi desenvolvida uma versão seqüencial da extensão funcional *SMPL* para plataforma *UNIX*, denominada *SMPLX*. Em uma segunda etapa, foram realizadas as alterações necessárias na versão seqüencial para o desenvolvimento da extensão distribuída.

4.5 Abordagem para o Desenvolvimento de um Programa de Simulação

O desenvolvimento de um programa de simulação pode ser realizado por quatro abordagens [Santana 1990]:

- **Desenvolvimento do programa de simulação em uma linguagem usual (como C ou Pascal).** Nesta abordagem o usuário tem liberdade para escolher uma linguagem que seja de seu conhecimento, porém tem que desenvolver todas as estruturas de dados e primitivas exigidas em um ambiente de simulação;
- **Utilização de um pacote de uso específico para criar o programa de simulação.** Nessa abordagem a construção do modelo é facilitada, sendo necessária apenas a parametrização do mesmo. Uma desvantagem é a necessidade de existência de um pacote que se adapte às necessidades do usuário;
- **Utilização de linguagens de simulação de uso geral.** Permitem que um número maior de sistemas seja modelado em relação aos pacotes específicos. As linguagens já contêm todas as estruturas necessárias para a criação de um ambiente de simulação, facilitando o trabalho do usuário.

No entanto, essa abordagem representa para o usuário a necessidade do aprendizado de uma nova linguagem;

- **Utilização de extensões funcionais de linguagens de programação convencionais.** Essas extensões são bibliotecas que fornecem as primitivas necessárias para o desenvolvimento da simulação, utilizando os recursos disponíveis da linguagem hospedeira.

Dentre as abordagens apresentadas anteriormente, a mais atrativa é a utilização das extensões funcionais, que facilitam o trabalho do usuário. Essas extensões não exigem o aprendizado de uma nova linguagem de programação e oferecem maior flexibilidade na transcrição do modelo para o programa de simulação.

SMPL constitui uma extensão funcional da linguagem de programação C para simulação discreta orientada a eventos em plataformas compatíveis com *IBM-PC* [MacDougall 1987]. Além da flexibilidade na implementação dos modelos e na alteração das rotinas (uma vez que o código fonte dessas rotinas está disponível), uma característica a ser ressaltada é a facilidade na utilização dessa extensão.

A extensão *SMPL* é utilizada pelo Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP como uma ferramenta para o desenvolvimento de programas de simulação. O grupo dispõe de um ambiente de simulação automático, denominado ASiA [Santana et al. 1996a, Santana et al. 1996b]. Nesse ambiente o programa de simulação em *SMPL* é gerado automaticamente a partir de uma especificação gráfica do modelo (a representação é baseada em modelos de redes de filas).

Esses fatores, aliados com as vantagens oferecidas pela extensão funcional, determinaram a escolha do *SMPL* como a biblioteca básica para o desenvolvimento do ambiente de simulação distribuída. Os programas de simulação distribuída utilizando a extensão *ParSMPL* têm sua estrutura semelhante a um programa seqüencial em *SMPL*, como discutido na seção 4.8.

4.5.1 Estrutura do *SMPL*

Na extensão funcional *SMPL* existem três entidades básicas para a representação dos modelos: facilidades, *tokens* e eventos [MacDougall 1987].

Cada facilidade representa um recurso (centro de serviço) do sistema que está sendo modelado, como por exemplo uma *CPU* (*Central Processing Unity*) em um modelo de um sistema computacional ou um barramento em um modelo de rede de comunicação. Um sistema corresponde a uma coleção de facilidades conectadas. Com as primitivas do *SMPL* é possível definir, reservar e liberar as facilidades, bem como verificar seu estado. A interconexão das facilidades é determinada pelo roteamento dos *tokens* entre as mesmas.

Os *tokens* representam as entidades ativas do sistema, como por exemplo uma tarefa em um modelo de um sistema computacional ou um pacote de dados em um modelo de rede de comunicação. *SMPL* fornece dois tipos básicos de operações para o controle do fluxo dos *tokens* na simulação, que envolvem a reserva de uma facilidade e o escalonamento de eventos. Se um *token* reserva uma facilidade que não tem servidores disponíveis, esse *token* é enfileirado até que algum servidor fique disponível e possa atendê-lo. O único atributo que *SMPL* disponibiliza aos *tokens* é a prioridade, o que permite a preempção das facilidades.

A mudança de estado de qualquer entidade do sistema representa um evento. Por exemplo, quando uma tarefa ou processo encerra sua execução e libera a facilidade *CPU*, dois eventos ocorrerão: o final da execução da tarefa e a liberação da *CPU*. Um evento é individualizado pelo seu número, o tempo de ocorrência e a identificação do *token* relacionado com esse evento. *SMPL* fornece rotinas para submeter e para selecionar eventos que vão ocorrer, além de ferramentas para depuração, coleta de dados, análises estatísticas e uma interface interativa para a execução de programas de simulação, denominada *mtr* [MacDougall 1987].

Os programas desenvolvidos em *SMPL* são caracterizados pela existência de um laço principal que comanda as iterações da simulação. Dentro desse laço os eventos são processados e os *tokens* são escalonados para novos eventos, e são realizadas operações para a coleta de informações estatísticas. A figura 4.3 apresenta a estrutura de um programa de simulação utilizando o *SMPL*.

```
#include <smpl.h>

main()
{
prepara variáveis de estado
inicia variáveis

facility()

schedule()

cause(evento, token)
{
        evento 1:

        evento 2:

        evento n:
}
report()
}
```

Figura 4.3 – Estrutura Básica de um Programa de Simulação em *SMPL*.

Em *SMPL* um único elemento básico de dados é utilizado para representar todas as estruturas de dados necessárias. *SMPL* mantém um *pool* de elementos que podem ser requisitados como descritores para os recursos, indicações para as filas desses recursos e indicações para os elementos da lista de eventos futuros. Cada elemento possui cinco campos conforme ilustra a figura 4.4. Com esses elementos, *SMPL* cria estaticamente as listas encadeadas e as estruturas necessárias para a simulação (lista de eventos futuros, fila de eventos e descritores de recursos).

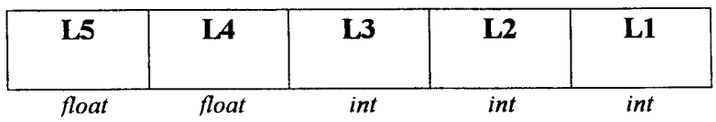

Figura 4.4 - Elemento Básico de Dados do *SMPL*. As estruturas de dados do *SMPL* são requisitadas a partir de um *pool* de elementos, baseado em unidades constituídas por cinco campos (L1, L2, L3, L4 e L5).

Os valores armazenados nos campos dos elementos dependem da estrutura que esse elemento representa. Se o elemento for requisitado como uma posição da lista de eventos futuros, devem ser armazenadas informações sobre o tempo de ocorrência do evento, o número do *token* e o número do evento. No caso de um descritor de recurso são armazenadas informações sobre o número de servidores do recurso, o número de servidores ocupados, o nome do recurso e contadores de fila e preempção. Para um elemento da fila de um recurso, devem ser armazenados identificação e prioridade do *token*, contadores para o número de vezes em que o recurso foi liberado e período de ocupação do recurso.

4.5.2 Primitivas do *SMPL*

SMPL fornece ao usuário um conjunto de primitivas que auxiliam o desenvolvimento de programas de simulação com orientação a eventos. As principais primitivas dessa extensão são descritas a seguir.

smpl(): responsável por iniciar o sistema para que a simulação possa ser executada. Atribui os valores iniciais às estruturas de dados e ao tempo de simulação (relógio);

facility(): essa primitiva cria, nomeia e retorna um descritor para cada facilidade do modelo. O descritor é utilizado para qualquer operação a ser realizada na facilidade, como a requisição ou o encerramento dos serviços. Cada facilidade possui um número *n* de servidores e uma fila de acesso. Quando a reserva da facilidade é executada, os servidores são examinados e se algum estiver disponível, esse será reservado para o *token* solicitante;

request(): essa primitiva é utilizada para requisitar o atendimento de um dos servidores de uma facilidade. Se um servidor está livre, esse servidor é reservado para o *token* solicitante. Se não existir servidor disponível, a primitiva retorna um valor indicando que a facilidade está ocupada. Neste caso, a requisição é inserida na fila da facilidade;

release(): libera um servidor de uma determinada facilidade. Após liberar o servidor, o sistema atualiza as estatísticas e decrementa o número de servidores ocupados da facilidade. Caso exista algum evento pendente na fila da facilidade, esse evento é inserido no início da lista de eventos futuros para execução imediata;

schedule(): responsável pelo escalonamento dos eventos da simulação. Todo evento a ser escalonado é inserido na lista de eventos futuros em ordem crescente do tempo de ocorrência do evento;

cause(): essa primitiva remove o elemento do início da lista de eventos futuros e avança o tempo da simulação (relógio) para o tempo de ocorrência do evento a ser executado.

O *SMPL* possui também diversas primitivas para a manipulação das filas de eventos dos recursos e da lista de eventos futuros, além de primitivas para análises estatísticas.

4.6 Versão Seqüencial do *SMPL* para Plataforma *UNIX - SMPLX*

A etapa inicial no desenvolvimento do *ParSMPL* foi a implementação de uma nova versão do *SMPL* para a plataforma *UNIX* (denominada *SMPLX*), uma vez que as bibliotecas de passagem de mensagens *PVM* e *PVMe*, utilizadas para a comunicação entre os processos, são desenvolvidas para essa plataforma.

O *PVM* foi adotado considerando-se que é um padrão de fato entre os ambientes de passagem de mensagens, sendo amplamente utilizado no desenvolvimento de aplicações paralelas. Outro fator determinante é que *PVM* já vem sendo utilizado pelo Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC-USP. Além do *PVM*, está disponível no Centro de Informática de São Carlos (CISC-USP) o *PVMe* (uma versão da *IBM* para o ambiente *PVM*) que explora as características do *switch* de alto desempenho do *IBM SP2* (as plataformas utilizadas pelo *ParSMPL* são descritas no capítulo 5).

A principal alteração efetuada para a criação do *SMPLX* ocorreu nas estruturas de dados da extensão. *SMPL* emprega estruturas de dados estáticas, requisitadas a partir de um *pool* de elementos básicos. A utilização de estruturas estáticas limita a complexidade dos modelos a serem implementados, uma vez que as dimensões das estruturas devem ser definidas em tempo de compilação.

No *SMPLX* o *pool* de elementos foi substituído por estruturas de dados dinâmicas, o que permite maior flexibilidade na implementação dos modelos. Cada recurso que compõe o modelo é representado por uma estrutura do tipo **Recurso,** que contém todas as informações sobre o mesmo. Para cada recurso existe uma fila com informações sobre os eventos não executados (estrutura **Fila_Eventos**) e os diversos servidores que compõem o recurso (estrutura **Lista_Servidores**). A figura 4.5 apresenta as estruturas de dados definidas para a nova versão da extensão funcional e as tabelas 4.2, 4.3 e 4.4 apresentam a descrição dessas estruturas.

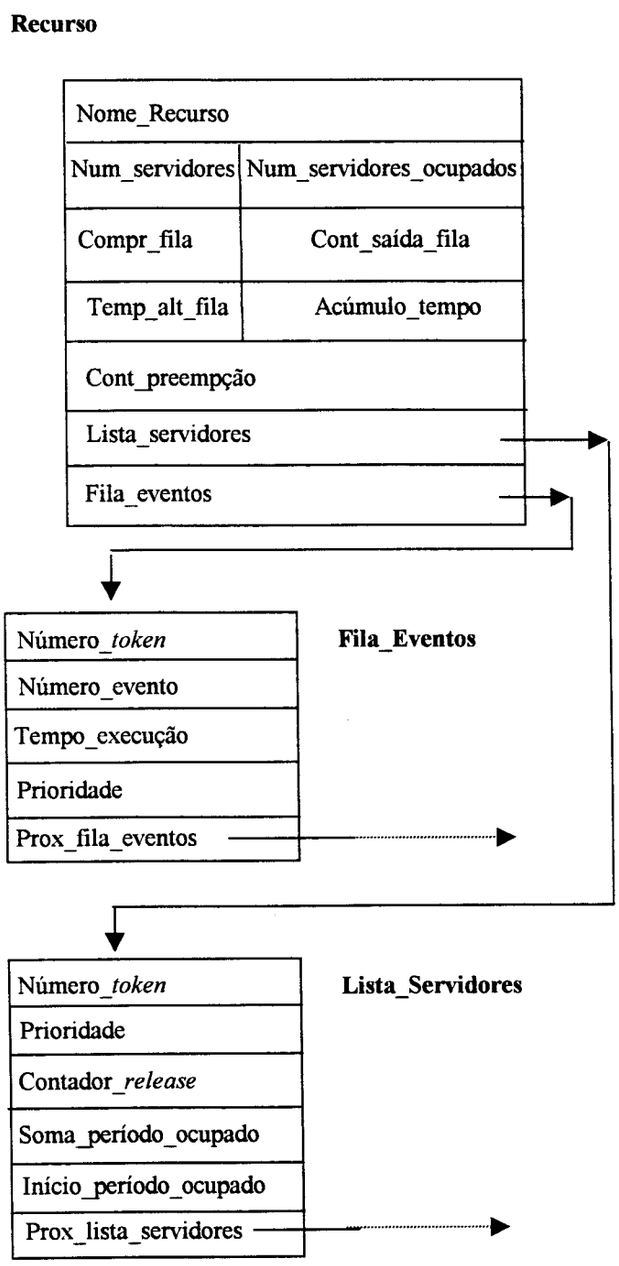

Figura 4.5 - Estruturas de Dados do *SMPLX*. As estruturas, caracterizadas por serem dinâmicas, foram desenvolvidas para a nova versão da extensão funcional.

Tabela 4.2 – Descrição das Estruturas de Dados do *SMPLX* – Recurso.

Campo	Descrição	Tipo
Nome_recurso	Identificação do recurso	*char*
Cont_preempção	Contador de preempção do recurso	*int*
Num_servidores	Número de servidores disponíveis para o recurso	*int*
Num_servidores_ocupados	Número de servidores ocupados	*int*
Compr_fila	Número de *tokens* na fila do recurso	*int*
Cont_saída_fila	Contador do número de vezes que um *token* deixa a fila do recurso	*int*
Temp_alt_fila	Tempo de ocorrência da última alteração da fila	*float*
Acúmulo_tempo	Somatório dos períodos de tempo ocupado, para cálculo do comprimento médio da fila	*float*
Lista_servidores	Informações sobre os servidores que compõem o recurso	*ponteiro*
Fila_eventos	Fila de eventos que aguardam execução no recurso	*ponteiro*

Tabela 4.3 – Descrição das Estruturas de Dados do *SMPLX* - Lista_Servidores.

Campo	Descrição	Tipo
Número_*token*	Identificação do *token*	*char*
Prioridade	Prioridade do *token* (para efetuar preempção)	*int*
Contador_*release*	Número de vezes em que o servidor é liberado	*int*
Soma_período_ocupado	Somatório do período em que o servidor ficou ocupado	*float*
Início_período_ocupado	Marca de tempo em que o servidor iniciou o atendimento ao *token*	*foat*
Prox_lista_servidores	Ponteiro para próximo elemento da lista	*ponteiro*

Tabela 4.4 – Descrição das Estruturas de Dados do *SMPLX* – Fila_Eventos.

Campo	Descrição	Tipo
Número_*token*	Identificação do *token*	*char*
Número_evento	Identificação do evento	*int*
Tempo_execução	Tempo de ocorrência do evento	*float*
Prioridade	Prioridade do *token*	*int*
Prox_fila_eventos	Ponteiro para próximo elemento da fila	*ponteiro*

Além das estruturas apresentadas na figura 4.5, *SMPLX* utiliza uma estrutura de lista encadeada para a representação da lista de eventos futuros (figura 4.6). A tabela 4.5 descreve as informações armazenadas nessa lista.

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

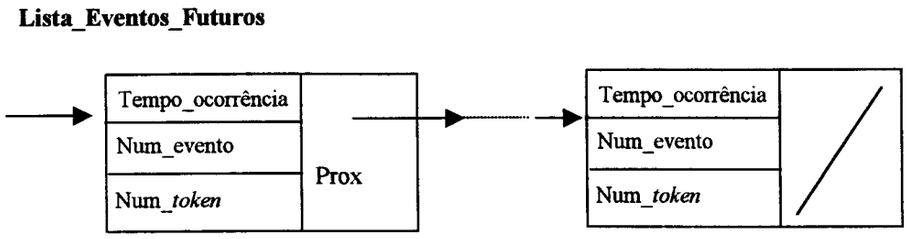

Figura 4.6 – Lista de Eventos Futuros no *SMPLX*. A estrutura representa a lista de eventos futuros implementada por meio de estruturas dinâmicas.

Tabela 4.5 – Descrição - Lista de Eventos Futuros.

Campo	**Descrição**	**Tipo**
Tempo_ocorrência	Tempo de ocorrência do evento	*float*
Num_evento	Identificação do evento	*int*
Num_*token*	Identificação do *token*	*int*
Prox	Próximo elemento da lista	*ponteiro*

O *SMPLX* foi validado por meio da comparação dos resultados obtidos com a execução de modelos descritos na literatura, utilizando-se a versão original do *SMPL* e a versão desenvolvida para *UNIX*, e também por intermédio da realização de análises estatísticas, com o cálculo de intervalos de confiança. As tabelas 4.6 e 4.7 apresentam os dados obtidos com uma execução da simulação para dois dos modelos utilizados (Servidor Central e Sistema Computacional Simplificado, detalhados no capítulo 5). A utilização é apresentada em porcentagem, o período médio ocupado é apresentado em unidades de tempo e o comprimento médio de fila é apresentado em número de tarefas.

Tabela 4.6 – Execução - Servidor Central.

IDENTIFICAÇÃO: SERVIDOR CENTRAL				
Extensão Funcional: ***SMPL***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,7067	10,004	0,628
	DISCO1	0,5245	29,977	0,195
	DISCO2	0,5282	29,781	0,219
	DISCO3	0,5328	30,097	0,224
	DISCO4	0,5276	29,862	0,208
Extensão Funcional: ***SMPLX***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,7067	10,005	0,629
	DISCO1	0,5243	29,976	0,197
	DISCO2	0,5280	29,783	0,217
	DISCO3	0,5330	30,094	0,228
	DISCO4	0,5273	29,865	0,209

Tabela 4.7 Execução – Sistema Computacional Simplificado.

IDENTIFICAÇÃO: SISTEMA COMPUTACIONAL SIMPLIFICADO				
Extensão Funcional: ***SMPL***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,4040	9,910	0,270
	DISCO	0,4195	20,375	0,290
Extensão Funcional: ***SMPLX***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,3997	9,949	0,272
	DISCO	0,4190	20,384	0,289

Na validação da extensão foram realizadas diferentes execuções, com variação na semente utilizada para a geração dos números aleatórios, e cálculo de intervalos de confiança.

Os resultados obtidos demonstram que a nova versão da extensão funcional é uma implementação válida do *SMPL* para a plataforma *UNIX*. O desenvolvimento dessa extensão vem de encontro com as necessidades da comunidade de usuários dessa plataforma por ferramentas que auxiliem no processo de avaliação de desempenho de sistemas computacionais. Essa ferramenta mantém as mesmas características do *SMPL*, uma vez que as alterações realizadas são transparentes ao usuário. Assim, um usuário familiarizado com a extensão *SMPL* não encontrará dificuldades para utilizar a nova extensão, uma vez que o código fonte desenvolvido pode ser migrado para as plataformas *UNIX* com poucas modificações.

Além das vantagens citadas, o fato dessa ferramenta ser uma extensão funcional permite ao usuário o desenvolvimento de programas de simulação com orientação a eventos sem a necessidade do aprendizado de uma nova linguagem.

4.7 Implementação do Protocolo *CMB*: A Extensão Funcional *ParSMPL*

Após a validação do *SMPLX*, foram realizadas as alterações nas primitivas dessa extensão para permitir a execução distribuída da simulação. Algumas das primitivas foram alteradas e outras foram introduzidas para viabilizar a comunicação e a sincronização entre os processos paralelos.

As estruturas de dados básicas da versão seqüencial foram mantidas. Foram introduzidas na lista de eventos futuros informações sobre a identificação dos processos para o escalonamento correto dos eventos e para a comunicação entre esses processos. Nesta seção são descritas a estrutura e as principais características do *ParSMPL*.

4.7.1 A Estrutura do *ParSMPL*

A extensão *SMPLX* foi modificada com a introdução de novas primitivas e mecanismos que permitiram mudar seu comportamento seqüencial, possibilitando assim a comunicação e a sincronização entre os processos paralelos e o desenvolvimento do *ParSMPL*. A estrutura de dados básica do *SMPLX* também foi alterada, incorporando novos dados necessários para o desenvolvimento de programas de simulação distribuída [Ulson et al. 1997].

Os processos da simulação são implementados utilizando-se processos padrão *UNIX* (criados com as primitivas do *PVM*). Todos os processos são criados no início da execução da simulação e destruídos no final da mesma, ou seja, não existe a criação dinâmica de processos (característica do protocolo *CMB*).

A extensão distribuída é constituída por dois módulos básicos: o núcleo, que contém as primitivas para comunicação, sincronização e prevenção do *deadlock*, e a biblioteca, que contém todas as primitivas necessárias para o desenvolvimento de um programa de simulação, além de primitivas para a coleta e análise estatística dos dados obtidos.

A implementação de uma simulação distribuída é composta por um conjunto de processos lógicos, cada um representando um processo no sistema real. Todas as interações entre os processos físicos são representadas por mensagens trocadas pelos processos lógicos correspondentes. Cada processo possui seu próprio relógio local (o mínimo entre os relógios de cada canal, como explicado no capítulo 3) que controla o seu tempo de simulação e uma lista de eventos na qual ficam armazenadas (em ordem cronológica) as mensagens recebidas de outros processos.

Cada processo tem um número determinado de canais lógicos bidirecionais através dos quais as mensagens são recebidas e enviadas. Nas plataformas utilizadas não existe um canal físico para a comunicação entre quaisquer dois elementos de processamento, como ocorre em arquiteturas baseadas em *transputers* [Almasi; Gottlieb 1994]. Todos os elementos de processamento disputam o meio de comunicação para a transmissão de suas mensagens. Para organizar a comunicação entre os processos no *ParSMPL* foram criadas abstrações (canais lógicos ou virtuais) e para possibilitar a utilização desses canais é utilizada uma tabela de identificação, que permite aos processos reconhecerem os demais processos da simulação.

Toda a comunicação entre os processos é efetuada utilizando-se as primitivas de comunicação não bloqueantes disponíveis no *PVM*, e todos os aspectos relativos à comunicação e sincronização entre os processos paralelos são transparentes ao usuário.

Os programas desenvolvidos no *ParSMPL* têm sua estrutura semelhante à estrutura de um programa de simulação desenvolvido no *SMPL*. *ParSMPL* foi projetado para permitir ao usuário liberdade na criação de seus próprios modelos, porém o usuário deve ser responsável por garantir que novos processos não sejam criados durante a execução da simulação, já que uma das características do protocolo *CMB* é a configuração estática dos canais de comunicação entre os processos.

Cada um dos processos que compõem a simulação possui a estrutura interna apresentada pela figura 4.7 [Ulson et al. 1997] e descrita na tabela 4.8. A criação de todas as estruturas de dados é executada automaticamente pelo *ParSMPL* e esse processo é totalmente transparente para o usuário.

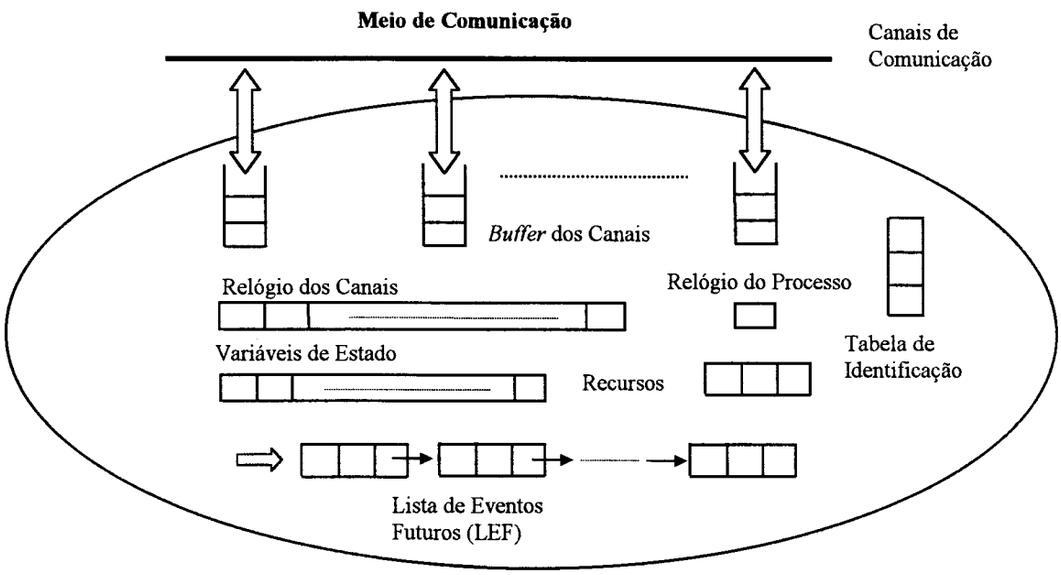

Figura 4.7 - Estrutura de Processos no *ParSMPL*. Cada processo lógico no *ParSMPL* apresenta a estrutura da figura. Todas as estruturas de dados são descritas na tabela 4.8.

Tabela 4.8 - Estruturas de Dados dos Processos Lógicos no *ParSMPL*.

Canais de Comunicação	Permitem aos processos enviar e receber mensagens completas (que são inseridas na Lista de Eventos Futuros do processo) e mensagens nulas.
***Buffer* de Canais**	Armazena as marcas de tempo de todas as mensagens recebidas pelo processo e que ainda não foram executadas.
Relógio dos Canais	Armazena o valor da marca de tempo da primeira posição do *buffer*, que indica o relógio do respectivo canal. Se o *buffer* estiver vazio, assume o valor da última mensagem recebida pelo canal. Se não recebeu mensagem, armazena o valor zero.
Relógio do Processo	Indica o menor valor entre os relógios dos canais e corresponde ao tempo da simulação em execução.
Lista de Eventos Futuros (LEF)	Contém os eventos a serem executados pelo processo, ordenados de maneira ascendente pelo tempo de execução.
Variáveis de Estado	Armazena as informações necessárias para a execução da simulação, como identificadores de processo e informações relativas à comunicação entre os processos e coleta de estatísticas.
Recursos	Armazena todas as informações sobre os recursos pertencentes ao processo.
Tabela de Identificação	Contém os identificadores que permitem a comunicação entre os processos, utilizando as primitivas do *PVM*.

4.7.2 Principais Rotinas do *ParSMPL*

Algumas rotinas do *SMPLX* foram alteradas e novas rotinas foram introduzidas para permitir a execução da simulação distribuída. As alterações realizadas e as primitivas desenvolvidas são descritas a seguir.

Rotinas Alteradas

smpl(): além das variáveis existentes na versão seqüencial e que foram mantidas (como o tempo de simulação), essa rotina inicia as novas variáveis e estruturas de dados, como os descritores de recursos, o descritor da lista de eventos futuros e os contadores de mensagens nulas e completas (o número de mensagens é uma das métricas utilizadas para avaliação do protocolo *CMB* e discutida no capítulo 5);

schedule(): antes de inserir um evento na lista de eventos futuros do processo, essa rotina verifica se o mesmo deve ser escalonado localmente ou em um outro processo (escalonamento remoto). Se o evento deve ser executado localmente, o mesmo é inserido na lista de eventos futuros do processo. Se o evento deve ser executado em um outro processo, as informações necessárias para a execução desse evento são enviadas ao processo destino;

cause(): essa rotina é acionada sempre que um evento deve ser executado e verifica se o tempo de ocorrência do próximo evento é menor ou igual ao relógio do processo. Em caso afirmativo, remove o elemento do início da lista de eventos futuros do processo e avança o tempo da

simulação local para o tempo de ocorrência do evento a ser executado. Caso contrário (se o processo está bloqueado) é acionado o mecanismo de envio de mensagens nulas para todos os processos com os quais o processo se comunica. Essa rotina também verifica se o processo tem alguma mensagem a receber. Se houver mensagem, essa é recebida, identificada e tratada corretamente, de acordo do seu tipo (mensagem nula ou completa). Se for uma mensagem completa (resultado de um escalonamento remoto), o evento correspondente é inserido na lista de eventos futuros do processo. Se for uma mensagem nula, o relógio do canal é atualizado.

Nas demais rotinas foram realizadas as adaptações necessárias em função das estruturas de dados dinâmicas que foram inseridas na extensão, além de informações necessárias para a comunicação entre os processos paralelos, como a identificação dos processos origem e destino.

Novas Rotinas

ReceiveMessage(): essa rotina é chamada pela rotina *cause()* para verificar se o processo tem alguma mensagem (completa ou nula) para ser recebida, conforme descrito no capítulo 3;

SendNull(): quando o processo é bloqueado, ou seja, quando o tempo de ocorrência do elemento do início da lista de eventos futuros for maior que o relógio do processo, ou então a lista de eventos estiver vazia,

mensagens nulas são enviadas para os demais processos com os quais o processo se comunica;

SendComplete(): envia uma mensagem completa para o processo destino contendo informações sobre o evento a ser escalonado, o tempo de ocorrência desse evento, a identificação do *token* e as identificações dos processos origem e destino;

InsertFEL(): insere um evento na lista de eventos futuros do processo;

InsertChannelBuf(): insere na estrutura de *buffer* de canais o tempo de ocorrência da mensagem recebida.

A hierarquia das principais rotinas no *ParSMPL* é apresentada na figura 4.8. No *ParSMPL* o usuário interage apenas com as primitivas do primeiro nível da hierarquia (destacadas por uma linha tracejada).

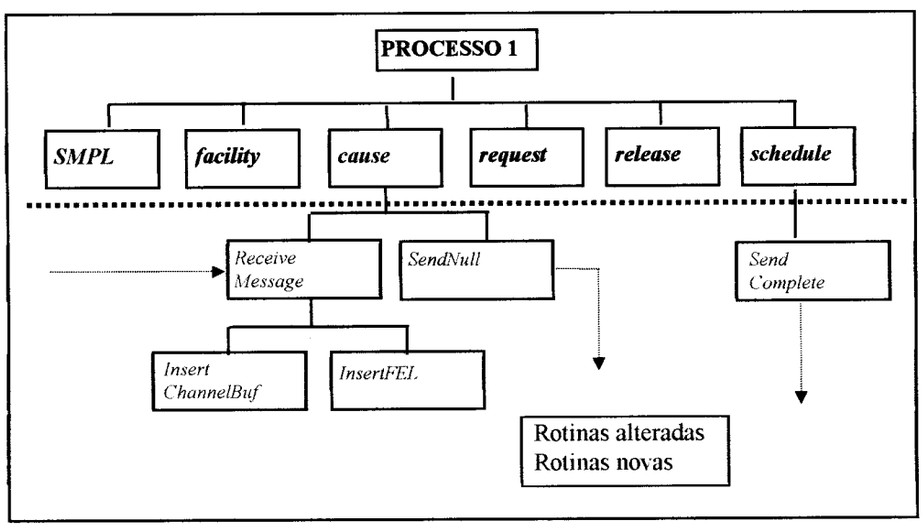

Figura 4.8 – Hierarquia das Rotinas no *ParSMPL*. A figura destaca as rotinas alteradas e as novas rotinas inseridas na extensão *ParSMPL*.

4.7.3 Tabela de Identificação de Processos

Como citado na seção 4.7.1, cada processo possui um número determinado de canais lógicos bidirecionais para receber e enviar mensagens. Para possibilitar a utilização desses canais é necessário preparar a tabela de identificação, que contém o identificador de todos os processos que compõem a simulação (esse identificador é um número inteiro que o *PVM* usa para a identificação das tarefas criadas no sistema).

A tabela de identificação contém uma entrada para cada processo lógico da simulação. Cada entrada é constituída por dois campos: um identificador de processo (inserido no código dos processos, assumindo valor 0, 1, ... n-1, onde n representa o número total de processos) e um identificador que o *PVM* retorna quando os processos são criados. A tabela 4.9 apresenta a estrutura da tabela de identificação.

Tabela 4.9 – Tabela de Identificação de Processos.

Identificação Processo	**Identificação *PVM***
0	Tid_processo0
1	Tid_processo1
⋮	⋮
n-1	Tid_processon-1

No *ParSMPL*, após a criação dos processos lógicos, a tabela de identificação é criada e enviada para todos os processos que compõem a simulação. A simulação tem início a partir do momento em que todos os processos receberam uma cópia dessa tabela. A criação da tabela é efetuada pelo processo mestre ou processo pai em função do modelo de programação utilizado. Esses modelos são descritos na seção 4.7.4.

4.7.4 O Modelo de Programação no *ParSMPL*

No desenvolvimento dos programas de simulação utilizando o *ParSMPL* é empregado o paradigma *MPMD*, ou seja, os processadores armazenam partes distintas do código, dependendo do processo a ser executado. Dentro desse paradigma, pode-se utilizar dois modelos de programação no *ParSMPL*. No modelo 1 um processo especial, denominado processo pai, é responsável por iniciar todos os demais processos e preparar a tabela de identificação (paradigma mestre/escravo). No modelo 2 um dos processos que compõem a simulação é responsável por essa tarefa. Os modelos de programação do *ParSMPL* são apresentados nas figura 4.9 e 4.10.

- **Modelo de Programação 1: Utilização do processo pai**

Nesse modelo o processo Pai não representa nenhum processo no sistema físico simulado. Sua função é controlar a execução da simulação, definindo e enviando a tabela de identificação aos demais processos lógicos que compõem a simulação e criando os processos filhos. A figura 4.9 apresenta um exemplo com o processo pai e três processos filhos.

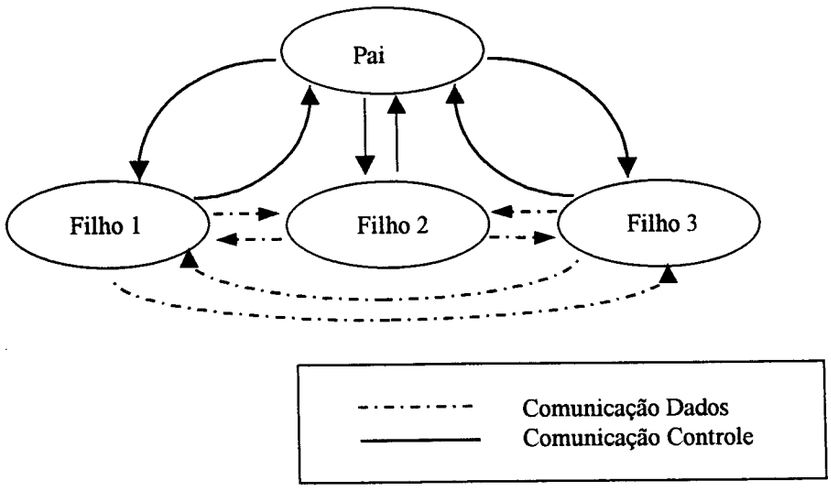

Figura 4.9 - Modelo de Programação 1 no *ParSMPL*. Nesse modelo o processo Pai é responsável apenas pela iniciação e controle da simulação.

- **Modelo de Programação 2: Sem a utilização do processo pai**

Como a criação do processo pai em alguns casos pode gerar uma sobrecarga desnecessária (a execução do processo pai concorrendo com a execução de um processo filho), foi criado outro modelo de programação no *ParSMPL*. Nesse modelo não existe a figura do processo pai e um dos processos que compõem a simulação (chamado processo mestre) é responsável também por iniciar os demais processos e preparar a tabela de identificação. A figura 4.10 apresenta um exemplo com três processos lógicos executados em elementos de processamento distintos.

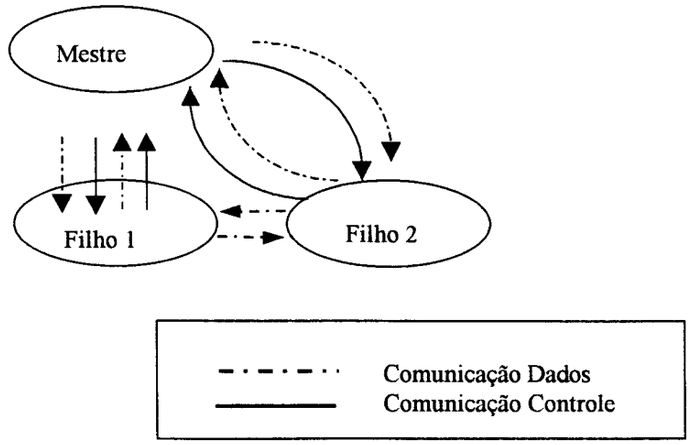

Figura 4.10 - Modelo 2 de Programação no *ParSMPL*. Nesse modelo um dos processos que compõe a simulação (processo mestre) também é responsável pela iniciação e controle dos demais processos.

4.7.5 Tipos de Mensagens

De acordo com as características do protocolo *CMB*, cada processo no *ParSMPL* pode enviar para outro processo dois tipos distintos de mensagens, denominadas mensagens completas e mensagens nulas.

Mensagens Completas

Os *tokens* que circulam pelo sistema de simulação são escalonados para eventos específicos, que podem ser locais ou remotos. Isso implica então em escalonamento local e escalonamento remoto de mensagens. O escalonamento local ocorre quando o evento deve ser executado no processo em questão.

O escalonamento remoto ocorre quando o evento a ser executado deve ser escalonado para processamento em outro processo lógico. Neste caso é necessário enviar ao processo destino todas as informações por meio de uma mensagem

completa para que ocorra o escalonamento. A figura 4.11 apresenta a estrutura de uma mensagem completa.

Te (*float*)	Num_evento(*int*)	*Token* (*int*)	Ident_Origem*(int)*	Ident_destino(*int*)

Te: Tempo de ocorrência do evento
Num_evento: Identificação do evento a ser escalonado
Token: Identificação do *token*
Ident_origem: Identificação do processo emissor da mensagem
Ident_destino: Identificação do processo receptor da mensagem

Figura 4.11 - Estrutura de uma Mensagem Completa. A mensagem completa é constituída por cinco campos, que identificam o número do evento, o tempo de ocorrência desse evento, o *token* associado ao evento, o processo origem e o processo destino da mensagem.

Mensagens Nulas

De acordo com o protocolo *CMB*, quando um processo está bloqueado, deve enviar mensagens nulas para os demais processos com os quais se comunica, para evitar que a simulação entre em *deadlock*.

As mensagens nulas não representam qualquer evento do sistema real, apenas permitem que os processos lógicos atualizem seus relógios internos, com o objetivo de permitir o progresso da simulação e evitar a ocorrência de *deadlock*. O incremento de tempo das mensagens nulas é determinado em função do modelo que está sendo simulado. A figura 4.12 apresenta a estrutura de uma mensagem nula.

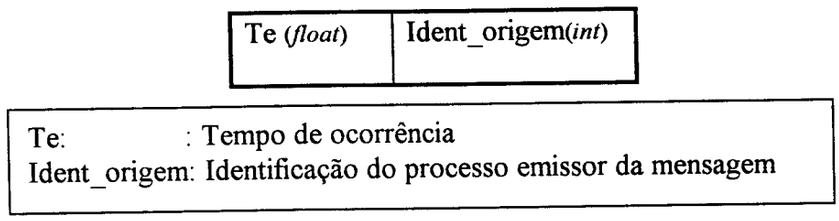

Figura 4.12 - Estrutura de uma Mensagem Nula. As mensagens nulas são constituídas por dois campos, contendo informações sobre o tempo de ocorrência e a identificação do processo emissor da mensagem.

4.8 Implementação dos Processos e Comunicação no *ParSMPL*

Os programas de simulação desenvolvidos em *SMPL* são caracterizados pela existência de um laço principal que comanda as iterações da simulação. Dentro desse laço os eventos são executados e escalonados para ocorrência, além da execução de operações para coleta de informações estatísticas.

No *ParSMPL* os programas de simulação têm estrutura semelhante aos programas desenvolvidos em *SMPL*, tornando transparente para o usuário os mecanismos de comunicação e sincronização entre os processos. As rotinas que implementam os mecanismos de simulação no *ParSMPL* têm nomes similares às rotinas do *SMPL*, com o intuito de facilitar a utilização da extensão pelos usuários. A figura 4.13 apresenta a estrutura dos processos lógicos para o modelo de programação 2. Para o modelo 1, a estrutura dos processos filhos continua a mesma, e o processo pai (principal) contém apenas as linhas identificadas por um (*) no processo mestre.

PROCESSO MESTRE	**PROCESSO FILHO**
Inclui bibliotecas + núcleo *ParSMPL*	Inclui bibliotecas + núcleo *ParSMPL*
Inicia Processos (*)	
Prepara Tabela Identificação (*)	
Envia Tabela de Identificação (*)	
	Recebe Tabela de Identificação
Prepara Variáveis de Estado	Prepara Variáveis de Estado
Inicia variáveis	Inicia variáveis
smpl()	*smpl()*
facility()	*facility()*
schedule()	*schedule()*
cause(evento, token)	*cause(evento, token)*
Evento 1:	Evento 1:
Evento 2:	Evento 2:
Evento n:	Evento n:
Recebe Resultado (*)	Envia Resultado
Encerra Processo (*)	Encerra Processo

Figura 4.13 – Descrição de um Processo no *ParSMPL*. A estrutura dos processos no *ParSMPL* é similar com a estrutura dos processos no *SMPL*.

A figura 4.14 [Ulson et al. 1997] apresenta a estrutura com as principais rotinas utilizadas na comunicação entre os processos no *ParSMPL*. Dentro do laço principal, no procedimento *cause()*, o processo executa *ReceiveMessage()* para saber se existe alguma mensagem pendente no *buffer* de recepção. Caso exista alguma mensagem, essa é identificada (completa ou nula) e a rotina correspondente é executada: *InsertFEL()* para inserir a mensagem completa na lista de eventos futuros e *InsertChannelBuf()* para inserir a marca de tempo da mensagem no *buffer* do respectivo canal.

Se existe evento a ser executado, *cause()* retira esse evento da lista de eventos futuros e o executa. Para escalonar um novo evento para execução, *schedule()* identifica se o evento é local ou remoto. Se for um evento local, *schedule()* insere o evento na lista de eventos futuros do processo. Se for um evento remoto *SendComplete()* é executada, enviando uma mensagem completa para o processo destino.

Caso não existam eventos a serem executados (quando a marca de tempo do evento que está no início da lista de eventos futuros é maior que o relógio do processo ou a lista de eventos está vazia) a rotina *SendNull()* é executada para enviar mensagens nulas para os processos com os quais o processo em questão se comunica. A condição de término da simulação é determinada no processo responsável por iniciar a simulação (processo pai ou processo mestre).

FSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

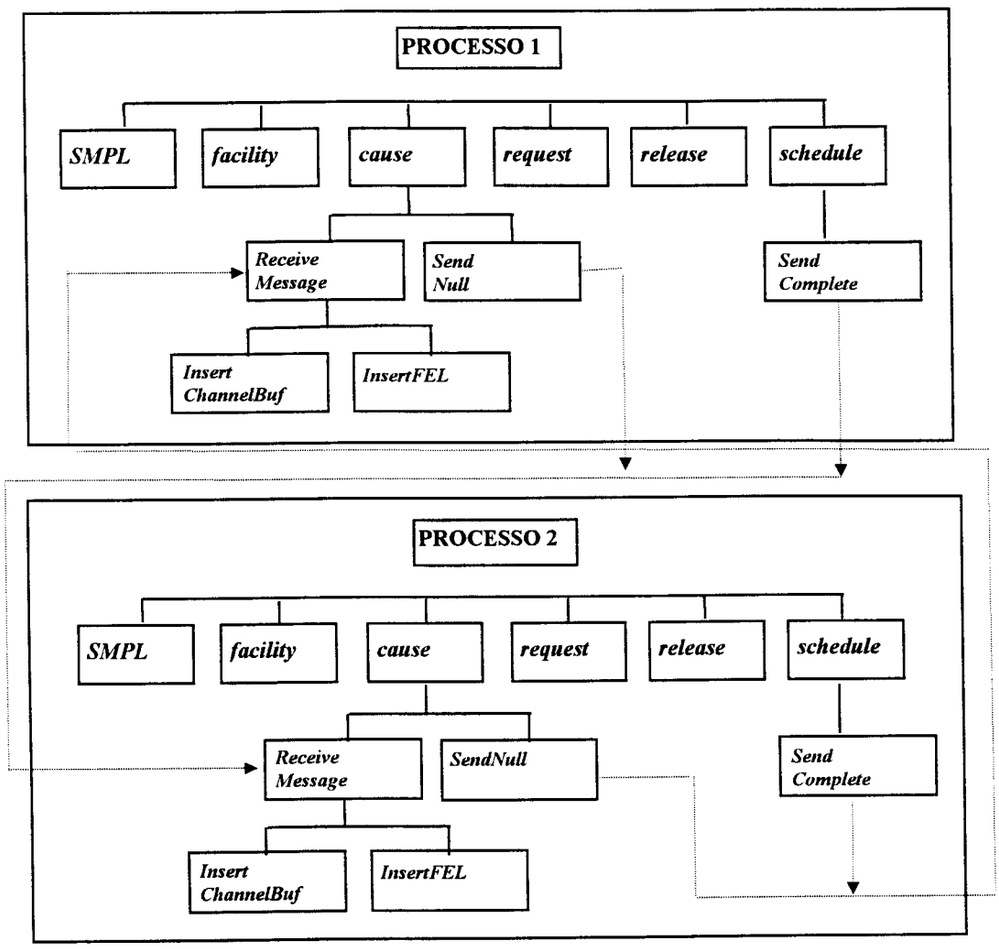

Figura 4.14 - Comunicação entre Processos no *ParSMPL*. A figura apresenta os principais módulos responsáveis pelo controle da comunicação entre os processos na simulação distribuída, considerando dois processos lógicos.

Cabe ressaltar que, com a utilização do *ParSMPL*, o usuário deve estar familiarizado apenas com as características de um programa de simulação orientado a eventos e ter noções sobre programação paralela para poder criar o código dos processos. O usuário não precisa ter conhecimento de programação em *PVM/PVMe*, uma vez que todas as operações de comunicação e sincronização são transparentes. Isto é, o usuário interage apenas com as rotinas da primeira camada da estrutura

apresentada na figura 4.14 (algumas dessas rotinas são *SMPL()*, *facility()*, *cause()*, *request()*, *release()* e *schedule()*).

4.9 Exemplo de Utilização do *ParSMPL*

O desenvolvimento de uma simulação distribuída utilizando o *ParSMPL* constitui uma tarefa simples, uma vez que não exige do usuário o prévio conhecimento das características do protocolo de sincronização ou mesmo do paradigma de programação paralela utilizados. Para um usuário já familiarizado com a simulação orientada a eventos, esse processo pode ser descrito nas seguintes etapas:

1. Desenvolver a aplicação em *SMPL* considerando o paradigma da simulação seqüencial;
2. Determinar a forma de partição dos processos lógicos (ver capítulo 6);
3. Definir o código correspondente a cada processo lógico, a partir da implementação seqüencial;
4. Configurar os parâmetros do *ParSMPL* e os parâmetros das primitivas alteradas do *SMPL*;
5. Gerar o código executável e executar a simulação distribuída.

Para usuários com maior experiência no desenvolvimento de aplicações com esse paradigma, os passos 1 e 3 podem ser substituídos pela implementação direta do código dos processos lógicos. Como exemplo para o processo de desenvolvimento pode-se considerar o Modelo do Servidor Central, descrito no capítulo 5. O código correspondente em *SMPL*, considerando uma versão simplificada para a simulação seqüencial do modelo é apresentado na figura 4.15.

```
1.  #include "smpl.h"

2.  #define TASKS 50

3.  main()
4.  {
5.  int
6.  indice,
7.  token,
8.  evento_execut,
9.  cpu,
10. disco1,
11. disco2,
12. disco3,
13. disco4,
14. num_tours = 10000,
15. num = num_tours;

16. double
17. tscpu = 10.0,
18. td = 30.0,
19. sd = 7.5;

20. smpl(0,"Processo CPU");
21. cpu = facility("cpu",1);
22. disco1 = facility("disco1",1);
23. disco2 = facility("disco2",1);
24. disco3 = facility("disco3",1);
25. disco4 = facility("disco4",1);

26. for (indice=1; indice<= TASKS; indice++)
27. schedule(1,0.0,indice);

28. while (num_tours)
29. {
30. cause(&evento_execut, &token);
31. switch(evento_execut)
32. {
33. case 1:
34. schedule(2,0.0,token);
35. break;

36. case 2:
37. if (request(cpu,2,token,0) == 0 )
38. schedule(3,expntl(tscpu),token);
39. break;

40. case 3:
41. release("cpu",token);
42. num_tours--;
43. indice = 2*random(2,5);
44. schedule(indice,0.0,token);
45. break;

46. case 4:
47. if (request(disco1,4,token,0) == 0)
48. schedule(5,erlang(30.0,7.5),token);
49. break;

50. case 5:
51. release(disco1,token);
52. schedule(1,0.0,token);
53. break;

54. case 6:
55. if (request(disco2,6,token,0) == 0)
56. schedule(7,erlang(30.0,7.5),token);
57. break;

58. case 7:
59. release(disco2,token);
60. schedule(1,0.0,token);
61. break;

62. case 8:
63. if (request(disco3,8,token,0) == 0)
64. schedule(9,erlang(td,ts),token);
65. break;

66. case 9:
67. release(disco3,token);
68. schedule(1,0.0,token);
69. break;

70. case 10:
71. if (request(disco4,10,token,0 == 0)
72. schedule(11,erlang(30.0,7.5),token);
73. break;

74. case 11:
75. release(disco4,token);
76. schedule(1,0.0,token);
77. break;

78. }
79. report(); /* resultados */
80. }
81. }
```

Figura 4.15 - Modelo do Servidor Central – Código em *SMPL*. A figura apresenta o código para a simulação seqüencial em *SMPL* do modelo da figura 5.2.

Considerando a opção pela configuração dos processos lógicos apresentada na figura 5.2, constituída por dois processos lógicos, o usuário deve criar os códigos

correspondentes a esses processos (utilizando o paradigma *MPMD*). Desta forma, devem ser criados dois programas representando os processos lógicos PL0, que inclui o recurso CPU, e PL1 que agrupa os recursos DISCOS.

O código simplificado obtido para os processos lógicos utilizando o modelo de programação 2 é apresentado nas figuras 4.16 e 4.17. Nas figuras são apresentadas as alterações necessárias para a utilização das rotinas do *ParSMPL* e as primitivas para a execução da simulação. As demais primitivas têm estrutura semelhante às primitivas do *SMPL*, com algumas alterações no tipo e número dos parâmetros dessas primitivas.

No exemplo os processos lógicos são identificados por CPU e DISCO_1. Considerando o desenvolvimento da simulação distribuída com o *ParSMPL*, as linhas destacadas identificam as primitivas executadas para a preparação da tabela de identificação e o envio dessa tabela para os processos filhos no processo lógico 1 (linhas 19 a 37 - figura 4.16) e recepção da tabela de identificação no processo lógico 2 (linhas 15 a 25 – figura 4.17). O usuário pode alterar o *lookahead* dos processos por meio do comando das linhas 18 (figura 4.16) e 14 (figura 4.17).

Além dessas primitivas, o usuário deve incluir no código dos processos a biblioteca *parsmpl.c* (linha 1 – figuras 4.16 e 4.17) e identificar os processos que vão constituir a simulação (linhas 2 e 3 – figuras 4.16 e 4.17). O comando das linhas 63 (figura 4.16) e 71 (figura 4.17) encerra o processo no *PVM*.

```
1.  #include "/u/rsulson/pvm3/parsmpl.c"

2.  #define CPU        0
3.  #define DISCO_1 1
4.  #define TASKS 50

5.  main()
6.  {
7.  int
8.  tscpu = 10.0,
9.  tid_processo_pai,
10. tid_cpu,
11. tid_disco1,
12. tabela_aux[10],
13. indice,
14. token,
15. evento_execut,
16. num_tours = 50000,
17. num = num_tours;

18. tempo_processo = 1.0;
19. proc_local = 0;

20. for (indice = 1; indice <10; indice++)
21. {
22. tabela_tids[indice] = 0;
23. tabela_aux[indice] =0;
24. }

25. tid_processo_pai = pvm_mytid();
26. pvm_spawn("d",(char**)0,0,"",1,&tid_disco);
27. tabela_tids[CPU] = tid_processo_pai;
28. tabela_tids[DISCO_1] = tid_disco1;
29. tabela_aux[CPU] = tid_processo_pai;
30. tabela_aux[DISCO_1] = tid_disco1;

31. for (indice = 0; indice < 10 ; indice++)
32. if (indice != DISCO_1)
33. clock_canais[indice] = -1.0;

34. Clock_canais[DISCO_1] = 0.0;
35. pvm_initsend(PvmDataRaw);
36. pvm_pkint(tabela_aux, 10, 1);
37. pvm_send(tid_disco1,1);

38. smpl(0,"Processo CPU");
39. for (indice=1; indice<= TASKS; indice++)
40. task[indice].ts = 0.0;

41. facility("cpu",1);

42. for (indice=1; indice<= TASKS; indice++)
43. schedule(1,0.0,indice,CPU);

44. while (num_tours)
45. {
46. cause(&evento_execut, &token);
47. switch(evento_execut)
48. {
49. case 1:
50. schedule(2,0.0,token,CPU);
51. break;

52. case 2:
53. if (request("cpu",2,token,0,CPU) == 0 )
54. schedule(3,expntl(tscpu),token,CPU);
55. break;

56. case 3:
57. release("cpu",token);
58. num_tours--;
59. indice = (2*randompar(0,3))+1;
60. schedule(indice, clock_sim(),
    token,DISCO_1);
61. break;
62. );

63. parsmpl_pvm_exit();
64. /* resultados*/
65. }
```

Figura 4.16 - Modelo do Servidor Central – Código Processo Lógico 0 – *ParSMPL*. A figura apresenta o código correspondente ao processo lógico PL0, que representa o recurso *CPU* do modelo da figura 5.2.

```
1.  #include "/u/rsulson/pvm3/parsmpl.c"

2.  #define CPU    0
3.  #define DISCO_1 1

4.  main()
5.  {
6.  int tid_processo_pai,
7.  my_tid,
8.  indice,
9.  contador,
10. token,
11. evento_execut;

12. double td = 30.0,
13. sd = 7.50;

14. tempo_processo = 1.0;
15. proc_local = DISCO_1;
16. my_tid = pvm_mytid();
17. tid_processo_pai = pvm_parent();

18. pvm_recv(tid_processo_pai,-1);
19. pvm_upkint(tabela_tids,10,1);
20. for (indice = 2; indice < 10; indice++)
21. tabela_tids[indice] = 0;
22. for (indice = 0; indice < 10; indice++)
23. if (indice != CPU)
24. clock_canais[indice] = -1.0;

25. clock_canais[CPU] = 0.0;

26. smpl(0,"Processo Disco 1");

27. facility("disco1",1);
28. facility("disco2",1);
29. facility("disco3",1);
30. facility("disco4",1);

31. contador = 10000;

32. while(contador != 0)
33. {
34. cause(&evento_execut, &token);
35. switch(evento_execut)
36. {
37. case 1:
38. if (request("disco1",1,token,0,DISCO_1) == 0 )
39. schedule(2,erlang(td,sd),token,DISCO_1);
40. break;

41. case 2:
42. release("disco1",token);
43. schedule(1,clock_sim(),token,CPU);
44. break;

45. case 3:
46. if (request("disco2",1,token,0,DISCO_1) == 0 )
47. schedule(4,erlang(td,sd),token,DISCO_1);
48. break;

49. case 4:
50. release("disco2",token);
51. schedule(1,clock_sim(),token,CPU);
52. break;

53. case 5:
54. if (request("disco3",1,token,0,DISCO_1) == 0 )
55. schedule(6,erlang(td,sd),token,DISCO_1);
56. break;

57. case 6:
58. release("disco3",token);
59. schedule(1,clock_sim(),token,CPU);
60. break;

61. case 7:
62. if (request("disco4",1,token,0,DISCO_1) == 0 )
63. schedule(8,erlang(td,sd),token,DISCO_1);
64. break;

65. case 8:
66. release("disco4",token);
67. schedule(1,clock_sim(),token,CPU);
68. break;

69. }
70. }

71. parsmpl_pvm_exit();
72. }
```

Figura 4.17 - Modelo do Servidor Central – Código Processo Lógico 1 – *ParSMPL*. A figura apresenta o código correspondente ao processo lógico PL1, que representa as unidades de disco do sistema considerando o modelo da figura 5.2.

Como pode ser observado nas figuras 4.16 e 4.17, o código correspondente aos processos lógicos PL0 e PL1 são semelhantes ao programa de simulação em *SMPL*. Apesar da inclusão das primitivas para a criação e comunicação entre os

processos lógicos, a estrutura do programa de simulação seqüencial é mantida na implementação da simulação distribuída. O trabalho maior do usuário consiste na inclusão das primitivas de comunicação do *ParSMPL*, envolvendo a criação e a identificação dos processos lógicos e a preparação e o envio da tabela de identificação desses processos.

Uma vez definida a estrutura dos processos lógicos, essa estrutura pode ser utilizada como um gabarito, simplificando e agilizando o trabalho do usuário. Desta forma, mesmo um usuário leigo em simulação distribuída e até mesmo em programação paralela pode utilizar o *ParSMPL* para o desenvolvimento de programas com o paradigma da simulação distribuída.

4.10 Validação do *ParSMPL*

A validação do *ParSMPL* foi realizada pela comparação dos resultados da execução de modelos nessa extensão e em *SMPL*, e pela análise estatística com o cálculo de intervalos de confiança.

Como exemplo, os resultados obtidos para uma execução de dois dos modelos utilizados (Servidor Central e Sistema Computacional Simplificado, descritos no capítulo 5) são apresentados nas tabelas 4.10 e 4.11 considerando as execuções em *SMPL* e em *ParSMPL* e uma das plataforma utilizadas neste trabalho (as plataformas são descritas no capítulo 5). Os dados coletados foram a utilização da facilidade, período médio ocupado e comprimento médio da fila, apresentados para os recursos (facilidades) que constituem os modelos.

Tabela 4.10 - Validação do *ParSMPL* – Servidor Central.

IDENTIFICAÇÃO: SERVIDOR CENTRAL				
Extensão Funcional: ***SMPL***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,7067	10,004	0,628
	DISCO1	0,5245	29,977	0,195
	DISCO2	0,5282	29,781	0,219
	DISCO3	0,5328	30,097	0,224
	DISCO4	0,5276	29,862	0,208
Extensão Funcional: ***ParSMPL (SP2/PVMe)***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,7068	10,004	0,629
	DISCO1	0,5243	29,998	0,190
	DISCO2	0,5287	29,782	0,215
	DISCO3	0,5327	30,099	0,226
	DISCO4	0,5271	29,859	0,201

Tabela 4.11 - Validação do *ParSMPL* – Sistema Computacional Simplificado.

IDENTIFICAÇÃO: SISTEMA COMPUTACIONAL SIMPLIFICADO				
Extensão Funcional: ***SMPL***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,4040	9,910	0,270
	DISCO	0,4195	20,375	0,290
Extensão Funcional: ***ParSMPL (SP2/PVMe)***				
	FACILIDADE	**UTILIZAÇÃO**	**PERÍODO MÉDIO OCUPADO**	**COMPRIMENTO MÉDIO FILA**
	CPU	0,3999	9,905	0,276
	DISCO1	0,4199	20,384	0,289

Os resultados obtidos com os modelos citados anteriormente mostram que *ParSMPL* representa uma implementação confiável do *SMPL* e constitui uma ferramenta poderosa e flexível para o desenvolvimento de programas utilizando o paradigma da simulação distribuída.

Além das vantagens oferecidas pelo fato do *ParSMPL* ser uma extensão funcional, a semelhança na estrutura de um programa de simulação desenvolvido com o *ParSMPL* e um programa de simulação em *SMPL* facilita a tarefa de migração da simulação seqüencial para o paradigma distribuído.

A transparência na implementação da comunicação e sincronização entre os processos paralelos e na implementação das características do protocolo *CMB* fazem do *ParSMPL* uma ferramenta atrativa também para usuários que não dominam as técnicas de programação paralela e simulação distribuída. Deve-se ressaltar que *ParSMPL* pode ser migrado para outros tipos de plataformas, como *Windows95* e *UNIX*, e mesmo outros ambientes de passagem de mensagens, como *MPI*.

4.11 Considerações Finais

Este capítulo apresentou o desenvolvimento das extensões funcionais *SMPLX* e *ParSMPL*, visando a execução da simulação seqüencial e distribuída em plataformas *UNIX*. O desenvolvimento dessas extensões justifica-se pelo fato da simulação ser uma ferramenta poderosa na avaliação de desempenho de sistemas computacionais e pela existência de poucos ambientes desse tipo para simulação distribuída. Os ambientes existentes atualmente caracterizam-se pela complexidade e por exigir do usuário o aprendizado de técnicas e particularidades que dificultam o desenvolvimento da simulação distribuída, como *Yaddes*, *Parsec* e *CPSIM*.

Com o desenvolvimento do *SPMLX*, além dos benefícios oferecidos pelo fato de ser uma extensão funcional, pode-se oferecer ao usuário uma ferramenta flexível para o desenvolvimento da simulação orientada a eventos. A semelhança com o *SMPL* e a possibilidade de migração das aplicações para diferentes plataformas *UNIX*

constituem atrativos dessa extensão.

O desenvolvimento de uma simulação exige do usuário conhecimentos não só sobre o sistema a ser simulado, mas também das técnicas de simulação. No paradigma distribuído, existe ainda a necessidade de conhecimentos sobre programação paralela, o que torna esse processo complexo e limita a sua utilização. Além disso, o número reduzido de ferramentas para o desenvolvimento desse tipo de simulação dificulta ainda mais o trabalho e restringe a utilização do paradigma da simulação distribuída.

A extensão *ParSMPL* vem de encontro com essas necessidades, oferecendo ao usuário a possibilidade de utilizar todos os benefícios da simulação distribuída sem exigir conhecimentos específicos. A atenção e o esforço do usuário se restringem ao sistema a ser modelado. As principais características do ambiente que devem ser ressaltadas são: portabilidade, detalhes da comunicação e sincronização transparentes ao usuário e disponibilidade de funções estatísticas e relatório para análise dos resultados.

É importante destacar a portabilidade da extensão *ParSMPL*, que pode ser facilmente migrada para diversos tipos de plataformas, desde *cluster* de microcomputadores e estações de trabalho até arquiteturas paralelas. Além disso, o desenvolvimento modular do ambiente permite ao usuário a realização de modificações nas características do protocolo por meio de alterações em rotinas específicas, sem a necessidade de alterações em todo o código da extensão.

Deve-se salientar também o fato de *ParSMPL* ser uma extensão funcional, que facilita o desenvolvimento dos programas de simulação. Não exige do usuário a aprendizagem de uma nova linguagem de programação e oferece maior flexibilidade,

uma vez que todos os recursos da linguagem de programação utilizada continuam disponíveis para o usuário.

O *ParSMPL* foi utilizado na implementação de modelos cujos resultados permitiram a realização das análises sobre o comportamento do protocolo *CMB*. O capítulo 5 apresenta os diferentes modelos de sistemas computacionais definidos para a avaliação do protocolo, as métricas empregadas no processo de avaliação e as características das plataformas utilizadas neste trabalho.

CAPÍTULO 5

DESENVOLVIMENTO DE MODELOS PARA A AVALIAÇÃO DE DESEMPENHO DO PROTOCOLO *CMB*

Este capítulo discute os principais tópicos relacionados com avaliação de desempenho, métricas, plataformas e modelos utilizados na realização dos testes para a avaliação do comportamento do protocolo CMB em diferentes situações.

5.1 Introdução

O capítulo 4 deste trabalho discute as possíveis formas de avaliação dos protocolos para sincronização dos processos paralelos de uma simulação distribuída. Para a avaliação de desempenho do protocolo *CMB* optou-se pelo desenvolvimento da extensão funcional *ParSMPL*.

Diferentes fatores podem exercer influência no desempenho da simulação distribuída com a utilização do protocolo *CMB*. A influência desses fatores não é facilmente identificada, uma vez que não depende apenas do protocolo, mas também da plataforma de *hardware* e *software* utilizada, do modelo a ser simulado e da partição dos processos lógicos.

Assim, o comportamento do protocolo *CMB* deve ser analisado em diferentes situações, considerando-se diversos modelos e determinando-se, para cada caso, os fatores críticos para o desempenho da simulação distribuída. Para a realização das análises foram desenvolvidas simulações utilizando diferentes modelos.

Este capítulo descreve como o ambiente *ParSMPL* será utilizado neste trabalho para avaliar o protocolo *CMB* sob os diferentes aspectos apresentados nos parágrafos anteriores. A seção 5.2 discute as metodologias e as métricas a serem utilizadas. A seção 5.3 aborda as plataformas de *hardware* e *software* consideradas nos testes. O restante do capítulo apresenta os diferentes tipos de modelos a serem considerados na avaliação do protocolo *CMB*.

5.2 Metodologia e Métricas Utilizadas

Na avaliação do comportamento do protocolo *CMB*, diversos modelos foram executados em diferentes plataformas. Para efetuar a análise dos dados obtidos, a abordagem utilizada foi a comparação dos resultados da execução da simulação dos modelos em cada uma das plataformas definidas na seção 5.5. O *benchmark Ping-Pong* foi utilizado para determinar o tempo de transmissão das mensagens, e para a coleta dos tempos de execução foi empregada a função *gettimeofday()*, obtida da biblioteca da linguagem C utilizada. Todos os testes foram executados um número

significativo de vezes com o intuito de se obter dados estatisticamente corretos. Devido ao grande número de modelos e variações a serem analisados, optou-se por trabalhar com um número pequeno de amostras (até 30 amostras), e escolheu-se o teste estatístico adequado à essas amostras. O apêndice A descreve o teste estatístico utilizado, a formulação das hipóteses e um exemplo dos testes realizados considerando o modelo do Sistema Computacional Simplificado.

Neste trabalho foram utilizadas as seguintes métricas para a avaliação do desempenho da simulação distribuída:

- **Tempo de Execução Seqüencial (*Ts*)**: tempo de execução do programa de simulação em um único elemento de processamento;
- **Tempo de Execução em Paralelo (*Tp*)**: tempo de execução do programa de simulação em um computador paralelo ou *cluster* com p elementos de processamento (abrange o tempo de execução em cada processador acrescido do tempo de comunicação entre os processadores) [Hwang; Xu 1998].
- ***Speedup* (*Sp*)**: relação entre o tempo de execução da simulação seqüencial e o tempo de execução da simulação distribuída;
- **Eficiência (*E*)**: porcentagem de elementos de processamento efetivamente utilizados durante a execução da simulação;
- **Número de Mensagens Geradas**: o número de mensagens nulas geradas durante a execução da simulação foi coletado com o intuito de verificar o comportamento do protocolo sob diversas situações e a influência dessas mensagens no desempenho da simulação. Em cada modelo foram determinados o total de mensagens geradas para cada processo lógico, distinguindo-se mensagens nulas e mensagens completas;

- **Tempo de Transmissão de Mensagens Nulas e Mensagens Completas**: determinados por meio do *benchmark Ping-Pong*.

5.3 Plataformas Utilizadas

Atualmente os sistemas distribuídos têm sido vastamente utilizados como ambientes para execução de programas paralelos, representando uma alternativa de grande versatilidade e menor custo, quando comparado às arquiteturas paralelas. Nos ambientes distribuídos fracamente acoplados (tais como arquiteturas paralelas com memória distribuída e sistemas distribuídos) um paradigma largamente utilizado no desenvolvimento de aplicações paralelas é a programação baseada em troca de mensagens.

Considerando esses fatores, foram utilizadas três plataformas para a realização dos testes: o *IBM RISC System/6000 Scalable POWERparallel System (SP2)*, um *cluster* de estações de trabalho *IBM RISC System/6000* interligado por uma rede *Token Ring* (16*Mbits/s*), e um *cluster* de estações de trabalho *IBM RISC System/6000* interligado por uma rede *Ethernet* (10*Mbits/s*).

Para controlar a comunicação entre os processos paralelos, foram utilizados dois ambientes de passagens de mensagens, o *PVM* de domínio público nos *clusters* e no *IBM SP2*, e o *PVMe* (uma versão proprietária da *IBM*) no *IBM SP2*.

As tabelas 5.1 e 5.2 apresentam respectivamente, as características do *IBM SP2* e *IBM RS/6000* (que constituem os *clusters*) utilizados na avaliação de desempenho.

Tabela 5.1 – Características: *IBM SP2.*

Plataforma	*IBM RISC System/6000 Scalable POWERparallel System*
Sistema Operacional	*IBM AIX 4.1*
Biblioteca de Passagem de Mensagens	*IBM PVMe for AIX* *PVM* domínio público versão 3.3.10
Nós	1 nó *wide* 2 nós *thin*
Thin	
Processador	*POWER 2 66,7 MHz*
RAM	*256 Mbytes*
Cache de dados	*128 Kbytes*
HD	*4.5 Gbytes*
Wide	
Processador	*POWER 2 66,7 MHz*
RAM	*256 Mbytes*
Cache de dados	*256 Kbytes*
HD	*18 Gbytes*
Comunicação	*Switch* de *Alto Desempenho*

Tabela 5.2 - Características: *IBM RISC System/6000.*

Plataforma	*IBM RISC System/6000*
Sistema Operacional	*IBM AIX 4.1*
Biblioteca de Passagem de Mensagens	*PVM* domínio público versão 3.3.10
Processador	*IBM POWER PC 66,7 MHz*
RAM	*32 Mbytes*
HD	*1 Gbyte*
Comunicação	*Ethernet 10Mbits/s* *Token Ring 16Mbits/s*

A figura 5.1 sintetiza a estrutura do ambiente *ParSMPL* e as ferramentas utilizadas. O desenvolvimento do programa de simulação independe do tipo de plataforma de *hardware* utilizada, ou seja, o mesmo código fonte é empregado em uma arquitetura paralela ou em um sistema distribuído constituído por um *cluster* de estações de trabalho. Ocorrem diferenças apenas na maneira como o código fonte é compilado e submetido para execução.

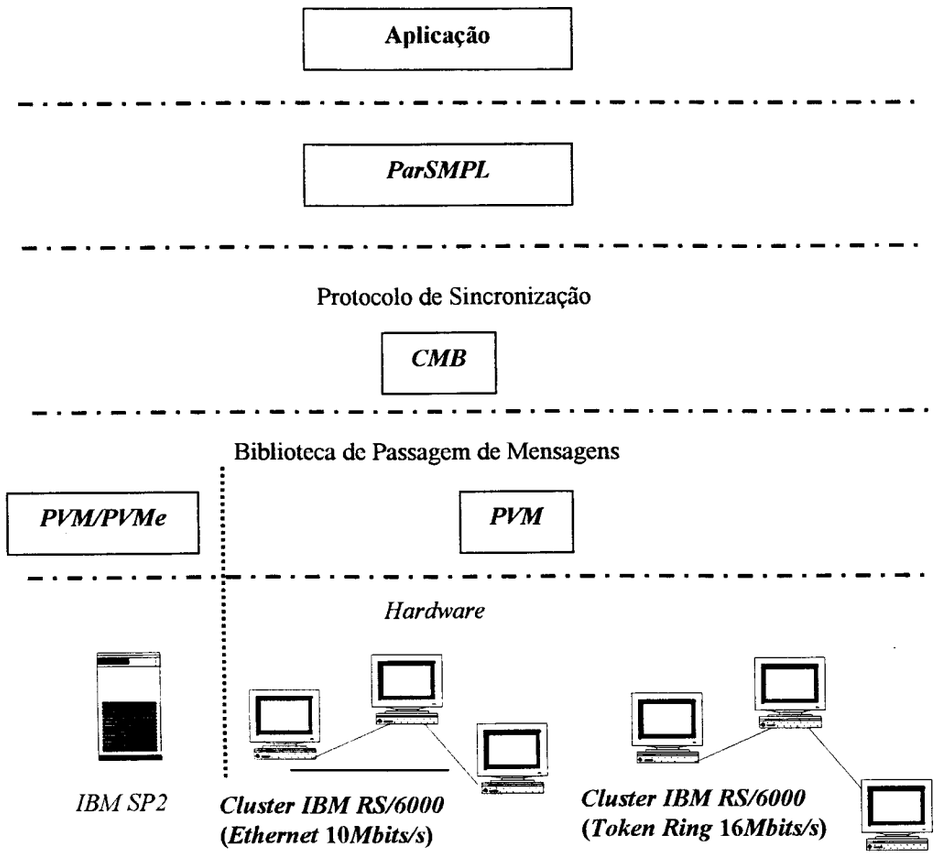

Figura 5.1 – O Ambiente de Desenvolvimento. Na plataforma *IBM SP2* foram utilizados *PVM* (domínio público) e *PVMe*. Nos *clusters Ethernet* e *Token Ring* utilizou-se o *PVM* (domínio público).

O *ParSMPL* oferece uma interface que esconde os detalhes do protocolo de sincronização utilizado e do ambiente de passagem de mensagens adotado. Desta forma, o usuário interage apenas com o *ParSMPL*, não existindo a necessidade de conhecimentos sobre as características do protocolo de sincronização e as características da arquitetura. O usuário necessita apenas de noções básicas de programação paralela. Existe, contudo, a flexibilidade para que usuários com maior experiência possam interagir com o ambiente de passagem de mensagens e com o sistema operacional da plataforma utilizada.

A forma como o protocolo *CMB* foi implementado facilita a migração do *ParSMPL* para outros ambientes de passagem de mensagens, como por exemplo o *MPI*. Além disso, o ambiente de passagem de mensagens utilizado (*PVM*) é portátil, oferecendo ao usuário a possibilidade de utilizar diferentes tipos de arquiteturas para o desenvolvimento de suas aplicações.

Definidas as métricas e plataformas a serem utilizadas, foram então desenvolvidos modelos para a avaliação do comportamento da simulação distribuída. Esses modelos são descritos na próxima seção.

5.4 Modelos Propostos

Os exemplos propostos para avaliação do comportamento do protocolo *CMB* são representados por modelos de redes de filas, devido às características do *ParSMPL*. Esses modelos apresentam um grande número de variações e características, para que se possa avaliar o comportamento do protocolo *CMB* em diferentes situações. Para cada modelo verifica-se a influência dos parâmetros considerados no tempo de simulação (tais como a variação do tempo entre chegadas de novas tarefas e número de tarefas no sistema).

Com o intuito de abranger diferentes situações, este trabalho considera dois tipos de modelos. O primeiro tipo representa sistemas reais, como servidor de arquivos, servidor central e sistemas computacionais. Para alguns desses casos são consideradas cargas reais (obtidas em sistemas em funcionamento e na literatura) e diversas variações dessas cargas. O segundo tipo de modelo engloba redes de filas representando diversas situações que podem aparecer na simulação de um sistema. Com esses modelos é possível cobrir uma grande variedade de situações, sendo

consideradas diferentes parametrizações para cada um dos modelos. A análise dos resultados obtidos permite definir os tipos de modelos e a forma de dividir o sistema a ser simulado em processos lógicos que mais se adaptam à simulação distribuída utilizando protocolos conservativos.

5.4.1 Modelo 1: Servidor Central

Esse exemplo representa um modelo de um sistema computacional constituído por um elemento de processamento e quatro unidades de disco. Tarefas requisitam a utilização do elemento de processamento e, após o término do processamento, uma das unidades de disco é selecionada [MacDougall 1987]. Esse tipo de modelo é bastante utilizado na literatura para a avaliação da simulação seqüencial e também da simulação distribuída, fator que motivou sua escolha [Ayani; Rajaei 1992, MacDougal 1987].

Para esse modelo foram analisadas duas formas distintas de divisão em processos lógicos, com o intuito de verificar a influência da configuração dos processos no desempenho da simulação, uma vez que o aumento do número de processos pode levar a um aumento na comunicação. Outro objetivo foi verificar qual a configuração mais adequada para esse tipo de modelo.

Descrição do Modelo

Nesse modelo são utilizados cinco recursos que representam *CPU* e unidades de disco. Como o modelo é fechado, um número fixo de tarefas (clientes) utiliza os recursos disponíveis no sistema.

Inicialmente, as tarefas são escalonadas para execução. Uma tarefa requisita a utilização da *CPU*, que executa o processamento quando estiver disponível. Após o término do processamento, uma das unidades de disco é determinada aleatoriamente e essa requisição é escalonada (a solicitação é inserida na fila do recurso que vai executá-la assim que esse fique disponível). Encerrado o processamento na unidade de disco, uma nova requisição para utilizar a *CPU* é escalonada e o ciclo de processamento se repete.

As figuras 5.2 e 5.3 apresentam o modelo de redes de filas para o Modelo do Servidor Central utilizando diferentes configurações dos processos lógicos[1]. Na figura 5.2 as unidades de disco são representadas em um único processo lógico. No modelo da figura 5.3, as unidades de disco são divididas em dois processos lógicos.

[1] Em todos os modelos neste trabalho, a configuração dos processos lógicos é indicada nas figuras pelas linhas tracejadas. Os processos lógicos são identificados pela sigla PL.

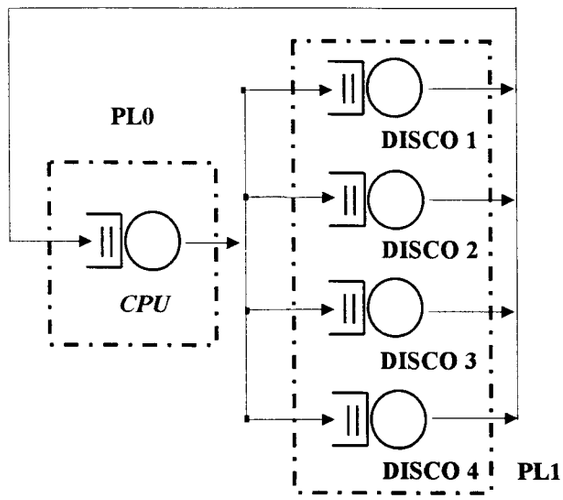

Figura 5.2 – Modelo de Redes de Filas: Servidor Central (configuração 1). Nessa configuração são utilizados dois processos lógicos, um processo contendo a unidade de processamento e outro processo contendo as unidades de disco.

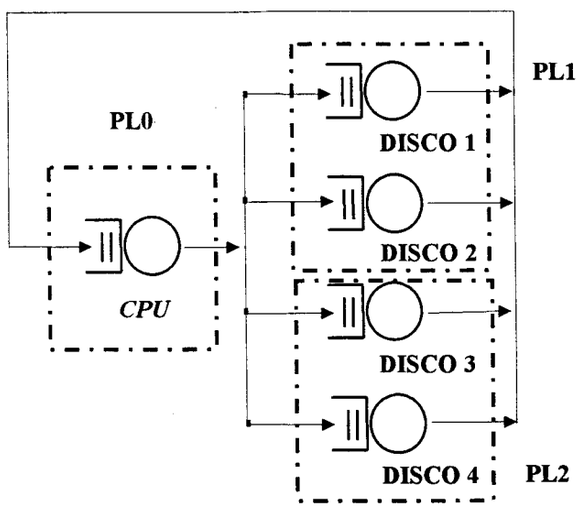

Figura 5.3 – Modelo de Redes de Filas: Servidor Central (configuração 2). Nessa configuração as unidades de disco foram divididas em processos distintos.

As configurações dos processos lógicos apresentadas nas figuras 5.2 e 5.3, além das características inerentes à configuração, diferem nos aspectos relativos a

comunicação e granulosidade dos processos lógicos. Na configuração 2, tem-se um sistema com granulosidade mais fina e maior número de processos. Por outro lado, o aumento do número de processos lógicos pode levar a um aumento da comunicação. Na configuração 1, optou-se por utilizar apenas dois processos lógicos, reduzindo a comunicação entre esses processos.

A parametrização inicial do Modelo do Servidor Central foi obtida em [MacDougall 1987]. As execuções da simulação distribuída foram realizadas inicialmente com os parâmetros originais do modelo apresentados pelo autor e, posteriormente, foram realizadas alterações no número de tarefas escalonadas inicialmente para a utilização do sistema.

5.4.2 Modelo 2: Servidor de Arquivos do Sistema Distribuído *Trice*

Um sistema distribuído pode ser definido como um conjunto de computadores independentes que parecem aos seus usuários como um único computador. Em um sistema distribuído, o sistema de arquivos é um componente chave e o serviço de arquivos especifica o que o sistema de arquivos oferece para os clientes, descrevendo as primitivas disponíveis, quais os parâmetros necessários e quais ações essas primitivas executam.

Servidores de arquivos são sistemas especiais que oferecem armazenamento de dados remoto para os usuários de um sistema computacional distribuído [Tanenbaum 1995]. O servidor de arquivos é um processo que executa em determinada máquina e implementa o serviço de arquivos. Um sistema distribuído pode ter um servidor de arquivos central ou vários servidores. O cliente não sabe quantos servidores estão disponíveis, a sua localização e qual a função de cada um.

Trice (*Transportable Integrated Computing Environment*) é um sistema computacional distribuído fracamente acoplado baseado em redes locais [Santana 1990] constituído por sub-redes conectadas a um anel central. O sistema de comunicação é organizado segundo o paradigma cliente-servidor, onde as estações de trabalho dos usuários operam como clientes, requisitando serviços específicos a um conjunto de servidores especializados [Orlandi 1995]. A informação é trocada por meio de chamadas a procedimentos remotos (*RPC - Remote Procedure Call*).

O servidor de arquivos do sistema distribuído *Trice* é fundamental dentro da estrutura desse sistema, uma vez que a maioria das estações de trabalho não têm disco local, tendo suas atividades de organização e manipulação do sistema vinculadas à operação e ao desempenho desse servidor. O comportamento do servidor de arquivos é baseado na execução das tarefas de cada um de seus componentes: processador *Front-End* (*PFE*), processador central e unidade de disco.

O *PFE* gerencia a comunicação entre o sistema de rede e o servidor. Assim, cada pacote que chega pela rede é recebido pelo *PFE*. O *PFE* opera em modo *full-duplex* com transmissão e recepção concorrente dos pacotes, sugerindo um modelo com duas unidades de *PFE*: um para gerenciamento do tráfego de entrada (*PFEin*) e outro para gerenciamento do tráfego de saída (*PFEout*). O *PFEin* opera de acordo com a disciplina *FIFO* e manipula somente um pacote por vez. Os pacotes recebidos são transferidos ao processador central que atende as requisições.

O processador central e a unidade de disco operam segundo o esquema cliente-servidor, onde o processador (cliente), requisita operações de controle ou pedidos de leitura e/ou escrita para o disco (servidor). Quando um acesso ao disco é necessário, o processador central requisita-o (colocando seu pedido na fila de espera

do disco) e inicia o processamento da próxima tarefa. Quando o disco conclui a tarefa, a resposta é liberada ao processador. Encerrado o processamento de uma tarefa, a mesma é inserida na fila do *PFEout* para deixar o sistema.

A escolha desse modelo é justificada pelo fato dos sistemas distribuídos constituírem atualmente uma opção para o desenvolvimento de aplicações paralelas. Além disso, por se tratar de um sistema real, foi possível utilizar os parâmetros do sistema na execução da simulação.

Nesse modelo foram analisadas três formas distintas de configuração dos processos lógicos, permitindo verificar o comportamento e qual situação é mais adequada ao tipo de protocolo utilizado na simulação distribuída.

Descrição do Modelo

O Modelo do Servidor de Arquivos do Sistema Distribuído *Trice* é representado por quatro recursos: os processadores *Front-End*, o processador central (*CPU*) e a unidade de disco.

A ação executada no servidor de arquivos, quando chega um novo pacote, consiste na requisição para utilização do *PFEin*. Se esse estiver ocupado, o pacote é descartado e deve ser reenviado ao servidor após um tempo específico. Existe um limite quanto ao número de tentativas para a transmissão de um pacote, uma vez que não existe fila de pedidos pendentes (o *PFEin* é representado por um recurso sem fila). Caso esteja disponível, o *PFEin* é reservado para o recebimento do pacote. Assim que o pacote for recebido, o *PFEin* é liberado. Após essa etapa, o processador

central é requisitado para processar o pedido. Se estiver ocupado, o pedido é inserido na fila de espera, caso contrário seu processamento tem início.

Durante o processamento, pode ocorrer a necessidade de operações com a unidade de disco. Nesse caso, o processador central bloqueia o processamento do pedido e o transfere para o disco, prosseguindo sua atividade com o próximo pedido. O pedido é então inserido na fila de espera do disco e assim que o disco concluir as tarefas pendentes o mesmo é processado. Concluídas essas etapas, o *PFEout* é requisitado para efetuar a transferência de dados e a resposta é inserida em uma fila de espera de respostas a serem enviadas pela rede.

As figuras 5.4, 5.5 e 5.6 apresentam diferentes formas de configuração dos processos lógicos para o modelo de redes de filas do servidor de arquivos do sistema distribuído *Trice*.

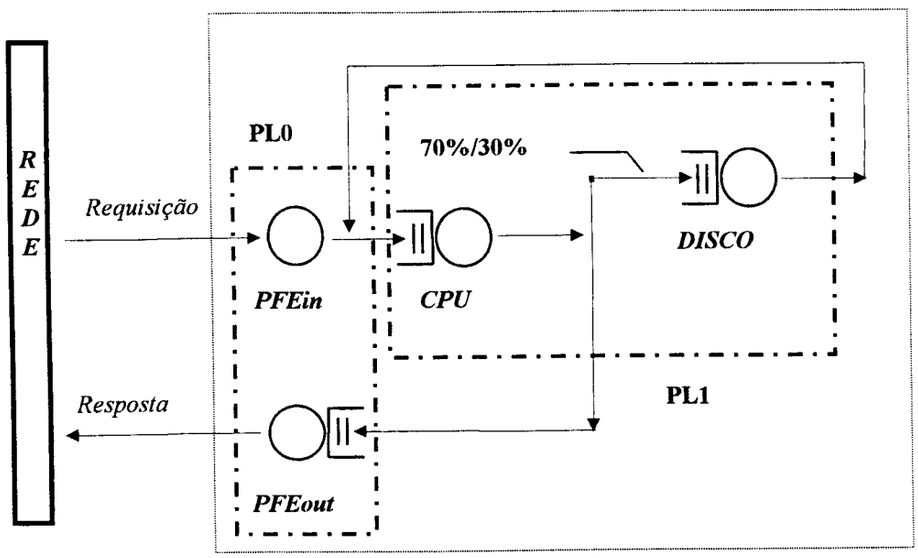

Figura 5.4 - Modelo de Redes de Filas: Servidor de Arquivos do Sistema Distribuído *Trice* (configuração 1). Nessa configuração os processadores *Front-End* são agrupados em um processo lógico, e as unidades de processamento e disco são representadas em outro processo lógico.

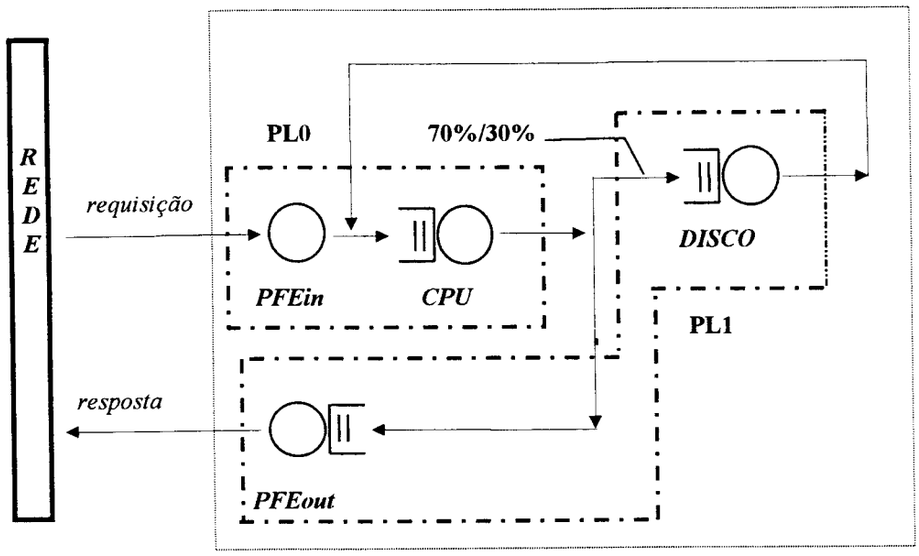

Figura 5.5 - Modelo de Redes de Filas: Servidor de Arquivos do Sistema Distribuído *Trice* (configuração 2). Essa configuração agrupa *PFEin* e *CPU* em um processo lógico e unidade de disco e *PFEout* em outro processo lógico.

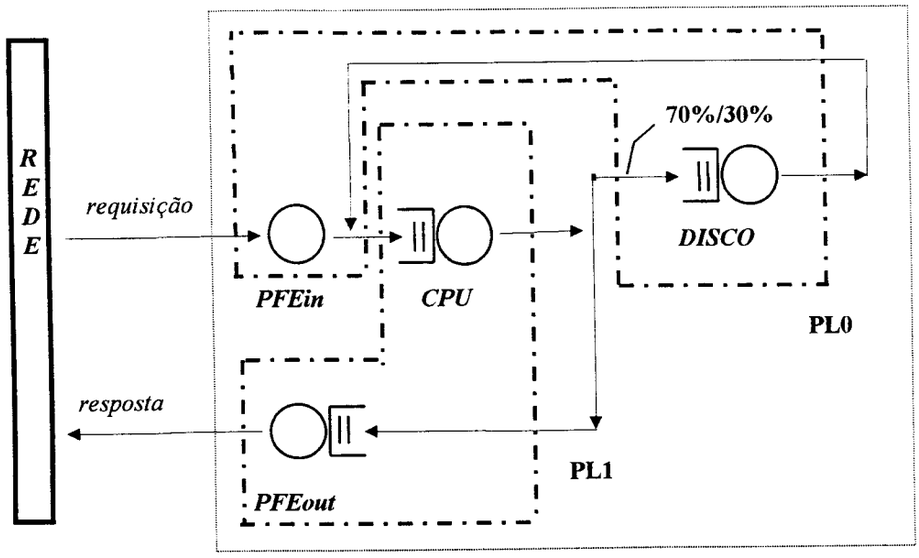

Figura 5.6 - Modelo de Redes de Filas: Servidor de Arquivos do Sistema Distribuído *Trice* (configuração 3). Essa configuração agrupa *PFEin* e unidade de disco em um processo lógico e *PFEout* e unidade de processamento em outro processo lógico.

As configurações dos processos lógicos apresentadas diferem nos aspectos relativos ao balanceamento e à comunicação. Na configuração 1 (figura 5.4) optou-se por agrupar em um processo lógico os recursos do sistema envolvidos em realimentação (*CPU* e DISCO), procurando reduzir a comunicação entre os processos. Na configuração 2 esses recursos foram associados em processos lógicos distintos, buscando distribuir de forma mais homogênea a carga de trabalho. A configuração 3 permitiu a análise do comportamento do protocolo com o aumento da comunicação.

A parametrização inicial utilizada para o Modelo do Servidor de Arquivos do Sistema Distribuído *Trice* é apresentada em [Orlandi 1995]. As execuções da simulação distribuída foram realizadas inicialmente com os parâmetros originais do modelo e posteriormente foram realizadas alterações relativas ao tempo entre chegadas de tarefas no sistema.

5.4.3 Modelo 3: Sistema Computacional Simplificado

Esse modelo apresenta uma visão macroscópica de um sistema computacional, composto por uma unidade de processamento (*CPU*) e uma unidade de disco, representando uma simplificação do Modelo do Servidor Central apresentado na seção 5.4.1 (nesse modelo a população do sistema, ou seja, o número de tarefas presentes no sistema, não é fixa como ocorre no modelo do servidor central). Por meio dos testes efetuados com o modelo foram avaliadas variações relativas aos tempos entre chegadas de tarefas e a taxa de probabilidade de uma tarefa deixar o sistema, com o intuito de verificar a influência desses parâmetros no comportamento da simulação distribuída.

Descrição do Modelo

Requisições (tarefas) são escalonadas para a utilização da *CPU* de acordo com a taxa de chegada (essas requisições são inseridas na fila da *CPU* se a mesma não estiver disponível). Quando disponível, a *CPU* retira uma requisição da lista e a executa. Após o término do processamento, a tarefa pode deixar o sistema ou solicitar os serviços da unidade de disco. Neste caso, uma requisição é escalonada para utilização da unidade de disco e após a execução a tarefa retorna para utilização da *CPU*. O modelo de redes de filas para o sistema computacional simplificado é apresentado na figura 5.7.

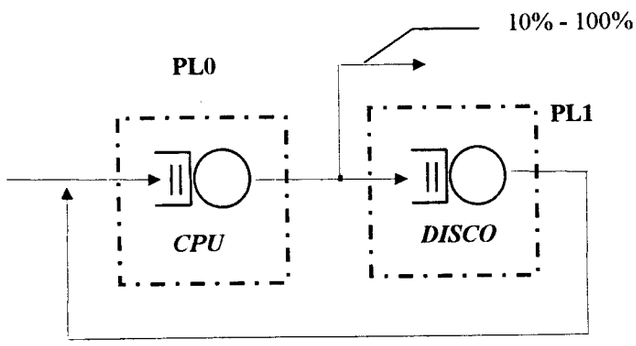

Figura 5.7 – Modelo de Redes de Filas: Sistema Computacional Simplificado. Nesse modelo as requisições (tarefas) utilizam os serviços da unidade de processamento (*CPU*). Encerrado o processamento, essas tarefas deixam o sistema ou solicitam os serviços da unidade de disco. Após deixar a unidade de disco, a requisição retorna para a fila da unidade de processamento.

Os parâmetros utilizados para a execução da simulação do Modelo do Sistema Computacional Simplificado não são representativos de um sistema real e foram definidos para a análise da simulação distribuída.

5.4.4 Modelo 4: Sistema Computacional

Esse modelo representa um sistema computacional, consistindo em uma variação do modelo descrito na seção 5.4.3, e é composto por uma unidade de processamento e duas unidades de disco. Foram analisadas três formas distintas de configuração dos processos lógicos com alteração na carga de trabalho dos processos.

Descrição do Modelo

As requisições (tarefas) são escalonadas para a utilização da *CPU* de acordo com a taxa de chegada. Essas requisições são inseridas na fila da unidade de processamento se esse recurso estiver ocupado. Quando disponível, a *CPU* retira uma requisição da fila e executa. Após o processamento, a tarefa pode deixar o sistema ou solicitar os serviços de uma das unidades de disco disponíveis de acordo com taxas de probabilidade. Neste caso, uma requisição para utilização é escalonada e após a execução a tarefa retorna para utilização da *CPU*.

As figuras 5.8, 5.9 e 5.10 apresentam o modelo de redes de filas para o Sistema Computacional com diferentes configurações dos processos lógicos e taxas de probabilidades nos pontos de decisão dos modelos (a escolha das unidades de disco é feita com igual probabilidade, ou seja, se a probabilidade da tarefa deixar o sistema é 20%, a probabilidade de solicitar a utilização de cada uma das unidades de disco é 40%).

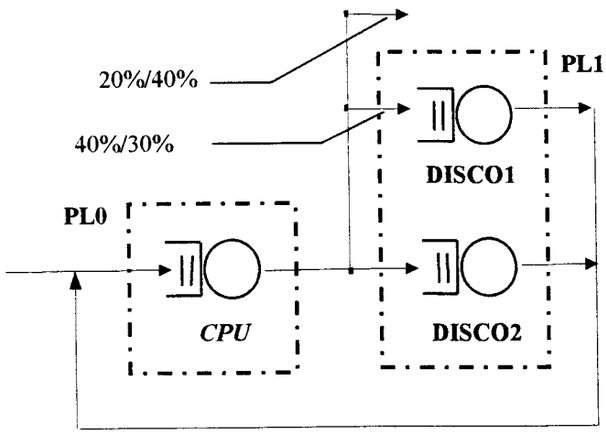

Figura 5.8 – Modelo de Redes de Filas: Sistema Computacional (configuração 1). Nessa configuração as unidades de disco estão agrupadas em um processo lógico. Foram analisadas também variações nas taxas de probabilidade nos pontos de decisão do modelo.

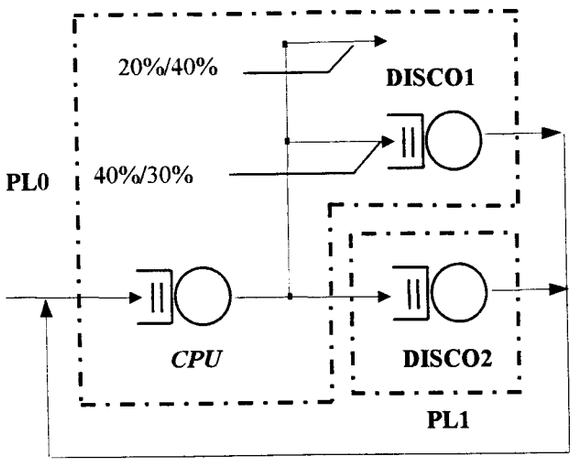

Figura 5.9 – Modelo de Redes de Filas: Sistema Computacional (configuração 2). As unidades de disco estão agrupadas em processos lógicos distintos. Foram analisadas também variações nas taxas de probabilidade nos pontos de decisão do modelo.

IFSC-USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

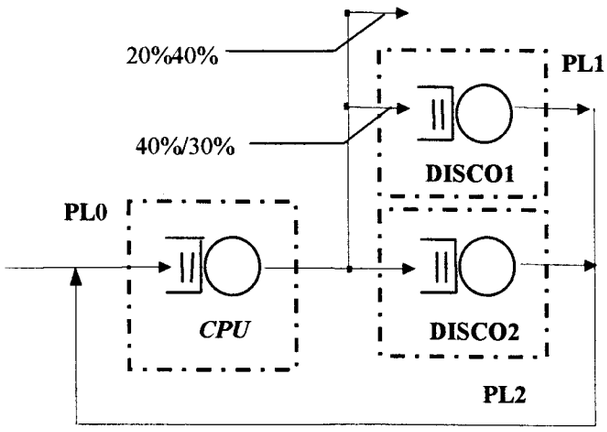

Figura 5.10 – Modelo de Redes de Filas: Sistema Computacional (configuração 3). Nessa configuração a unidade de processamento e as unidades de disco estão agrupadas em processos lógicos distintos. Foram analisadas também variações nas taxas de probabilidade nos pontos de decisão do modelo.

As configurações dos processos lógicos para o modelo em questão diferem nos aspectos relacionados ao balanceamento, comunicação e número de processos. Na configuração 1 optou-se por agrupar em um processo lógico as unidades de disco e a *CPU* em outro processo lógico. Com isso procurou-se uma melhor distribuição da carga de trabalho entre os processos lógicos.

Na configuração 2 uma das unidades de disco foi agrupada com o recurso *CPU* em um único processo lógico. Nessa configuração, parte da comunicação entre *CPU* e unidade de disco fica restrita ao processo lógico, enquanto na configuração 1, toda a comunicação entre *CPU* e unidades de disco ocorre entre os processos lógicos.

Na configuração 3 cada um dos recursos que compõe o sistema é associado em um processo lógico distinto, alterando a comunicação entre esses processos.

A parametrização inicial do modelo não é representativa de um sistema real, e foi definida para a execução do modelo no *ParSMPL*. Foram realizadas alterações

nos tempos entre chegadas de tarefas no sistema e nas probabilidades nos pontos de decisão, com o intuito de verificar a influência desses parâmetros no desempenho da simulação distribuída.

5.4.5 Modelo 5: Sistema de Computação com Processamento por Transação

Um modelo de um sistema de computação com processamento por transação apresenta uma característica particular que é a posse simultânea de recursos. Nesse sistema uma tarefa deve obter memória para iniciar o processamento e a mesma só é liberada no final do processamento da tarefa. O modelo de redes de filas para um sistema de computação com processamento por transação é representado pelos recursos Memória, *CPU* e unidades de Disco [Soares 1992].

Descrição do Modelo

Quando uma tarefa chega ao sistema, requisita a memória necessária para a sua execução. Obtida a memória necessária (a tarefa libera a memória somente depois de encerrar sua atividade no sistema), a tarefa requisita a utilização da unidade de processamento. A requisição é inserida na fila da *CPU* e, após a execução do processamento, uma das unidades de disco é escolhida. A requisição é inserida na fila do recurso se não existir servidor disponível. Liberada a unidade, a tarefa pode continuar seu processamento, escalonando nova requisição para a *CPU* ou deixando o sistema. Nesse caso a memória que estava sendo utilizada pela tarefa é liberada.

Nesse modelo os recursos do sistema foram divididos em dois processos lógicos, um processo contendo os recursos Memória e *CPU* e outro processo

contendo as unidades de disco. A figura 5.11 apresenta o modelo de redes de filas para o Sistema de Computação com Processamento por Transação [Soares 1992].

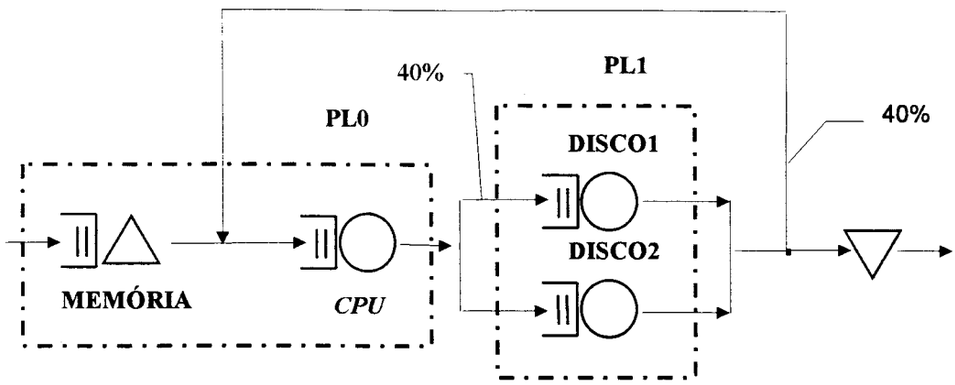

Figura 5.11 – Modelo de Redes de Filas: Sistema de Computação com Processamento por Transação. No modelo, a tarefa que chega ao sistema entra em processamento após reservar a memória, que somente é liberada após o término dos serviços no sistema.

Para a realização dos testes com esse modelo foram definidos parâmetros hipotéticos e foram analisadas variações no tempo entre chegadas das tarefas no sistema.

5.4.6 Modelo 6: Sistema de Computação com Processamento Interativo

Um sistema de computação com processamento interativo é composto pelos recursos Memória, *CPU* e unidades de disco, e o funcionamento do sistema é semelhante ao descrito para o modelo com processamento por transação (seção 5.4.5), porém nesse modelo o número de tarefas no sistema é fixo [Soares 1992]. Um número de tarefas é escalonado inicialmente para execução e essas tarefas, após encerrarem o processamento (um ciclo), retornam para um novo ciclo de execução.

Descrição do Modelo

Nesse modelo um número fixo de tarefas circula pelo sistema. Quando uma tarefa inicia o processamento no sistema, requisita a memória necessária para a sua execução. Obtida a memória (a tarefa libera a memória somente depois de encerrar sua atividade no sistema), a tarefa requisita a utilização da *CPU*. A requisição é inserida na fila da unidade de processamento se essa estiver ocupada. Encerrado o processamento, uma das unidades de disco é determinada e a requisição para utilização dessa unidade é inserida na fila do recurso.

Liberado o recurso, a tarefa pode continuar seu processamento, escalonando nova requisição para utilização da *CPU* ou liberando a memória que estava sendo utilizada. Neste caso, a tarefa volta ao sistema solicitando novo processamento. O modelo de redes de filas representando o sistema com processamento interativo é apresentado na figura 5.12 [Soares 1992].

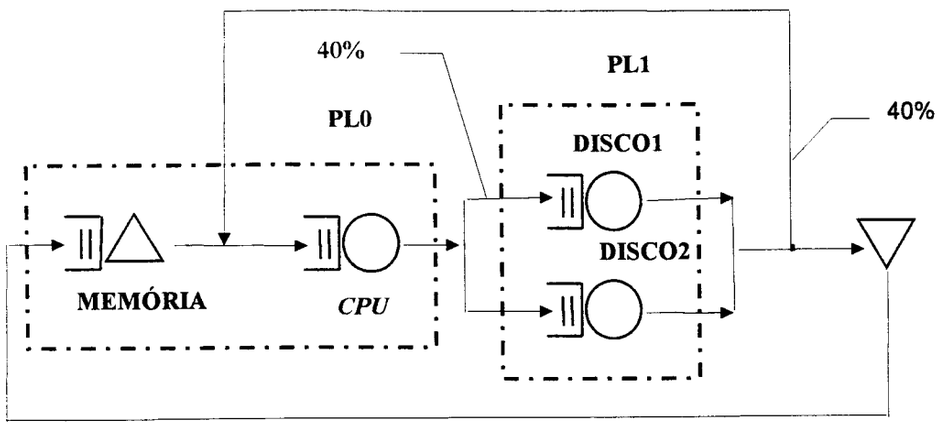

Figura 5.12 – Modelo de Redes de Filas: Sistema de Computação com Processamento Interativo. Nesse sistema, com número fixo de tarefas, uma tarefa entra em processamento após reservar a memória que somente é liberada após o término dos serviços no sistema.

Para a realização dos testes foram definidos parâmetros hipotéticos e foram analisadas variações na população (número de tarefas) inicialmente escalonada para execução no sistema.

5.4.7 Modelos Hipotéticos

Nas seções anteriores, foram analisados diversos modelos que representam diferentes sistemas computacionais. Nesta seção são apresentados alguns modelos de redes de filas que não necessariamente representam algum sistema computacional em particular.

Com a utilização dos modelos hipotéticos tentou-se abordar diferentes formas de se combinar os recursos, analisando em cada caso os fatores determinantes para o desempenho da simulação distribuída.

5.4.7.1 Modelo Hipotético 1

O modelo de redes de filas apresentado nas figuras 5.13 e 5.14 descreve um sistema hipotético constituído por sete recursos. Três desses recursos estão organizados de forma a constituir uma estrutura em série e os demais recursos estão relacionados com pontos de decisão no sistema. Foram analisadas duas formações distintas para a configuração dos processos.

Descrição do Modelo

Um número fixo de tarefas circula pelo sistema. Quando uma tarefa chega ao sistema, utiliza os serviços dos recursos identificados por REC1, REC2 e REC3. Após processamento no recurso REC3, ocorre um ponto de decisão no sistema e por meio de uma escolha probabilística, determina-se qual o próximo recurso a ser reservado para utilização. Em função desse recurso, a tarefa pode retornar para o recurso REC3 ou retornar ao início do sistema para utilizar os serviços oferecidos pelo recurso REC1.

As configurações do modelo de redes de filas para o Modelo Hipotético 1 são apresentados nas figuras 5.13 e 5.14, considerando diferentes formas para a partição dos processos.

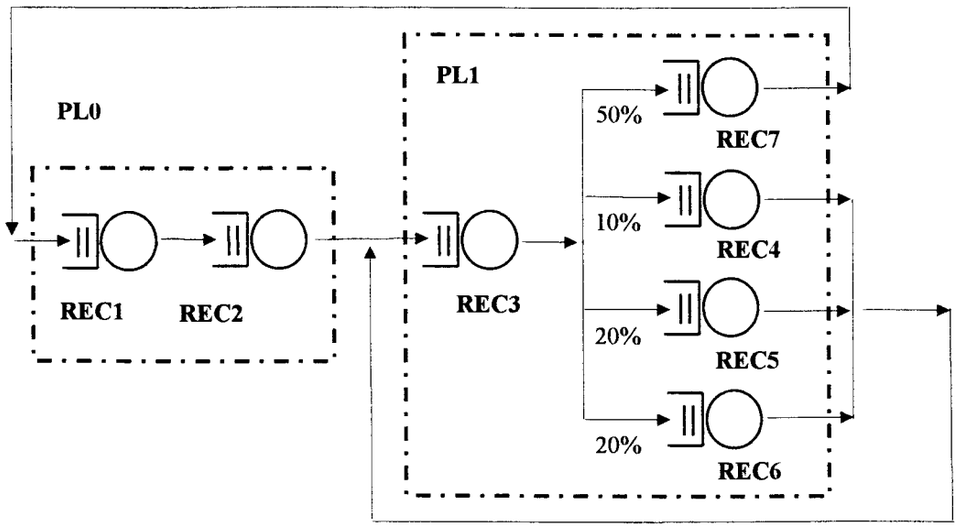

Figura 5.13 – Modelo de Redes de Filas: Modelo Hipotético 1 (configuração 1). Nessa configuração apenas dois dos recursos que constituem uma estrutura em série foram agrupados em um mesmo processo lógico; o terceiro recurso (REC3), foi agrupado juntamente com os demais recursos em um processo lógico distinto.

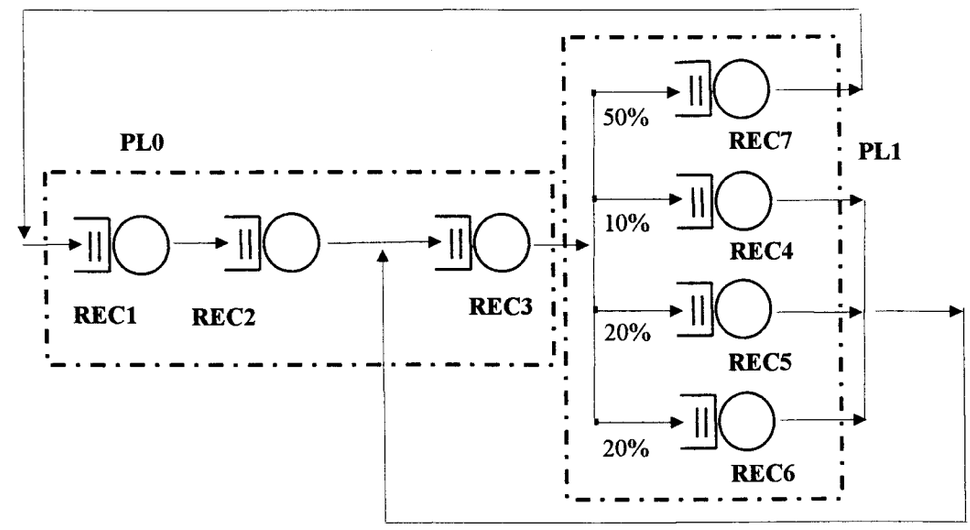

Figura 5.14 – Modelo de Redes de Filas: Modelo Hipotético 1 (configuração 2). Na configuração os três recursos que constituem uma estrutura em série são agrupados em um processo lógico e os demais recursos em outro processo lógico.

Para a realização dos testes foram definidos parâmetros hipotéticos, e foram analisadas variações na população (número de tarefas) inicialmente escalonada para execução no sistema. Foram definidas duas formas de configuração dos processos lógicos. Na configuração 1 apenas dois dos recursos que constituem uma estrutura em série foram agrupados em um mesmo processo lógico. O terceiro recurso (REC3) foi agrupado juntamente com os demais recursos em um processo lógico distinto. Na configuração 2 os três recursos que constituem uma estrutura em série são agrupados em um processo lógico, e os demais recursos em outro processo. Essas configurações foram analisadas com o intuito de verificar qual a configuração que oferece melhor desempenho para a simulação distribuída.

5.4.7.2 Modelo Hipotético 2

No Modelo Hipotético 2 o número de tarefas que circula pelo sistema não é fixo como ocorre no modelo descrito anteriormente. O sistema é constituído por sete recursos; três dos recursos constituem uma estrutura em série (REC1, REC2 e REC3) e os demais estão relacionados aos pontos de decisão no sistema.

Descrição do Modelo

Ao chegar no sistema, uma tarefa utiliza os serviços dos recursos identificados por REC1 e REC2, nessa ordem. Após o processamento no recurso REC2 ocorre um ponto de decisão no sistema e, por meio de taxas de probabilidade, a tarefa pode deixar o sistema ou prosseguir a execução. Se a tarefa continua no sistema, solicita os serviços do recurso REC3. Encerrado o processamento nesse recurso, tem-se

outro ponto de decisão para utilização dos demais recursos. A tarefa continua no sistema, retornando para utilização dos serviços de REC1 ou REC3.

O modelo de redes de filas, considerando diferentes configurações para os processos lógicos, é apresentado nas figuras 5.15 e 5.16.

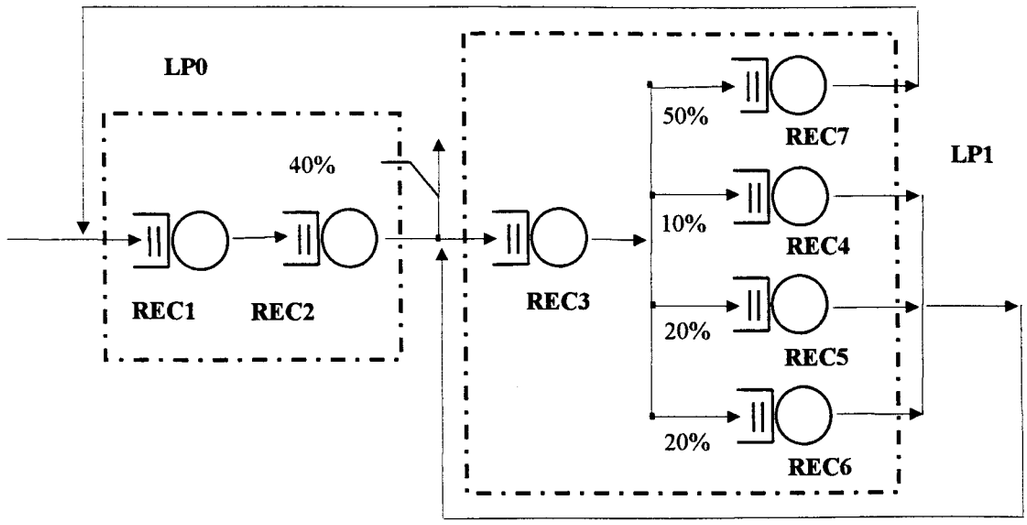

Figura 5.15 – Modelo de Redes de Filas: Modelo Hipotético 2 (configuração 1). Nessa configuração os recursos que constituem um *feedback* são agrupados em um processo lógico distinto.

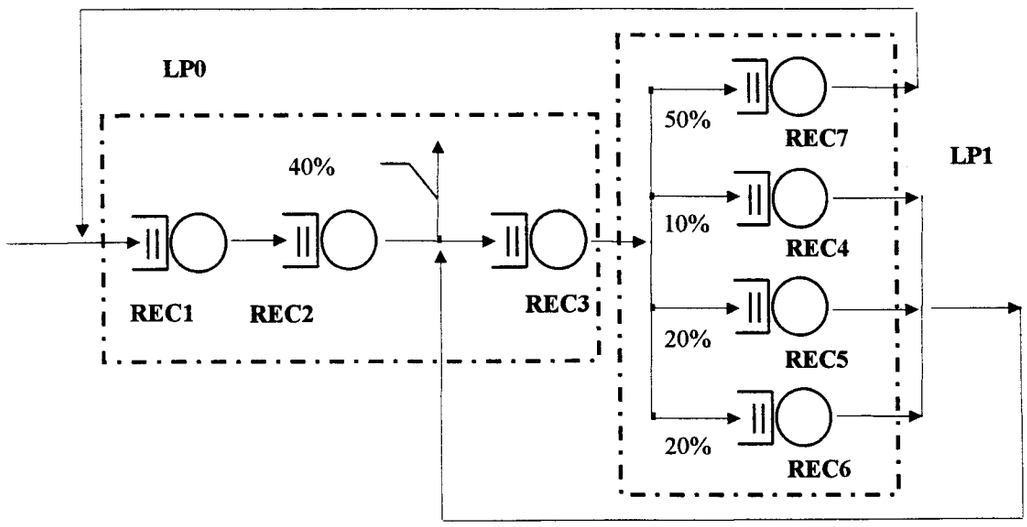

Figura 5.16 – Modelo de Redes de Filas: Modelo Hipotético 2 (configuração 2). Nessa configuração os recursos que constituem a estrutura em série são agrupados em um processo lógico distinto.

Para a realização dos testes no Modelo Hipotético 2 foram definidos parâmetros hipotéticos, e foram analisadas variações no tempo entre chegadas de clientes (tarefas) no sistema. Foram analisadas duas formas de configuração dos processos lógicos. Na configuração 1 apenas dois dos recursos que constituem uma estrutura em série foram agrupados em um mesmo processo lógico. O terceiro recurso (REC3) foi agrupado juntamente com os demais recursos em um processo lógico distinto.

Na configuração 2 os três recursos que constituem uma estrutura em série são agrupados em um processo lógico e os demais recursos em outro processo. Essas configurações foram analisadas para verificar qual configuração oferece melhor desempenho para a simulação distribuída.

5.4.7.3 Modelo Hipotético 3

O Modelo Hipotético 3 é constituído por seis recursos estruturados em seqüência. A definição desse modelo tem por objetivo analisar o comportamento da simulação distribuída em modelos com estruturas em série, semelhantes às características do modelo apresentado.

Descrição do Modelo

No modelo desenvolvido as requisições (tarefas) são escalonadas para a utilização do primeiro recurso de acordo com a taxa de chegada. Após o término do serviço, a tarefa solicita a utilização do próximo recurso. A tarefa prossegue requisitando os serviços dos demais recursos até deixar o sistema. As figura 5.17 e 5.18 apresentam o modelo de redes de filas para o exemplo, considerando diferentes formas de partição dos processos lógicos.

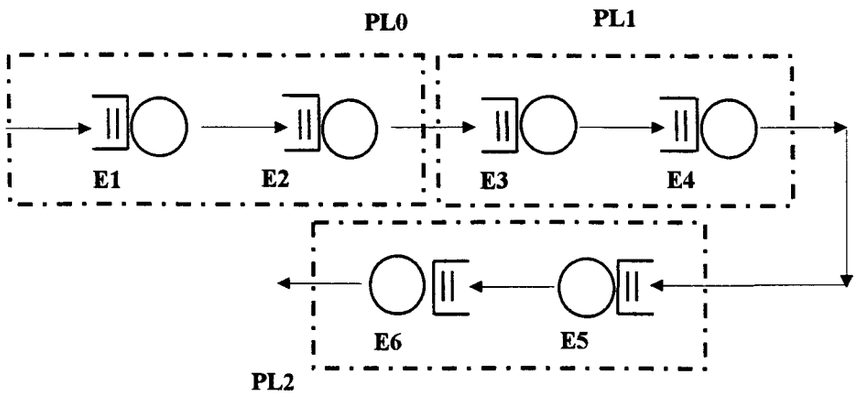

Figura 5.17 – Modelo de Redes de Filas: Modelo Hipotético 3 (configuração 1). Nessa configuração os recursos que constituem o sistema são agrupados em três processos.

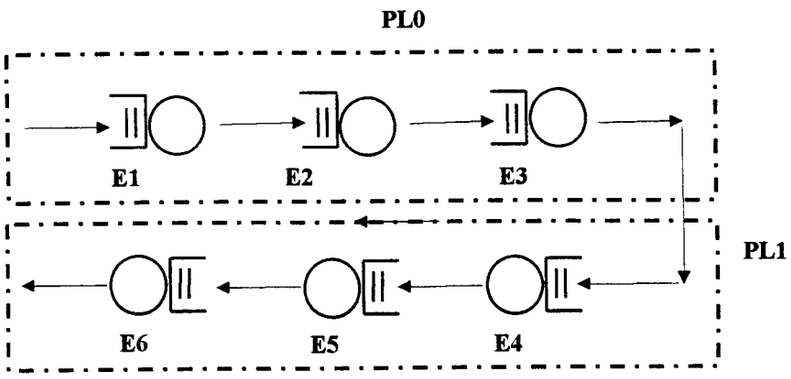

Figura 5.18 – Modelo de Redes de Filas: Modelo Hipotético 3 (configuração 2). Nessa configuração os recursos do sistema são agrupados em dois processos lógicos.

Para a realização dos testes foram definidos parâmetros hipotéticos e foram analisadas variações nos tempos entre chegadas das tarefas ao sistema. As diferentes configurações foram analisadas para verificar o comportamento da simulação distribuída nesse tipo de modelo.

5.4.7.4 Modelo Hipotético 4

O Modelo Hipotético 4 é constituído por três recursos, apresentando um ponto de decisão. Dois dos recursos do sistema estão relacionados com um *feedback*. A população do sistema não é fixa, e foram analisadas duas configurações para o modelo, considerando também alterações das probabilidades nos pontos de decisão.

Descrição do Modelo

Quando uma tarefa chega ao sistema, utiliza os serviços dos recursos identificados por REC1 e REC2. Após o processamento no recurso REC2, ocorre um ponto de decisão no sistema e, por meio de taxas de probabilidade, a tarefa pode

retornar para novo processamento no recurso REC1 ou prosseguir a execução. Se a tarefa prossegue a execução, solicita os serviços do recurso REC3 e após o término do processamento a tarefa deixa o sistema.

O modelo de redes de filas considerando diferentes configurações para os processos lógicos é apresentado nas figuras 5.19 e 5.20.

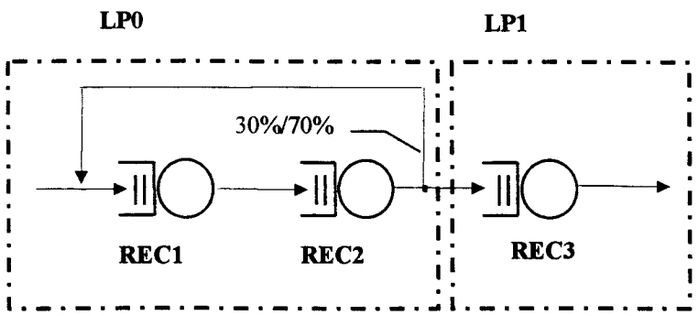

Figura 5.19 – Modelo de Redes de Filas: Modelo Hipotético 4 (configuração 1). Nessa configuração os recursos envolvidos em um *feedback* são agrupados em um mesmo processo lógico.

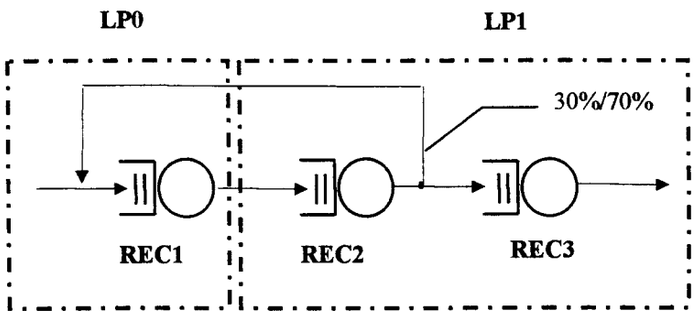

Figura 5.20 – Modelo de Redes de Filas: Modelo Hipotético 4 (configuração 2). Nessa configuração optou-se por agrupar em processos distintos os recursos que constituem o *feedback*.

Na configuração 1 os recursos envolvidos no *feedback* são agrupados no mesmo processo lógico, buscando reduzir a comunicação entre os processos que

compõem a simulação. Na configuração 2 o recurso REC1 é agrupado em um processo lógico, e os demais recursos em outro processo. As configurações foram analisadas para verificar a forma mais adequada para a configuração dos processos considerando o *feedback* presente no sistema.

Para a realização dos testes no modelo foram definidos parâmetros hipotéticos e analisadas variações no tempo entre chegadas de clientes (tarefas) no sistema e no ponto de decisão do sistema.

5.4.7.5 Modelo Hipotético 5

O Modelo Hipotético 5 é constituído por quatro recursos formando uma estrutura em série, um ponto de decisão e um *feedback* relacionando dois desses recursos. A população do sistema não é fixa, e foram analisadas três configurações distintas para os processos lógicos que constituem o modelo.

Descrição do Modelo

Quando uma tarefa chega ao sistema, utiliza os serviços dos recursos identificados por REC1, REC2 e REC3. Encerrado o processamento no recurso REC3, ocorre um ponto de decisão no sistema e por meio de taxa de probabilidade a tarefa pode retornar para novo processamento no recurso REC2 ou prosseguir a execução. Se a tarefa continua a execução, solicita os serviços do recurso REC4. Encerrado o processamento, a tarefa deixa o sistema.

O modelo de redes de filas considerando diferentes configurações dos processos lógicos é apresentado nas figuras 5.21, 5.22 e 5.23.

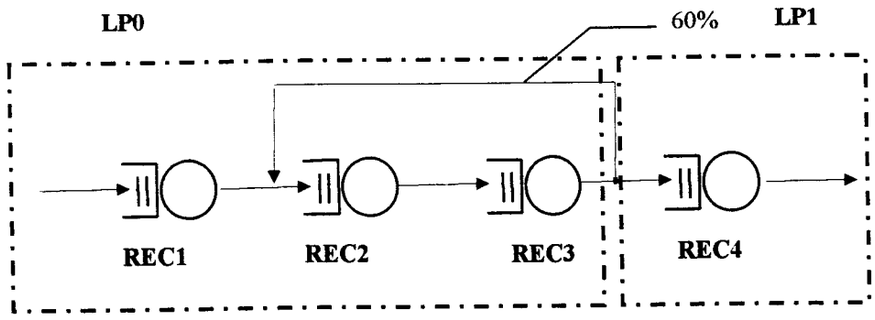

Figura 5.21 – Modelo de Redes de Filas: Modelo Hipotético 5 (configuração 1). Nessa configuração optou-se por agrupar no mesmo processo os recursos que constituem o *feedback*.

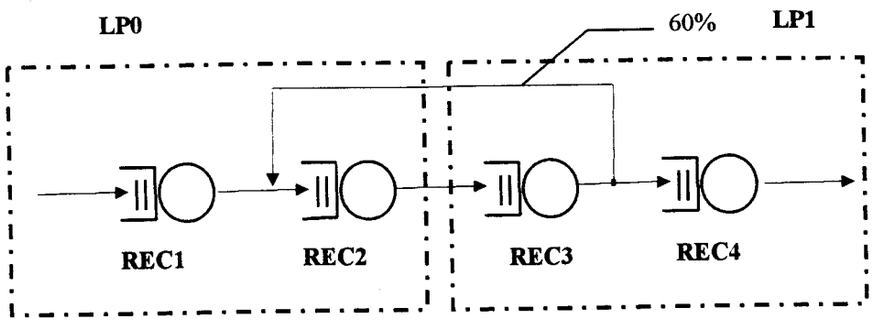

Figura 5.22 – Modelo de Redes de Filas: Modelo Hipotético 5 (configuração 2). Nessa configuração optou-se por agrupar em processos distintos os recursos que constituem o *feedback*.

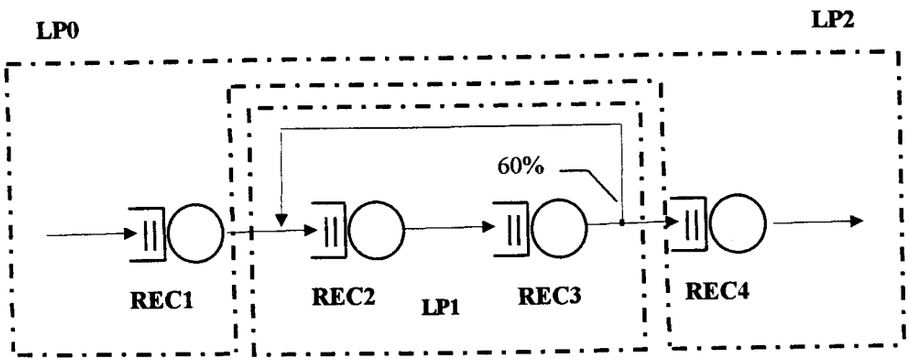

Figura 5.23 – Modelo de Redes de Filas: Modelo Hipotético 5 (configuração 3). Nessa configuração os recursos que constituem o *feedback* foram isolados em um mesmo processo.

As diferentes configurações dos processos foram analisadas para verificar o impacto da comunicação no desempenho da simulação. As configurações 1 e 3 permitem a redução da comunicação entre os processos, enquanto na configuração 2 a separação dos recursos envolvidos no *feedback* tende a provocar um aumento na comunicação entre os processos na simulação distribuída.

Para a realização dos testes no modelo foram definidos parâmetros hipotéticos e analisadas variações no tempo entre chegadas de tarefas no sistema.

5.5. Considerações Finais

O desempenho da simulação distribuída depende de diversos fatores, tais como as características do modelo simulado e da plataforma de *hardware/software* utilizada. Neste trabalho, o protocolo *CMB* é analisado considerando diferentes tipos de modelos, com o objetivo de verificar o comportamento do protocolo e o grau de influência que esses diferentes fatores podem exercer no desempenho da simulação distribuída. Para realizar as análises e determinar a influência desses fatores foram desenvolvidos diversos modelos.

Os modelos abrangem diferentes tipos de sistemas computacionais, inclusive sistemas reais que disponibilizam os parâmetros para a realização das análises, e modelos hipotéticos, criados para estudar diferentes características que podem ocorrer na prática, como pontos de tomada de decisão, recursos estruturados em série e *feedback*. Para cada modelo estudado foram consideradas diferentes situações, como a forma de divisão dos processos lógicos, os parâmetros da simulação e a granulosidade dos processos.

Deve-se ressaltar a importância do conjunto de modelos desenvolvido para a avaliação do desempenho de uma simulação distribuída conservativa. Esse conjunto pode ser utilizado como caso de testes em diversas situações, possibilitando a avaliação do protocolo *CMB* em outras plataformas e até mesmo a avaliação de outros tipos de protocolos de sincronização em simulação distribuída.

Além dos modelos de simulação, este capítulo descreve as plataformas utilizadas para a análise de desempenho do protocolo utilizando o *ParSMPL*, as métricas e os *benchmarks* aplicados na tarefa de avaliação do protocolo. No capítulo 6 são apresentados os resultados obtidos com a execução dos modelos e as análises e considerações sobre esses resultados.

A tabela 5.3 apresenta a relação dos modelos apresentados neste capítulo, identificando também a figura e a página onde foram definidos, com o intuito de facilitar a localização dos detalhes de cada modelo. Nos próximos capítulos, esses modelos são mencionados por meio dos nomes definidos nessa tabela.

Tabela 5.3 – Identificação dos Modelos.

Modelo	Figura	Página
Servidor Central (1)	5.2	136
Servidor Central (2)	5.3	136
***Trice* (1)**	5.4	140
***Trice* (2)**	5.5	141
***Trice* (3)**	5.6	141
Sistema Computacional Simplificado	5.7	143
Sistema Computacional (1)	5.8	145
Sistema Computacional (2)	5.9	145
Sistema Computacional (3)	5.10	146
Sistema de Processamento por Transação	5.11	148
Sistema de Processamento Interativo	5.12	150
Modelo Hipotético 1 (1)	5.13	152
Modelo Hipotético 1 (2)	5.14	152
Modelo Hipotético 2 (1)	5.15	154
Modelo Hipotético 2 (2)	5.16	155
Modelo Hipotético 3 (1)	5.17	156
Modelo Hipotético 3 (2)	5.18	157
Modelo Hipotético 4 (1)	5.19	158
Modelo Hipotético 4 (2)	5.20	158
Modelo Hipotético 5 (1)	5.21	160
Modelo Hipotético 5 (2)	5.22	160
Modelo Hipotético 5 (3)	5.23	160

CAPÍTULO 6

INFLUÊNCIA DAS CARACTERÍSTICAS DA PLATAFORMA NO DESEMPENHO DA SIMULAÇÃO DISTRIBUÍDA

Este capítulo apresenta uma análise sobre o speedup obtido com a simulação distribuída de alguns modelos apresentados no capítulo 5, com enfoque sobre a influência das características de hardware/software da plataforma utilizada.

6.1 Introdução

Conforme discutido no capítulo 1, o enfoque central deste trabalho é a análise do comportamento do protocolo conservativo *CMB* utilizando ambientes de passagem de mensagens, e a verificação de quais aspectos e como esses aspectos podem influenciar no desempenho de uma simulação distribuída.

Três elementos principais exercem influência no desempenho de uma simulação distribuída utilizando o protocolo *CMB*:

1. a capacidade de *lookahead* do modelo;
2. a necessidade do programador conhecer os detalhes do protocolo de

sincronização e da aplicação para obter um bom desempenho;

3. a sobrecarga inerente do protocolo e a impossibilidade de se explorar integralmente o paralelismo da simulação devido às características desse protocolo.

Para viabilizar a implementação dos protocolos conservativos são utilizadas técnicas que permitem prever o comportamento futuro de uma simulação. Essas técnicas são baseadas na capacidade de *lookahead*, que representa uma característica particular de cada modelo. Diferentes estudos analisam o impacto do *lookahead* no desempenho da simulação distribuída conservativa (referências para estudos relativos a esse tópico são apresentadas no capítulo 3).

A necessidade do usuário ter experiência em simulação distribuída e conhecimento da aplicação para a obtenção de bom desempenho está relacionada à possibilidade de maximizar o balanceamento de carga do programa paralelo e de minimizar a comunicação necessária (incluindo-se nessa comunicação tanto as mensagens completas quanto as mensagens nulas). O capítulo 7 analisa o comportamento da simulação distribuída considerando diferentes formas de partição dos processos lógicos e características dos modelos, estabelecendo diretrizes para o desenvolvimento de uma simulação distribuída conservativa.

Este capítulo analisa a influência da comunicação entre processos e do balanceamento de uma simulação distribuída considerando diferentes plataformas e tipos de aplicação distintas. Essa influência não se restringe à simulação distribuída. Quando são considerados programas paralelos de um modo geral, esses dois fatores também são críticos. Em muitos casos, a obtenção de um bom balanceamento de carga é mais significativa em termos de desempenho do que a redução da

comunicação, principalmente quando são consideradas arquiteturas paralelas que possuem sistema de comunicação de alto desempenho [Geist 1996].

No paradigma de simulação distribuída (*CMB*) o problema da comunicação entre os processos paralelos se agrava pois, além da comunicação para troca de dados, tem-se a necessidade das mensagens nulas, cuja finalidade é evitar que a simulação entre em *deadlock*. Essas mensagens são geradas apenas com o propósito de sincronização e não representam atividades no sistema físico, consistindo então em uma sobrecarga adicional que pode comprometer o desempenho da simulação.

Alguns trabalhos sobre simulação distribuída encontrados na literatura apontam a necessidade das mensagens nulas como o principal obstáculo para o desempenho com o protocolo *CMB* [Ayani 1993, Vries 1990]. Outros autores citam a impossibilidade de se explorar integralmente o paralelismo existente na simulação e a dificuldade de se alcançar um bom balanceamento de carga como o fator determinante do desempenho para esse tipo de simulação [Bagrodia et al. 1998, Fujimoto 1990].

O objetivo do estudo apresentado neste capítulo é mostrar quando cada uma das duas hipóteses comentadas no parágrafo anterior está correta, considerando diferentes tipos de plataformas computacionais e modelos. Ou seja, este capítulo analisa os resultados obtidos com a execução de diversos modelos na extensão *ParSMPL* considerando diferentes plataformas de *hardware* e *software*, comparando número de mensagens nulas, número de mensagens completas, balanceamento e *speedup* atingido.

Por meio dos resultados obtidos observa-se que o *speedup* da simulação distribuída depende, na realidade, de dois fatores: o tempo gasto com a comunicação

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

entre os processos paralelos e a potência computacional dos processadores utilizados. De forma geral, quando elementos de processamento de alta potência computacional são utilizados, necessita-se de uma forma de comunicação muito eficiente para que seja possível obter-se *speedup*. Por outro lado, quando os elementos de processamento adotados são mais lentos, mesmo com sistemas de comunicação pouco eficientes, pode-se obter bons resultados para o *speedup*.

Neste capítulo são analisados os resultados obtidos nas quatro plataformas (*hardware* e *software*), descritas no capítulo 5. Para cada uma dessas plataformas o *speedup* é estudado, considerando os tempos envolvidos com a comunicação e com o processamento.

Um conjunto significativo das simulações executadas apresentou um *speedup* com valor inferior a 1,0. Esse comportamento mostra que a sobrecarga gerada pelo sincronismo da simulação distribuída, em determinadas plataformas, dificulta a utilização do paralelismo. Mesmo para os casos em que o *speedup* é inferior a 1,0, os resultados são importantes para o estudo apresentado, pois auxiliam a caracterização do *speedup* em função da plataforma (eficiência do sistema de comunicação e potência dos elementos de processamento).

Para efetuar a análise descrita no parágrafo anterior é necessária a obtenção de dados confiáveis. A metodologia empregada para a obtenção desses dados é descrita na seção 6.2. A seção 6.3 apresenta um estudo sobre o tempo de comunicação entre processos paralelos e a potência computacional nas diferentes plataformas consideradas, a fim de possibilitar melhor análise das mesmas e dos resultados obtidos nos programas de simulação distribuída desenvolvidos. A seção 6.4 apresenta os modelos utilizados para as análises neste capítulo e os dados obtidos com a execução

da simulação distribuída desses modelos. A seção 6.5 apresenta uma análise do desempenho obtido para a simulação distribuída nas diferentes plataformas consideradas. A seção 6.6 sintetiza os resultados alcançados e apresenta um estudo para a previsão da tendência do comportamento da simulação distribuída. A seção 6.7 conclui o capítulo.

6.2 Metodologia

Na avaliação do comportamento do protocolo *CMB* foram executados diversos modelos de simulação considerando diferentes tipos de sistemas (os modelos representativos para esses sistemas são descritos no capítulo 5). As análises foram realizadas por meio da comparação dos resultados obtidos com a simulação distribuída e a correspondente implementação seqüencial. Desta forma, para os modelos avaliados, foram desenvolvidas versões seqüenciais do programa de simulação (a versão seqüencial não utiliza protocolos de sincronização e ambientes de passagem de mensagens) e foram utilizadas as métricas descritas no capítulo 5 para a realização das análises.

Para garantir que os dados obtidos estivessem estatisticamente corretos, todos os testes foram executados várias vezes com o objetivo de aumentar o tamanho da amostra e garantir a precisão dos dados. Para cada conjunto de dados obtido foram determinados média aritmética, desvio padrão, variância e outros valores relacionados à amostra, além dos testes estatísticos necessários (o Apêndice A apresenta esses testes para os dados obtidos com a execução da simulação do Modelo do Sistema Computacional Simplificado – figura 5.7, descrito no capítulo 5). Considerando cada modelo avaliado, foram utilizados diversos parâmetros.

Ainda com o intuito de garantir a precisão dos resultados obtidos, as simulações foram executadas em momentos em que as plataformas não estavam atendendo a outros usuários. Isto é, procurou-se reduzir as influências externas, provocadas por processos de outros usuários competindo com os processos da simulação pelos recursos utilizados.

6.3 Avaliação e Classificação das Plataformas Utilizadas

Para alcançar o objetivo de avaliar o desempenho de simulações distribuídas em diferentes tipos de plataformas, é necessário inicialmente analisar as características das plataformas que estão sendo consideradas. Assim, esta seção apresenta uma avaliação das plataformas utilizadas em função da eficiência de comunicação (seção 6.3.1) e da eficiência de processamento (seção 6.3.2).

6.3.1 Comunicação

Os tempos de transmissão para os dois tipos de mensagens utilizadas na simulação distribuída (mensagens completas e mensagens nulas definidas no capítulo 4) foram determinados com o intuito de fornecer os requisitos necessários para analisar os resultados obtidos com essas simulações. A tabela 6.1 apresenta os tempos de transmissão obtidos para cada plataforma utilizada neste estudo. Os tempos são expressos em milisegundos.

É importante ressaltar que as plataformas consideradas foram utilizadas com o intuito de analisar o comportamento da simulação distribuída e não visando obter o melhor desempenho para a simulação.

Tabela 6.1 – Tempo de Transmissão de Mensagens.

Plataforma	Tempo Transmissão Mensagens (ms)	
	Nulas	Completas
SP2/PVMe	0,04	0,05
Token Ring	1,97	2,47
SP2/PVM	3,28	3,61
Ethernet	1,99	2,86

O menor tempo obtido na transmissão das mensagens foi para a plataforma *SP2/PVMe*. Esta situação é justificada pelas características da plataforma, que utiliza um *switch* de alto desempenho para a comunicação entre os elementos de processamento.

Toda a comunicação entre os processos paralelos nessa plataforma é realizada por meio do protocolo *MPCI* que executa sobre o *switch* e permite o uso exclusivo desse *switch* por uma determinada aplicação. O *PVMe* foi desenvolvido para explorar todas essas características e a homogeneidade dos elementos de processamento da arquitetura.

Na mesma arquitetura (*SP2*), porém considerando o *PVM* de domínio público para a comunicação entre os processos, o tempo de transmissão das mensagens sofre um aumento considerável, quando comparado com as demais plataformas. Este fato decorre do *PVM* não estar otimizado para explorar as características da arquitetura. Além disso, existe a necessidade do *PVM* executar sobre outro ambiente, o *MPI*, inserindo maior atraso na execução das primitivas de comunicação. Devido a esses fatores, a plataforma *SP2/PVM* apresentou o pior desempenho para a comunicação, dentre as plataformas analisadas.

Considerando as plataformas do tipo *cluster*, o tempo de transmissão obtido é decorrente das características do meio de comunicação utilizado, com baixa capacidade de transmissão (10*Mbits/s* em meio de comunicação utilizando *Ethernet* e 16*Mbits/s* em meio de comunicação utilizando *Token Ring*). Deve-se ressaltar que na plataforma *Ethernet* foi utilizado um compilador de domínio público, não otimizado para as características da arquitetura. A utilização desse compilador impõe um atraso na execução da aplicação.

Considerando as quatro plataformas utilizadas neste trabalho e os resultados apresentados na tabela 6.1 observa-se que:

- ***SP2/PVMe***: pode ser considerada como uma plataforma com tempo de comunicação relativamente baixo, apresentando o melhor desempenho para a comunicação quando comparada com as demais plataformas;
- ***Token Ring***: o tempo de comunicação obtido com essa plataforma é relativamente alto, quando comparado com os resultados obtidos analisando-se o *SP2/PVMe*. Esse fato decorre das características do meio de comunicação utilizado;
- ***SP2/PVM***: apesar de utilizar uma arquitetura paralela com um meio de comunicação de alto desempenho, a sobrecarga imposta pelo ambiente de passagem de mensagens faz com que essa plataforma apresente o pior desempenho relativo à comunicação, dentre as plataformas consideradas;
- ***Ethernet***: o desempenho obtido para a comunicação nessa plataforma é inferior ao desempenho obtido para a plataforma *Token Ring*, em função da capacidade do meio de comunicação utilizado.

Desta forma, as plataformas utilizadas podem ser classificadas como apresentado na figura 6.1.

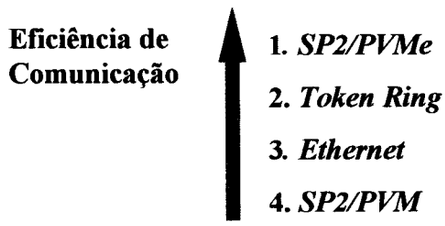

Figura 6.1 - Eficiência de Comunicação. A figura apresenta a classificação das plataformas utilizadas no desenvolvimento do trabalho, em função da eficiência de comunicação.

Como um exemplo do comportamento da comunicação em cada plataforma, pode-se considerar a execução da simulação distribuída para o Modelo *Trice*, com a configuração dos processos lógicos apresentada na figura 5.4. Nesse modelo as mensagens nulas correspondem, em média, a 50% do número total de mensagens geradas pelos processos lógicos.

A figura 6.2 apresenta o tempo de comunicação, considerando as mensagens nulas e completas, para a execução da simulação distribuída do referido modelo.

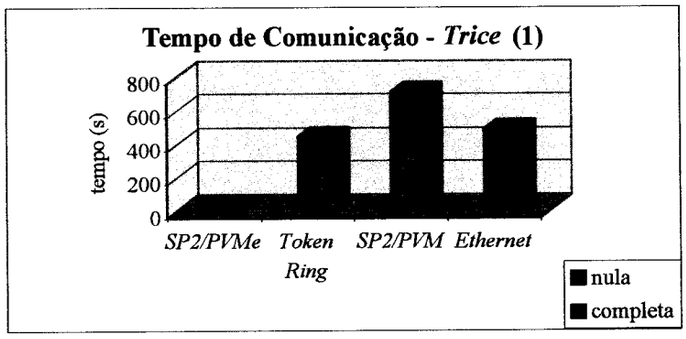

Figura 6.2 – Tempo de Comunicação – *Trice* (1). A figura apresenta o tempo de comunicação (mensagens nulas e completas) considerando o Modelo *Trice* (1) (figura 5.4).

Pode-se observar na figura o comportamento descrito anteriormente com relação a comunicação nas diferentes plataformas. Apesar do número de mensagens geradas ser semelhante, como verificado na tabela 6.2 (seção 6.4), o pior desempenho para a comunicação foi apresentado pela plataforma *SP2/PVM*. O melhor desempenho foi obtido com a plataforma *SP2/PVMe*. Nas plataformas do tipo *cluster* o pior desempenho foi observado para a plataforma *Ethernet*.

6.3.2 Elementos de Processamento

A classificação dos elementos de processamento de uma plataforma pode ser feita utilizando-se diferentes características ou ferramentas para avaliação de desempenho. A utilização de características dos elementos de processamento, tais como relógio do processador e quantidade de memória disponível, pode fornecer uma idéia de seu desempenho. No entanto, devido à complexidade dos elementos de processamento, essas características nem sempre levam a uma classificação exata.

Uma forma de comparar sistemas computacionais existentes oferecendo

resultados confiáveis é a utilização de *benckmarks*. O problema dessa abordagem é determinar qual *benchmark* melhor representa o tipo de aplicação que será executada no sistema computacional.

Como a classificação dos elementos de processamento neste trabalho tem por finalidade analisar o comportamento de simulações distribuídas optou-se por utilizar como "*benchmark*" a execução de simulações seqüenciais. Desta forma, pode-se caracterizar de forma adequada as aplicações em questão, possibilitando avaliar exclusivamente o processamento em cada plataforma.

Aplicando essa técnica, observa-se que o melhor desempenho foi obtido para os elementos de processamento da plataforma *Token Ring*, com uma pequena diferença no tempo de execução em relação à plataforma *SP2*. O pior desempenho foi obtido utilizando-se os elementos de processamento da plataforma *Ethernet*, em decorrência do tipo de compilador utilizado, que não explora as características da arquitetura. A figura 6.3 apresenta o tempo médio de execução da simulação seqüencial, considerando os modelos citados na seção 6.4.

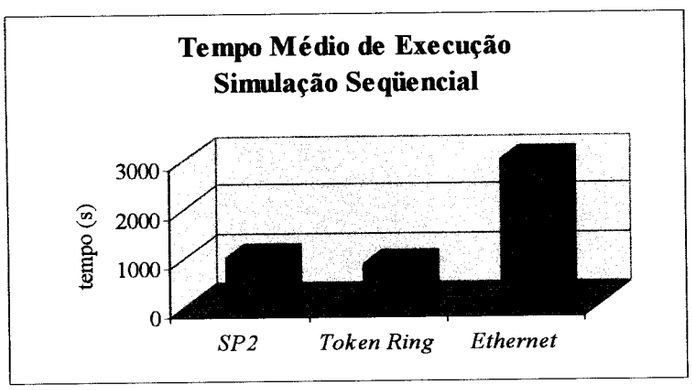

Figura 6.3 – Tempo Médio de Execução – Simulação Seqüencial. A figura apresenta o tempo médio de execução da simulação seqüencial considerando os modelos Servidor Central, *Trice*, Sistema Computacional 1 (1) e Modelo Hipotético 1 (1).

Com os resultados obtidos, pode-se classificar as arquiteturas utilizadas em função da eficiência dos elementos de processamento, conforme apresentado na figura 6.4.

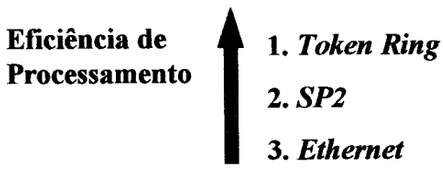

Figura 6.4 - Eficiência de Processamento. A figura apresenta a classificação das arquiteturas utilizadas em função da eficiência dos elementos de processamento.

Considerando as classificações anteriores, o desempenho obtido com a execução da simulação distribuída é analisado na seção 6.5, baseando-se nos resultados apresentados na seção 6.4.

6.4 Modelos Utilizados para a Análise de Desempenho

Esta seção apresenta alguns resultados obtidos na simulação de diversos modelos. Esses resultados servem de base para as análises apresentadas na seção 6.5 que têm por objetivo avaliar o impacto das mensagens nulas, das mensagens completas e do balanceamento no desempenho da simulação distribuída, em função da plataforma e do tipo de modelo considerado.

Essas análises foram realizadas considerando quatro dos modelos descritos no capítulo 5: Servidor Central (1) (figura 5.2), Servidor Central (2)[1] (figura 5.3), *Trice*

[1] No *PVM* (domínio público) executando no *SP2* o paradigma de programação é *SPMD*, já que o *PVM* executa sobre o *MPI*. Como no *ParSMPL* o paradigma utilizado é *MPMD*, o modelo não foi executado nessa plataforma.

(1) (figura 5.4), *Trice* (2) (figura 5.5), Sistema Computacional (1) (figura 5.8) e Modelo Hipotético 1 (1) (figura 5.13). Para o modelo *Trice (1)* utilizou-se 70% de probabilidade de uma tarefa requisitar a unidade de disco, e para o Sistema Computacional (1), 20% de probabilidade de uma tarefa deixar o sistema.

Esses modelos foram escolhidos por serem representativos da tendência do comportamento obtido para a execução da simulação distribuída, considerando o conjunto de modelos descritos no capítulo 5.

Os dados obtidos com a execução da simulação dos modelos citados são apresentados na tabela 6.2. Esses dados indicam *speedup*, tempo de execução da simulação seqüencial (TSeq), tempo de execução da simulação distribuída (TDist), tempo de transmissão de mensagens nulas (TMN), tempo de transmissão de mensagens completas (TMC), tempo total de comunicação (TTotal), número total de mensagens nulas (NN) e número total de mensagens completas (NC). Os tempos TMN e TMC foram obtidos utilizando-se os dados da tabela 6.1. Todos os tempos são denotados em segundos.

Todas essas informações são utilizadas nas seções 6.5 e 6.6 para a análise do desempenho obtido com a execução da simulação distribuída.

Tabela 6.2 - Resultados da Execução dos Modelos.

Modelo: Servidor Central (1)								
	Speedup	**TSeq**	**TDist**	**TMN**	**TMC**	**TTotal**	**NN**	**NC**
SP2/PVMe	1,34	355,5	265,9	2,8	4,9	7,7	63084	99996
Token Ring	0,47	301,5	647,6	165,5	198,8	364,3	66894	99996
SP2/PVM	0,32	355,5	1118,6	256,6	360,6	617,2	78129	99996
Ethernet	0,75	1200,7	1600,9	199,6	196,7	396,3	69892	99997
Modelo: Servidor Central (2)								
	Speedup	**TSeq**	**TDist**	**TMN**	**TMC**	**TTotal**	**NN**	**NC**
SP2/PVMe	1,46	355,5	244,3	3,9	5,0	8,9	98424	99995
Token Ring	0,43	301,5	695,8	210,9	247,0	457,9	107078	99993
Ethernet	0,68	1200,7	1765,7	205,0	286,0	490,9	102990	99998
Modelo: *Trice* (1)								
	Speedup	**Tseq**	**TDist**	**TMN**	**TMC**	**TTotal**	**NN**	**NC**
SP2/PVMe	1,12	1040,5	931,4	4,2	5,0	9,2	96082	100002
Token Ring	0,65	891,3	1372,1	243,6	247,0	490,6	98490	100000
SP2/PVM	0,44	1040,5	2386,2	349,8	361,0	710,8	106509	100000
Ethernet	1,04	2960,3	2848,5	287,8	286,0	573,8	100799	100001
Modelo: *Trice* (2)								
	Speedup	**Tseq**	**TDist**	**TMN**	**TMC**	**TTotal**	**NN**	**NC**
SP2/PVMe	1,43	1040,5	729,9	10,4	14,2	24,6	260843	284509
Token Ring	0,45	891,3	1989,4	619,4	717,0	1336,5	314440	290285
SP2/PVM	0,36	1040,5	2923,8	1034,3	772,0	1806,3	335347	283849
Ethernet	0,49	2960,3	6041,4	670,4	826,4	1496,8	336885	288945
Modelo: Sistema Computacional (1)								
	Speedup	**Tseq**	**TDist**	**TMN**	**TMC**	**TTotal**	**NN**	**NC**
SP2/PVMe	1,10	1557,6	1411,6	7,4	19,9	27,3	184136	397765
Token Ring	0,50	1355,0	2695,3	351,1	980,8	1331,9	178203	397093
SP2/PVM	0,39	1557,6	4026,0	604,3	1446,7	2051,0	184237	400749
Ethernet	0,60	4792,4	7987,3	328,8	1147,7	1476,6	165250	401310
Modelo: Modelo Hipotético 1 (1)								
	Speedup	**Tseq**	**TDist**	**TMN**	**TMC**	**TTotal**	**NN**	**NC**
SP2/PVMe	1,25	935,0	750,1	2,8	5,0	7,8	69752	100014
Token Ring	0,75	799,4	1069,1	134,4	247,0	381,4	68220	100014
SP2/PVM	0,50	935,0	1886,3	230,1	361,1	591,2	67982	100014
Ethernet	1,03	2886,0	2797,6	135,3	286,0	421,3	70163	100014

A figura 6.5 apresenta o *speedup* obtido com a execução dos modelos considerados nesta seção nas diferentes plataformas. Analisando-se o gráfico, observa-se que, para cada plataforma, esse *speedup* concentra-se em uma faixa

distinta de valores. Por exemplo, o *speedup* para os modelos na plataforma *SP2/PVMe* situa-se entre 1,0 e 1,5; para a plataforma *Ethernet* a variação ocorre no intervalo entre 0,5 e 1,0.

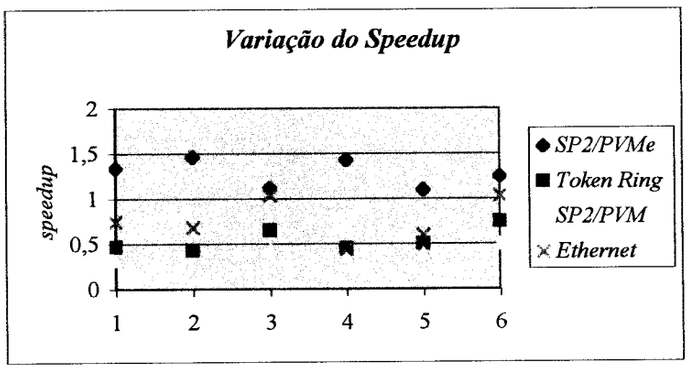

Figura 6.5 – Variação do *Speedup*. A figura apresenta o *speedup* obtido para os seguintes modelos: 1- Servidor Central (1), 2- Servidor Central (2), 3- *Trice* (1), 4- *Trice* (2), 5- Sistema Computacional 1 (1), 6- Modelo Hipotético 1 (1).

Considerando cada tipo de plataforma tem-se uma variação do *speedup* que pode ser justificada pelas características dos modelos considerados. A seção 6.6 deste capítulo relaciona as características das plataformas com o *speedup* obtido. A relação entre o *speedup* alcançado na simulação e as características do modelo não é constante para todas as plataformas. Essa característica é analisada na seção 6.5.

6.5 Avaliação de Desempenho da Simulação Distribuída

Nesta seção são analisados os desempenhos obtidos com a execução dos modelos cujos resultados foram apresentados na tabela 6.2. As análises são apresentadas enfocando individualmente cada uma das plataformas utilizadas.

6.5.1 Desempenho da Simulação Distribuída - *SP2/PVMe*

O desempenho obtido com a execução da simulação distribuída dos modelos descritos no capítulo 5, utilizando a plataforma *SP2/PVMe*, foi superior ao desempenho obtido nas demais plataformas. Tal fato é justificado pelas características do meio de comunicação e dos elementos de processamento da arquitetura, que possibilitam comunicação e processamento eficiente para as aplicações desenvolvidas, como demonstrado na seção 6.5.

A figura 6.6 apresenta o tempo de comunicação (considerando mensagens nulas e completas) e o tempo de processamento para a execução da simulação distribuída dos modelos citados na seção 6.4.

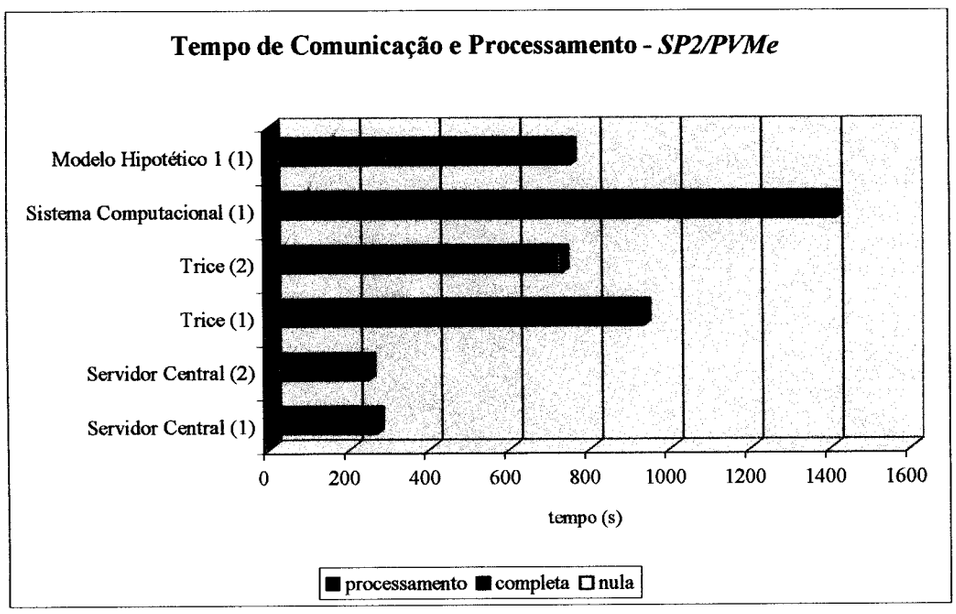

Figura 6.6 – Tempo de Comunicação e Processamento – *SP2/PVMe*. A figura apresenta o tempo e comunicação e processamento para a execução da simulação distribuída dos modelos na plataforma *SP2/PVMe*.

Nos modelos considerados, o número de mensagens nulas varia entre 30% e 55% do número total de mensagens geradas pelos processos lógicos. Mesmo considerando os modelos com um grande número de mensagens nulas, como o Modelo do Servidor Central (2) e o Modelo *Trice* (2), o desempenho obtido foi bom, possibilitando a obtenção de *speedup* com valores superiores a 1,0.

O número de mensagens não constitui uma métrica representativa do desempenho nessa plataforma. Apesar do grande número de mensagens (completas e nulas), a comunicação não representa um fator degradante para o desempenho da simulação. No Modelo *Trice* (2), por exemplo, com o total de 545352 mensagens, foi possível obter um *speedup* de 1,43. Já com o Modelo do Servidor Central (1), com 196084 mensagens, o *speedup* obtido foi 1,12.

Desta forma, a sobrecarga decorrente da comunicação entre os processos lógicos não constitui um fator degradante para o desempenho da simulação distribuída na plataforma *SP2/PVMe*. A comunicação representa, em média, 5% do tempo total de execução da simulação distribuída para os modelos considerados. Nessa plataforma os elementos determinantes para o desempenho são as características do modelo, que determinam a distribuição de carga de trabalho entre os processos lógicos, e o sincronismo imposto pelo protocolo.

Em relação aos modelos Servidor Central (1), Servidor Central (2), *Trice* (2) e Modelo Hipotético 1 (1) foi observado um bom desempenho para a simulação distribuída.

Analisando os modelos *Trice* (1) e Sistema Computacional (1), o desempenho apresentado foi modesto, quando comparado com o desempenho dos demais modelos analisados. Uma característica particular do Modelo *Trice* (1) é a presença de um

recurso sem fila, que controla o fluxo de entrada de tarefas no sistema e provoca distribuição desigual da carga de trabalho entre os processos. No Modelo Sistema Computacional (1) a probabilidade de 20% de uma tarefa deixar o sistema provoca um aumento da carga de trabalho nos processos. Para os dois modelos o sincronismo imposto pelo protocolo e as características particulares, como a forma de partição dos processos lógicos, justificam o desempenho obtido. Desta forma, o desempenho da simulação foi satisfatório em relação ao desempenho dos demais modelos, justificando também a pequena diferença do *speedup* quando comparado com o *speedup* obtido para a plataforma *Ethernet* (tabela 6.2).

Assim, pode-se concluir que o desempenho da simulação distribuída na plataforma *SP2/PVMe* depende fundamentalmente das características do modelo, que são refletidas na distribuição da carga de trabalho entre os processos lógicos e o sincronismo imposto pelo protocolo. A comunicação, nessa plataforma, não representa um fator determinante para o desempenho da simulação distribuída.

6.5.2 Desempenho da Simulação Distribuída - *Token Ring*

As características da plataforma *Token Ring*, constituída por elementos de processamento eficientes e um meio de comunicação relativamente lento, não possibilitaram a obtenção de um bom desempenho para a simulação distribuída dos modelos apresentados no capítulo 5.

A figura 6.7 apresenta o tempo de comunicação e processamento obtidos com os modelos citados na seção 6.4, considerando a execução da simulação distribuída na plataforma *Token Ring*.

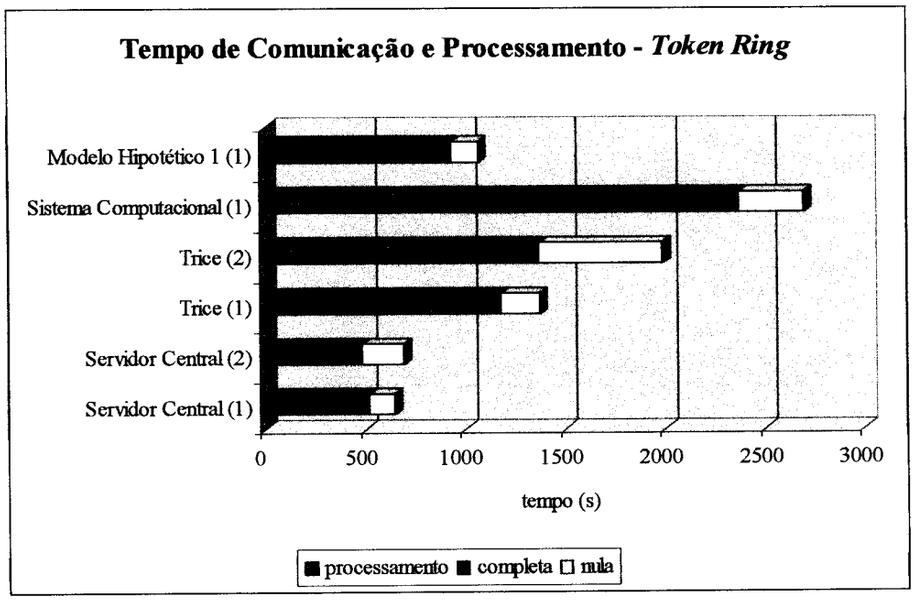

Figura 6.7 – Tempo de Comunicação e Processamento – *Token Ring*. A figura apresenta o tempo de comunicação e processamento para os modelos na plataforma *Token Ring*.

Para os modelos analisados nesta plataforma as mensagens nulas representam entre 31% a 55% do número total de mensagens geradas pelos processos lógicos. Como pode ser observado na figura 6.7 a comunicação pode exercer influência no desempenho da simulação, juntamente com as características do modelo e o sincronismo do protocolo.

O *speedup* obtido com a execução de todos os modelos analisados nessa plataforma foi inferior a 1,0. Considerando a execução da simulação distribuída, observa-se na tabela 6.2 que o tempo de execução é inferior aos tempos obtidos para as plataformas *SP2/PVM* e *Ethernet*. O baixo *speedup* é justificado pela eficiência de processamento da plataforma, que possibilitou a obtenção dos menores tempos da execução da simulação seqüencial dos modelos (tabela 6.2). Por exemplo, analisando o modelo Servidor Central (1), o tempo de execução da simulação seqüencial foi de

301,5 segundos, e na simulação distribuída, o tempo obtido foi de 647,6 segundos. Comparando com os tempos obtidos para a plataforma *SP2/PVMe*, a simulação seqüencial foi executada em 355,5 segundos e a simulação distribuída para o modelo foi executada em 265,9 segundos.

Para os modelos Servidor Central (1), Servidor Central (2), *Trice* (2) e Sistema Computacional (1) a comunicação representa um fator degradante para o desempenho. Nesses modelos a comunicação, considerando mensagens nulas e completas, corresponde em média a 61% do tempo total de execução da simulação distribuída.

Considerando os modelos *Trice* (1) e Modelo Hipotético 1 (1), além da sobrecarga da comunicação, que representa em média 35% do tempo total de execução, as características do modelo (partição dos processos e distribuição da carga de trabalho) e a sincronização imposta pelo protocolo foram responsáveis pelo desempenho pouco satisfatório da simulação.

Desta forma, pode-se concluir que os fatores determinantes para o desempenho da simulação distribuída na plataforma *Token Ring* são a comunicação e as características do modelo, que determinam a distribuição da carga de trabalho nos processos lógicos e o sincronismo imposto pelo protocolo.

6.5.3 Desempenho da Simulação Distribuída - *SP2/PVM*

A execução da simulação distribuída na plataforma *SP2/PVM* apresentou o pior desempenho dentre as quatro plataformas analisadas neste trabalho. Com a

eficiência dos elementos de processamento e o baixo desempenho da comunicação, o *speedup* obtido para o conjunto de modelos analisados foi insatisfatório.

A figura 6.8 apresenta os tempos de comunicação e processamento obtidos com a execução da simulação distribuída dos modelos citados na seção 6.4, considerando a plataforma *SP2/PVM*.

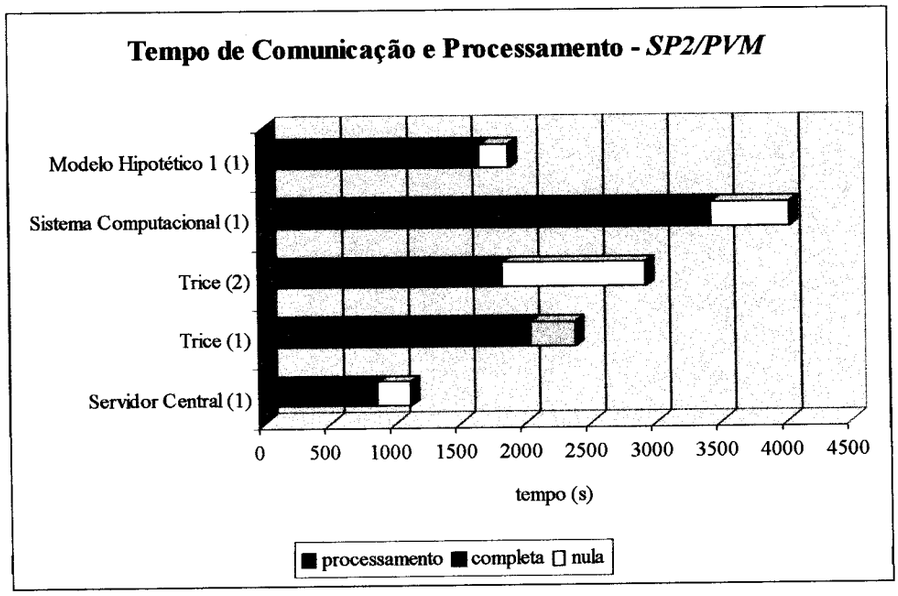

Figura 6.8 – Tempo de Comunicação e Processamento – *SP2/PVM*. A figura apresenta o tempo de comunicação e processamento para os modelos na plataforma *SP2/PVM*.

Nos modelos analisados, o número de mensagens nulas representa, em média, 44% do total de mensagens geradas na execução da simulação. Como pode ser verificado na figura 6.8, tanto a comunicação quanto as características do modelo podem degradar o desempenho da simulação distribuída. As mensagens completas representam a maior parte da comunicação nos modelos considerados.

A análise da execução da simulação distribuída dos modelos Servidor Central (1), *Trice* (2) e Sistema Computacional (1) mostra que a comunicação corresponde,

em média, a 59% do tempo total de execução da simulação. Nesses modelos a comunicação representa o fator degradante para o desempenho da simulação distribuída.

Estudando os modelos *Trice* (1) e Hipotético 1 (1), verifica-se que as características dos modelos, que determinam a distribuição da carga de trabalho nos processos lógicos e o sincronismo do protocolo, representam os fatores determinantes para o desempenho (nesse caso a comunicação representa em média 31% do tempo total de execução da simulação).

O *speedup* obtido com a execução da simulação dos modelos analisados foi inferior a 1,0 na plataforma *SP2/PVM*. Nessa plataforma o tempo de execução da simulação distribuída é superior aos tempos obtidos nas demais plataformas. A eficiência de processamento e baixa capacidade do meio de comunicação justificam o péssimo desempenho da simulação nessa plataforma. No Modelo do Sistema Computacional (1), o tempo de execução da simulação seqüencial foi de 1557,6 segundos, e para a simulação distribuída, o tempo obtido foi de 4026,0 segundos (tabela 6.2).

Desta forma, na plataforma *SP2/PVM*, a comunicação entre os processos e as características dos modelos representam os fatores determinantes para o desempenho da simulação distribuída.

6.5.4 Desempenho da Simulação Distribuída - *Ethernet*

O desempenho obtido com a simulação distribuída na plataforma *Ethernet* foi superior ao desempenho obtido nas plataformas *Token Ring* e *SP2/PVM*. Apesar do

meio de comunicação da plataforma *Ethernet* ser o menos eficiente, como demonstrado na seção 6.3, o compilador utilizado para as aplicações impõe grande atraso na execução seqüencial das aplicações na plataforma, resultando então em *speedup* superior ao *speedup* obtido no *SP2/PVM* e *Token Ring*.

A figura 6.9 apresenta o tempo de processamento e comunicação, considerando mensagens nulas e completas, obtido com a execução da simulação distribuída dos modelos citados na seção 6.4.

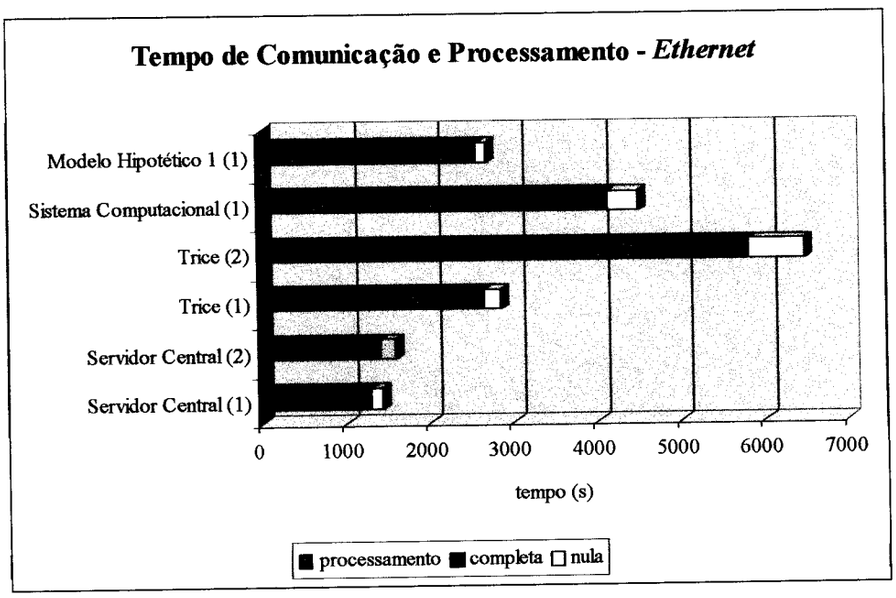

Figura 6.9 – Tempo de Comunicação e Processamento – *Ethernet*. A figura apresenta o tempo de comunicação e o tempo de processamento obtidos com a execução da simulação distribuída dos modelos na plataforma *Ethernet*.

Em todos os modelos considerados, o número de mensagens nulas representa entre 30% a 55% do total de mensagens geradas na execução da simulação distribuída e a comunicação representa, em média, 29% do tempo total de execução da simulação. Como pode ser observado na tabela 6.2, o tempo necessário para a

comunicação entre os processos na execução da simulação é semelhante ao tempo de comunicação na plataforma *Token Ring*.

Porém, o tempo de processamento (que representa a execução do trecho seqüencial da simulação) representa a maior parte do tempo de execução da simulação distribuída. A utilização de um compilador de domínio público que não explora as características da arquitetura justifica esse atraso. Mesmo com processamento pouco eficiente, é possível obter *speedup* com valor superior a 1,0 mesmo com a sobrecarga de comunicação. Como exemplo, pode-se citar os modelos *Trice* (1), com *speedup* de 1,04, e Modelo Hipotético 1 (1), com *speedup* de 1,03.

Desta forma, o desempenho da simulação distribuída nessa plataforma é afetado pelas características de processamento, pouco eficiente, e agravado pela sobrecarga de comunicação entre os processos lógicos e as características dos modelos.

6.5.5 Desempenho da Simulação – Mensagens Completas e Nulas

Como apresentado na tabela 6.2, o número de mensagens nulas e completas geradas com a execução da simulação distribuída de um determinado modelo é semelhante, quando consideradas as diferentes plataformas. A figura 6.10 apresenta o número de mensagens nulas, completas e o *speedup* de cada um dos modelos citados na seção 6.4. A identificação dos modelos é apresentada na legenda da figura.

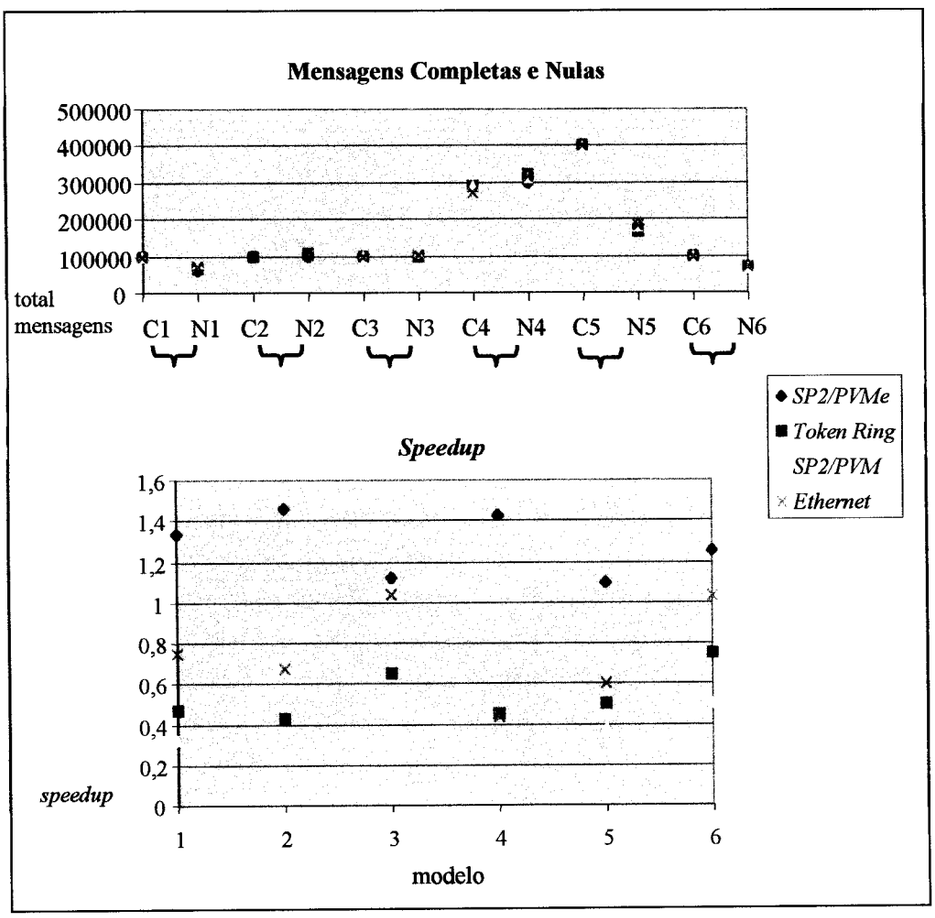

Figura 6.10 – Total de Mensagens Nulas, Completas e *Speedup*. A figura apresenta o número de mensagens nulas, completas e o *speedup* obtido para cada um dos modelos considerados. Os modelos são identificados da seguinte forma: 1- Servidor Central (1), 2- Servidor Central (2), 3- *Trice* (1), 4- *Trice* (2), 5- Sistema Computacional (1), 6- Modelo Hipotético 1 (1). No gráfico, o total de mensagens completas é indicado pela letra C acrescida da identificação do modelo. O número de mensagens nulas é indicado pela letra N acrescida da identificação do modelo. Por exemplo, C1 indica o número de mensagens completas e N1 indica o número de mensagens nulas do modelo 1 (Servidor Central (1)).

Considerando as análises apresentadas nas seções anteriores e o gráfico da figura 6.10, verifica-se que o *speedup* não está relacionado com o número de mensagens geradas. Um determinado modelo com maior número de mensagens pode

apresentar *speedup* maior quando comparado com o desempenho de outros modelos com menor número de mensagens, considerando a mesma plataforma. A situação inversa também ocorre.

Considerando por exemplo, um número médio de 200.000 mensagens pode-se observar que, para o Modelo do Servidor Central (2) foi obtido o *speedup* de 1,46. Para o Modelo Trice (1), com a mesma média de mensagens, o *speedup* obtido foi de 1,12.

Considerando um número médio de 600.000 mensagens, o *speedup* obtido para o Modelo Sistema Computacional (1) na plataforma *SP2/PVMe* foi de 1,10. Já para o Modelo *Trice* (2), o *speedup* obtido foi de 1,43.

Como verificado nas seções anteriores, o desempenho da simulação distribuída depende diretamente das características da plataforma utilizada e do modelo considerado. Na plataforma paralela o número de mensagens geradas não exerce grande influência no desempenho da simulação distribuída. Nessas plataformas as características do modelo e o sincronismo do protocolo representam os fatores determinantes para o desempenho.

Nas plataformas distribuídas a comunicação pode representar um elemento degradante para o desempenho da simulação, juntamente com as características do modelo e o sincronismo imposto pelo protocolo.

6.6 Estudo para Previsão do Comportamento da Simulação

Por meio das análises realizadas nas seções anteriores, pode-se definir quatro categorias distintas, em função da eficiência de comunicação e processamento, para a classificação das plataformas computacionais. Essas categorias são apresentadas na figura 6.11.

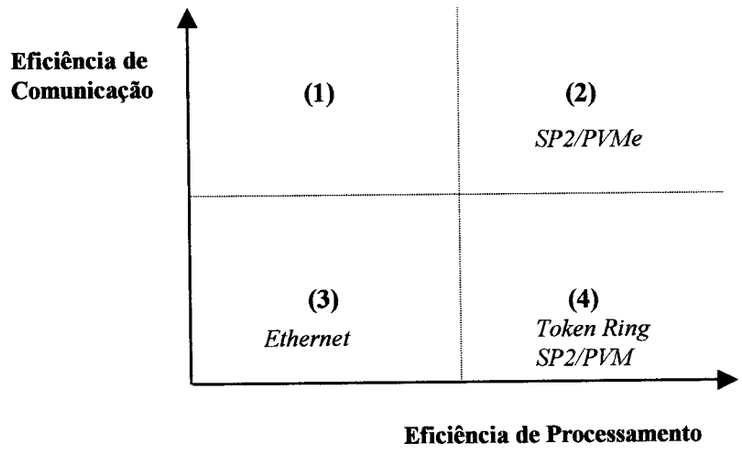

Figura 6.11 - Diagrama Comunicação x Processamento. A figura apresenta a classificação das plataformas utilizadas no trabalho em função da comunicação e processamento.

A categoria **1** representa plataformas com comunicação eficiente e processamento pouco eficiente. Neste trabalho não foram utilizadas plataformas representativas dessa categoria. A categoria **2** engloba plataformas com comunicação e processamento eficientes, como a plataforma *SP2/PVMe*. Na categoria **3** estão representadas as plataformas com comunicação e processamento ineficientes, como a plataforma *Ethernet*. As plataformas *Token Ring* e *SP2/PVM*, com processamento eficiente e comunicação pouco eficiente, são representativas da categoria **4.**

Como verificado na figura 6.5, o *speedup* obtido com a simulação distribuída dos modelos fica restrito em determinados intervalos, de acordo com a plataforma

utilizada. Na plataforma *SP2/PVMe* o *speedup* variou no intervalo de 1,0 a 1,5. Para a plataforma *Ethernet*, o *speedup* obtido apresentou variações no intervalo entre 0,5 e 1,0. Analisando a plataforma *Token Ring*, verifica-se que essa variação ocorreu no intervalo entre 0,25 e 0,75. Na plataforma *SP2/PVM* a variação do *speedup* ocorreu entre 0,0 e 0,5.

Desta forma, extrapolando a classificação definida para as plataformas, pode-se estabelecer uma abordagem para analisar o desempenho da simulação distribuída. Utilizando esse estudo é possível para o usuário prever a tendência do comportamento da simulação, em função das características da plataforma utilizada. Como as análises do desempenho da simulação foram realizadas com a utilização de técnicas de aferição, a precisão e a confiabilidade dos dados obtidos possibilitam utilizar esse estudo considerando diferentes tipos de plataformas.

A figura 6.12 apresenta a abordagem definida, considerando o desempenho da simulação distribuída nas plataformas utilizadas neste trabalho. Nessa figura a linha tracejada na diagonal indica um *speedup* para a simulação distribuída com valor igual a 1,0. As setas indicam a variação do *speedup* para diferentes tipos de modelos, em cada uma das plataformas.

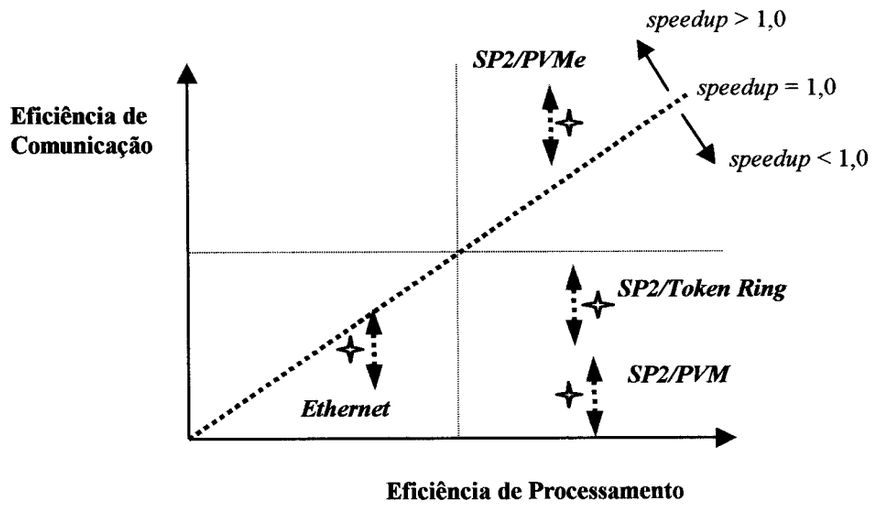

Figura 6.12 - Desempenho da Simulação Distribuída. A figura apresenta uma tendência do comportamento da simulação distribuída, em função das características da plataforma e do modelo. A linha tracejada indica *speedup* da simulação com valor igual a 1,0. As setas indicam a variação do *speedup* decorrente das características do modelo considerado.

Analisando a figura 6.12 pode-se prever a tendência do comportamento da simulação distribuída para uma determinada plataforma. Na plataforma *SP2/PVMe*, pode-se observar que a simulação distribuída tende a apresentar *speedup* superior a 1,0. Nas plataformas *SP2/PVM* e *Token Ring*, a tendência é obter um desempenho insatisfatório para a simulação distribuída, com *speedup* inferior a 0,75. Para plataformas *Ethernet* o desempenho da simulação pode ser satisfatório, com *speedup* superior a 0,5.

Outro aspecto importante a ser considerado é a variação que ocorre no *speedup*. Essa variação é dependente das características de cada modelo, e a sua determinação depende do grau de impacto que cada característica pode determinar no desempenho da simulação (esse estudo é apresentado no capítulo 7).

A abordagem da figura 6.12 pode ser expandida para diferentes tipos de plataformas. Desta forma, é possível também para um usuário sem experiência em simulação distribuída prever a tendência do comportamento da aplicação antes mesmo do desenvolvimento de sua aplicação. Com isso, esse usuário pode avaliar a relação custo/benefício para determinar a viabilidade de utilização do paradigma da simulação distribuída.

6.7 Considerações Finais

Neste capítulo foram apresentadas as análises realizadas para verificar o impacto das características da plataforma no desempenho da simulação distribuída conservativa. Com os resultados obtidos, verifica-se que o desempenho é determinado em função do tipo de plataforma utilizada e das características do modelo desenvolvido, e nem sempre as mensagens nulas representam um elemento degradante para o desempenho da simulação.

Em determinadas situações a maximização do balanceamento pode implicar no aumento da comunicação entre os processos e mesmo assim possibilitar um bom desempenho para a simulação, quando considerada a plataforma *SP2/PVMe* com meio de comunicação de alto desempenho.

Nessa plataforma a sobrecarga das mensagens completas e nulas não constitui um fator determinante para o desempenho. As características do modelo, que determinam a distribuição da carga de trabalho, e o sincronismo decorrente do protocolo são os fatores determinantes para o desempenho da aplicação.

Os estudos realizados com plataformas que utilizam meio de comunicação de baixo desempenho mostram que a comunicação entre os processos (as mensagens nulas e completas) pode degradar o desempenho da simulação distribuída. Além da comunicação, as características do modelo e o sincronismo imposto pelo protocolo podem afetar o desempenho.

Desta forma, pode-se concluir que na plataforma paralela com meio de comunicação de alta capacidade (*SP2/PVMe*), as características do modelo e o sincronismo do protocolo constituem o principal obstáculo para a obtenção de um bom desempenho da simulação distribuída conservativa. Considerando as plataformas com meio de comunicação com baixo desempenho (*Token Ring, Ethernet e SP2/PVM*), além dos fatores citados anteriormente, tem-se que a comunicação representa um fator de risco para o desempenho da simulação.

Em função das análises realizadas, foi apresentado um estudo que possibilita a previsão da tendência do comportamento da simulação distribuída em função das características arquiteturais da plataforma utilizada.

A utilização de técnicas de aferição, que permite obter dados precisos e confiáveis, possibilita realizar um estudo sobre o comportamento da simulação distribuída em função das características da plataforma. Assim, com base nesse estudo, é possível analisar a tendência do comportamento da simulação distribuída. Porém, as características inerentes do modelo a ser simulado impõem uma variação no *speedup* obtido, dificultando a determinação precisa desse *speedup*.

Desta forma, o capítulo 7 apresenta uma análise sobre a influência de diferentes características no desempenho da simulação, de forma a determinar o grau

de impacto dessas características no desempenho e estabelecer diretrizes que possibilitam a otimização da simulação distribuída conservativa.

CAPÍTULO 7

ANÁLISE DA INFLUÊNCIA DAS CARACTERÍSTICAS DO MODELO E PARTIÇÃO DOS PROCESSOS LÓGICOS NO DESEMPENHO DA SIMULAÇÃO DISTRIBUÍDA

Este capítulo apresenta uma avaliação da influência da divisão dos processos e das características de uma aplicação no desempenho de uma simulação distribuída, utilizando o protocolo conservativo CMB. A avaliação é realizada com a análise dos resultados obtidos com a execução dos modelos apresentados no capítulo anterior, utilizando a extensão ParSMPL. Como resultado dessa análise são propostas diretrizes com o intuito de auxiliar o usuário com pouca experiência em programação paralela.

7.1 Introdução

Como discutido no capítulo 6, a necessidade do usuário ter conhecimento sobre o paradigma de simulação distribuída e da aplicação a ser desenvolvida para a obtenção de bom desempenho está relacionada à possibilidade de maximizar o balanceamento de carga e reduzir a comunicação entre os processos paralelos. Desta

IFSC-USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

forma, para alcançar um bom desempenho em uma simulação distribuída, o usuário deve:

- conhecer profundamente o sistema a ser simulado e seu modelo de forma a explorar todo o paralelismo inerente ao modelo;
- analisar a melhor forma de executar a partição do modelo, com o intuito de maximizar o balanceamento e minimizar a comunicação;
- conhecer as características arquiteturais da plataforma utilizada, para que seja possível avaliar a relação entre balanceamento de carga e comunicação. Isto é, determinar quando compensa comprometer o balanceamento de carga para minimizar a comunicação e vice-versa.

Os fatores citados anteriormente são analisados neste capítulo para verificar o grau de influência de diferentes características dos modelos no desempenho da simulação distribuída considerando diferentes plataformas. Para as análise são utilizados os modelos e as plataformas descritos no capítulo 5, considerando:

- como a estrutura do modelo influencia na partição dos processos lógicos;
- como a partição dos modelos influencia no desempenho da simulação distribuída;
- o relacionamento entre a partição do modelo em diferentes plataformas e o desempenho da simulação.

Desta forma, o objetivo deste capítulo é contribuir para que a necessidade do programador conhecer os detalhes do mecanismo de sincronização seja minimizada, por meio da apresentação de sugestões sobre como efetuar a partição de um modelo de simulação para alcançar o melhor desempenho possível em uma determinada

plataforma. Os modelos descritos no capítulo 5 são considerados para a realização das análises, variando-se suas estruturas, as probabilidades nos pontos de decisão e o tipo de sistema.

As análises foram realizadas por meio da comparação dos resultados obtidos com a execução de simulações distribuídas considerando-se diferentes partições ou com a comparação dos resultados da simulação distribuída e a correspondente implementação seqüencial. A metodologia adotada foi a mesma descrita no capítulo anterior, empregando-se as medidas necessárias para garantir a precisão dos dados e reduzir as influências externas na execução da simulação, além da realização dos testes estatísticos necessários para o conjunto de dados obtidos.

Na seção 7.2 cada uma das características que pode exercer influência no desempenho da simulação distribuída é considerada individualmente. Por meio dos resultados obtidos em diversas execuções, pode-se definir a melhor forma de efetuar a partição dos processos lógicos para a obtenção de melhor desempenho para a simulação distribuída. A seção 7.3 apresenta uma análise do desempenho da simulação distribuída sem o sincronismo, ou seja, com a ocorrência de erros de causa e efeito, como uma alternativa para melhorar o desempenho da simulação. A seção 7.4 conclui o capítulo, apresentando as diretrizes estabelecidas para orientar o desenvolvimento de uma simulação distribuída conservativa.

7.2 Análise do Desempenho da Simulação Distribuída

Para atingir o objetivo deste capítulo, que é definir um conjunto de diretrizes para guiar um usuário na implementação de uma simulação distribuída, optou-se por

especificar uma série de hipóteses para verificar o grau de influência de diferentes fatores no desempenho da simulação.

Ressalta-se que neste capítulo a preocupação não está no fato de justificar o *speedup* alcançado em cada simulação, mas na comparação entre os *speedups* alcançados em simulações executadas enfocando diferentes formas de particionar um modelo e características desse modelo. O *speedup* de cada simulação foi analisado no capítulo 6, cujo enfoque abrange a influência das plataformas utilizadas no desempenho da simulação.

As hipóteses a serem analisadas englobam diferentes aspectos de uma simulação distribuída. Um primeiro aspecto a ser analisado é a influência da partição de um modelo em processos lógicos no desempenho da simulação. Nessa etapa são analisadas diferentes configurações para um número constante de processos lógicos (seção 7.2.1).

O segundo aspecto a ser analisado refere-se à variação na granulosidade da simulação. Observa-se que uma simulação distribuída sincronizada pelo protocolo *CMB* deve explorar granulosidade grossa. Até que ponto pode-se diminuir a granulosidade, com o aumento do número de processos lógicos, sem prejudicar o desempenho alcançado na execução da simulação? Qual a influência da granulosidade no tempo de execução de uma simulação? A seção 7.2.2 analisa as questões referentes à granulosidade. Finalmente, a seção 7.2.3 analisa como as diferentes características de um modelo influenciam no comportamento de uma simulação distribuída.

7.2.1 Partição do Modelo em Processos Lógicos

A forma como os processos lógicos são particionados pode levar a diferenças no balanceamento e comunicação entre esses processos, e conseqüentemente, à variações no desempenho da simulação. Considerando a utilização da simulação distribuída com o protocolo *CMB*, a partição dos processos pode aumentar o número de mensagens geradas e assim, pode exercer influência no desempenho devido ao aumento da comunicação.

As diretrizes para especificar a partição e o mapeamento dos processos de uma simulação distribuída foram estabelecidas considerando:

- a avaliação de um número elevado de exemplos;
- a experiência adquirida pela autora durante o desenvolvimento de um grande número de simulações distribuídas;
- diretrizes sugeridas na literatura para atingir bom balanceamento e minimizar a comunicação em outros tipos de algoritmos paralelos.

Com o intuito de facilitar a leitura e tornar este capítulo um guia de fácil utilização para usuários menos experientes em simulação distribuída, as diretrizes são apresentadas por intermédio de hipóteses. Cada uma das hipóteses é especificada, em seguida é exemplificada e concluída. Os resultados obtidos referem-se aos modelos descritos no capítulo 5.

7.2.1.1 Hipótese 1: Partição do Modelo em Processos Lógicos (1)

A forma de partição do modelo em processos lógicos deve reduzir a comunicação ou maximizar o balanceamento, em função das plataformas de *hardware/software* disponíveis. Em arquitetura paralela com meio de comunicação de alto desempenho e um ambiente de *software* que permita explorar as características do *hardware*, a partição dos processos deve maximizar o balanceamento. Em plataformas que utilizam meio de comunicação relativamente lento, a comunicação pode representar um obstáculo para o desempenho da simulação distribuída e portanto deve ser minimizada.

Exemplo 1: *Trice*

Para avaliar o impacto da partição dos processos lógicos no desempenho da simulação distribuída foram utilizadas duas configurações para o modelo *Trice*, ambas constituídas por dois processos lógicos e com probabilidade de 70% de uma tarefa solicitar a utilização da unidade de disco. Na configuração 1 (*Trice* (1) - figura 5.4) um processo lógico contém os processadores *Front End*, responsáveis pelo gerenciamento do tráfego, e o outro processo é constituído pelas unidades de processamento e disco. Na configuração 2 (*Trice* (2) - figura 5.5) a unidade responsável pelo gerenciamento do tráfego de entrada (*PFEin*) foi agrupada com a unidade de processamento em um processo lógico. A unidade de disco e a unidade de gerenciamento do tráfego de saída (*PFEout*) constituem outro processo lógico.

O *speedup* obtido e o número total de mensagens geradas para cada caso são apresentados na tabela 7.1. A figura 7.1 apresenta uma representação gráfica do *speedup* para as duas configurações consideradas.

Tabela 7.1 – *Speedup –Trice* (Partição dos Processos Lógicos).

Modelo \ Plataforma	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
***Trice* (1)**				
Speedup	1,12	0,65	0,44	1,04
Total Mensagens	196084	198490	206509	200800
***Trice* (2)**				
Speedup	1,43	0,45	0,36	0,49
Total Mensagens	545352	604725	619196	625830

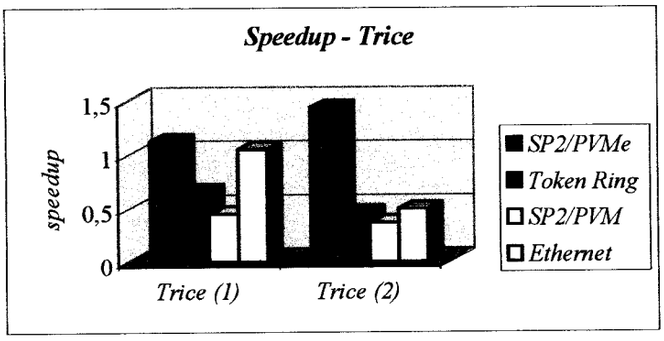

Figura 7.1 – *Speedup* - *Trice* (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido com o Modelo do Servidor de Arquivos do Sistema Distribuído *Trice*. As configurações dos processos lógicos são descritas nas figuras 5.4 (*Trice* (1)) e 5.5 (*Trice* (2)).

Considerando a plataforma *SP2/PVMe*, a configuração 2 foi mais eficiente. Nessa configuração tem-se melhor balanceamento de carga, uma vez que o servidor *PFEin* é um recurso sem fila (na configuração 1 o processo lógico 0 tem menor carga de trabalho em relação ao processo lógico 1). Na configuração 2 o impacto do aumento da comunicação não afetou o desempenho da simulação distribuída. Apesar

desse aumento de comunicação, o *speedup* obtido foi superior ao *speedup* obtido para a configuração 1.

Nas plataformas *Token Ring*, *SP2/PVM* e *Ethernet* a configuração 2 apresentou uma redução do desempenho, fato justificado pelo aumento do número de mensagens completas e nulas entre os processos [Ulson et al. 1999].

Exemplo 2: Sistema Computacional

Para o Modelo do Sistema Computacional foram utilizadas as configurações apresentadas nas figuras 5.8 e 5.9. Na configuração 1 (Sistema Computacional (1) - figura 5.8) a unidade de processamento foi isolada em um processo lógico, e as unidades de disco foram agrupadas em outro processo lógico. Para a configuração 2 (Sistema Computacional (2) - figura 5.9) a unidade de processamento e uma das unidades de disco foram colocadas em um processo lógico, e a outra unidade de disco foi representada em um processo lógico distinto. Em ambas configurações utilizou-se 40% de probabilidade de uma tarefa deixar o sistema.

O *speedup* obtido para o modelo e o número total de mensagens é apresentado na tabela 7.2. A figura 7.2 apresenta graficamente o *speedup* obtido para configuração do modelo.

Tabela 7.2 – *Speedup* – Sistema Computacional (Partição dos Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Sistema Computacional (1)				
Speedup	1,26	0,40	0,34	0,43
Total Mensagens	316023	335820	343315	341638
Sistema Computacional (2)				
Speedup	1,10	0,53	0,40	0,63
Total Mensagens	190085	209269	152920	208698

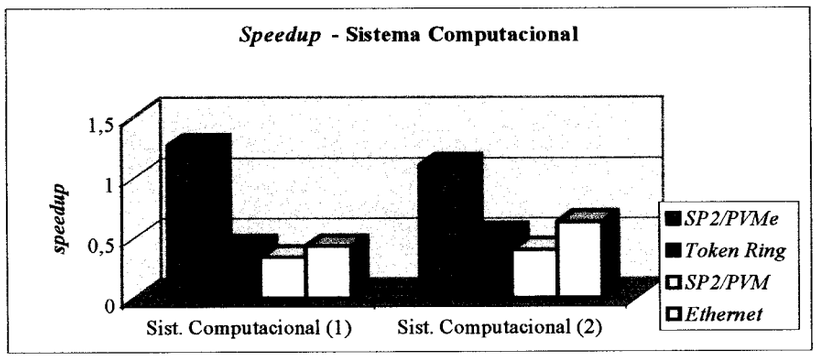

Figura 7.2 – *Speedup* - Sistema Computacional (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido para o modelo considerando as configurações apresentadas nas figuras 5.8 (Sistema Computacional (1)) e 5.9 (Sistema Computacional (2)).

Analisando o comportamento da simulação distribuída para o Modelo do Sistema Computacional, pode-se observar melhor desempenho da configuração 1 para a plataforma *SP2/PVMe*. Apesar do aumento médio de 59% na comunicação para essa configuração, a melhor distribuição da carga de trabalho possibilitou um melhor desempenho da configuração na plataforma *SP2/PVMe*.

Nas plataformas *Ethernet*, *Token Ring* e *SP2/PVM* a configuração 2 apresentou melhor desempenho. A restrição do meio de comunicação de baixo desempenho nas plataformas baseadas em *cluster* (*Ethernet* e *Token Ring*) e o fato do *PVM* não estar adaptado para a arquitetura *SP2* impõem uma sobrecarga na

comunicação entre os processos na execução da simulação distribuída. Assim, a redução média de 59% na comunicação para a configuração 2 possibilitou o melhor desempenho dessa configuração para o Modelo do Sistema Computacional.

Hipótese 1: Conclusão

A plataforma *SP2/PVMe* apresenta o menor tempo de comunicação entre os processos, em função das características da arquitetura. Por esse motivo o aumento da comunicação não provocou redução do desempenho. Já para o *SP2/PVM*, apesar das características físicas da arquitetura, obteve-se o pior desempenho devido ao tempo excessivo da execução distribuída e o tempo reduzido da execução seqüencial da simulação.

Nas plataformas baseadas em *cluster* com meio de comunicação de baixo desempenho, o tempo de execução da simulação distribuída impede a obtenção de um desempenho satisfatório. Apesar do *speedup* obtido para o *cluster Ethernet* ser superior ao *speedup* do *cluster Token Ring*, o tempo de execução da simulação seqüencial e distribuída no *Ethernet* são extremamente altos.

Desta forma, pode-se concluir que **a forma de partição dos processos lógicos na simulação distribuída deve ser realizada em função das características da plataforma de *hardware/software* utilizada**. Ou seja, a partição deve buscar reduzir a comunicação para plataformas com meio de comunicação de baixo desempenho e maximizar o balanceamento para plataforma paralela, com meio de comunicação de alto desempenho.

7.2.1.2 Hipótese 2: Partição do Modelo em Processos Lógicos (2)

Em modelos que apresentam estrutura com *feedback*, a forma de partição deve agrupar em um único processo lógico os recursos que constituem esse *feedback*, minimizando a comunicação em detrimento do balanceamento de carga dos processos.

Exemplo 1: Modelo Hipotético 1

O Modelo Hipotético 1, descrito na seção 5.4.7.1, foi analisado considerando duas configurações distintas para os processos (o impacto gerado pela presença de pontos de decisão no sistema para o desempenho da simulação é analisado na seção 7.3.3). Na configuração 1 (Modelo Hipotético 1 (1) - figura 5.13) o processo lógico 0 é constituído apenas por dois recursos, e o processo lógico 1 engloba os demais recursos pertencentes ao sistema (nesse caso, optou-se por isolar parte dos recursos que constituem um *feedback*).

Na configuração 2 (Modelo Hipotético 1 (2) - figura 5.14) foram agrupados no processo lógico 0 os recursos que constituem uma estrutura em série, e no processo lógico 1, os demais recursos do sistema. O *speedup* obtido para as configurações do modelo e o número total de mensagens são apresentados na tabela 7.3. A representação gráfica do *speedup* é apresentada na figura 7.3.

Tabela 7.3 – *Speedup* – Modelo Hipotético 1 (Partição dos Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Modelo Hipotético 1 (1)				
Speedup	1,25	0,75	0,50	1,03
Total Mensagens	169776	168234	167996	170177
Modelo Hipotético 1 (2)				
Speedup	1,20	0,60	0,43	0,84
Total Mensagens	274337	269194	287908	269194

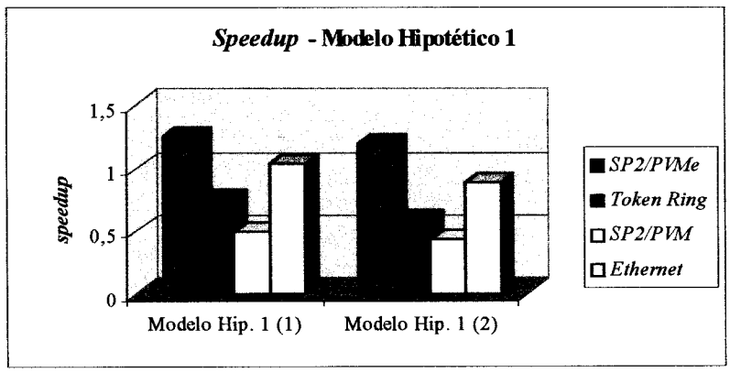

Figura 7.3 – *Speedup* - Modelo Hipotético 1 (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo Hipotético 1, com as configurações apresentadas nas figuras 5.13 (Modelo Hipotético 1 (1)) e 5.14 (Modelo Hipotético 1 (2)).

Para o Modelo Hipotético 1 a configuração 1 apresentou melhor desempenho para todas as plataformas utilizadas, como pode ser observado na tabela 7.3. Esse aumento ocorreu em função da otimização na forma de partição dos processos lógicos, que possibilitou agrupar parte dos recursos que constituem o *feedback* e assim, reduzir a comunicação entre esses processos.

A configuração 2 apresentou um aumento médio de 64% no número total de mensagens geradas pelos processos, em relação à configuração 1: as mensagens nulas tiveram um aumento médio de 40% e as mensagens completas, um aumento médio de

80%. Como analisado no capítulo 6, a comunicação não representa um fator decisivo para o desempenho na plataforma *SP2/PVMe*. Porém, a redução da comunicação, decorrente da forma de partição utilizada, pode implicar em um pequeno aumento do desempenho nessa plataforma.

Exemplo 2: Modelo Hipotético 2

Para o Modelo Hipotético 2, descrito na seção 5.4.7.2, foram consideradas duas configurações distintas para os processos lógicos. Na configuração 1 (Modelo Hipotético 2 (1) - figura 5.15) o processo lógico 0 é constituído apenas por dois recursos, e o processo lógico 1 engloba os demais recursos pertencentes ao sistema (nesse caso, optou-se por isolar parte dos recursos que constituem um *feedback*). Na configuração 2 (Modelo Hipotético 2 (2) - figura 5.16) foram agrupados no processo lógico 0 os recursos que constituem uma estrutura em série, e no processo lógico 1, os demais recursos do sistema.

O *speedup* obtido para as configurações do modelo é apresentado na tabela 7.4 juntamente com o número total de mensagens. A figura 7.4 apresenta o *speedup* representado graficamente.

Tabela 7.4 – *Speedup* – Modelo Hipotético 2 (Partição dos Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Modelo Hipotético 2 (1)				
Speedup	1,13	0,76	0,55	0,90
Total Mensagens	62589	62344	61933	62837
Modelo Hipotético 2 (2)				
Speedup	1,07	0,52	0,39	0,67
Total Mensagens	136173	138269	138208	138024

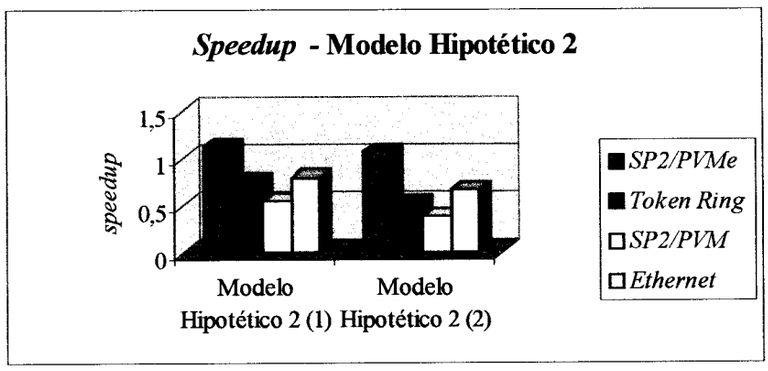

Figura 7.4 – *Speedup* - Modelo Hipotético 2 (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo Hipotético 2, com as configurações apresentadas nas figuras 5.15 (Modelo Hipotético 2 (1)) e 5.16 (Modelo Hipotético 2 (2)).

Com os dados obtidos pode-se observar melhor desempenho da simulação distribuída considerando a configuração 1 para a partição do modelo em processos lógicos. Nessa configuração o número total de mensagens geradas corresponde em média a metade do número de mensagens geradas na configuração 2. A redução da comunicação e a melhor distribuição dos recursos (agrupando parte dos recursos envolvidos no *feedback* em um mesmo processo) possibilitou o melhor desempenho da configuração 1 para o Modelo Hipotético 2.

Exemplo 3: Modelo Hipotético 4

O Modelo Hipotético 4, descrito na seção 5.4.7.4, foi analisado considerando duas configurações distintas constituídas por dois processos, considerando 30% de probabilidade de uma tarefa solicitar a utilização do recurso REC1. Na configuração 1 (Modelo Hipotético 4 (1) - figura 5.19) o processo lógico 0 é constituído pelos recursos que constituem o *feedback*, e o processo lógico 1 engloba o outro recurso.

Na configuração 2 (Modelo Hipotético 4 (2) - figura 5.20) o recurso REC1 foi agrupado no processo lógico 0 e os demais recursos foram agrupados no processo lógico 1. O *speedup* obtido para as configurações do modelo e o número total de mensagens são apresentados na tabela 7.5. A representação gráfica do *speedup* é apresentada na figura 7.5.

Tabela 7.5 – *Speedup* – Modelo Hipotético 4 (Partição dos Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Modelo Hipotético 4 (1)				
Speedup	1,20	0,61	0,45	0,93
Total Mensagens	139679	145552	142742	144628
Modelo Hipotético 4 (2)				
Speedup	1,07	0,25	0,12	0,53
Total Mensagens	259003	257990	257990	258649

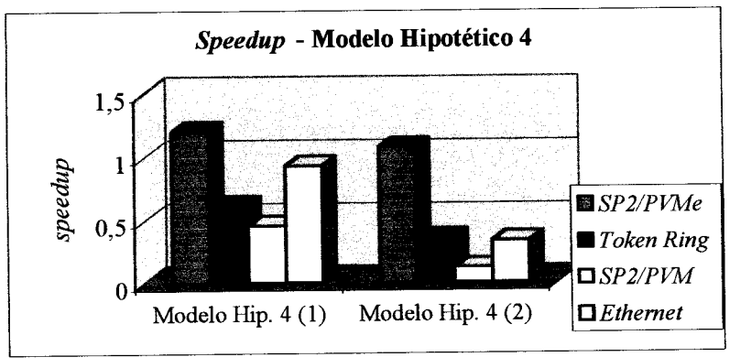

Figura 7.5 – *Speedup* - Modelo Hipotético 4 (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo Hipotético 4, com as configurações apresentadas nas figuras 5.19 (Modelo Hipotético 4 (1)) e 5.20 (Modelo Hipotético 4 (2)).

Para o modelo pode-se observar melhor desempenho da simulação distribuída para a configuração 1 em todas as plataformas consideradas. Nessa configuração o

número total de mensagens geradas foi praticamente metade do número de mensagens geradas na configuração 2.

A redução da comunicação e a melhor distribuição dos recursos (agrupando os recursos envolvidos no *feedback* em um mesmo processo) possibilitou o melhor desempenho da configuração 1.

Exemplo 4: Modelo Hipotético 5

O Modelo Hipotético 5 foi analisado considerando as configurações apresentadas nas figuras 5.21, 5.22 e 5.23. Na configuração 1 os recursos REC1, REC2 e REC3 foram agrupados em um processo lógico e o recurso REC4 foi associado em outro processo.

A configuração 2 apresenta os recursos REC1 e REC2 em um processo, e os recursos REC3 e REC4 em outro processo. Na configuração 3 os recursos que constituem o *feedback* (REC2 e REC3) foram agrupados em um processo, e os demais recursos (REC1 e REC4) foram agrupados em outro processo lógico. A tabela 7.6 apresenta o *speedup* e o número total de mensagens para cada caso e a figura 7.6 mostra a representação gráfica do *speedup*.

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Tabela 7.6 – *Speedup* – Modelo Hipotético 5 (Partição dos Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Modelo Hipotético 5 (1)				
Speedup	1,13	0,73	0,47	1,07
Total Mensagens	160749	166558	157443	151546
Modelo Hipotético 5 (2)				
Speedup	1,09	0,48	0,42	0,70
Total Mensagens	312800	314738	313591	318401
Modelo Hipotético 5 (3)				
Speedup	1,11	0,60	0,45	0,99
Total Mensagens	178316	187245	183471	185525

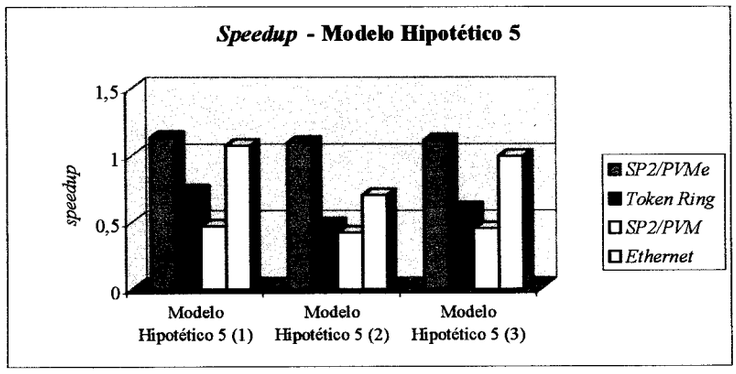

Figura 7.6 – *Speedup* - Modelo Hipotético 5 (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo Hipotético 5, com as configurações apresentadas nas figuras 5.21 (Modelo Hipotético 5 (1)), 5.22 (Modelo Hipotético 5 (2)) e 5.23 (Modelo Hipotético 5 (3)).

Para o Modelo Hipotético 5 o melhor desempenho foi obtido considerando a configuração 1 para todas as plataformas utilizadas. Essa configuração apresentou também o menor número de mensagens, em função da otimização na distribuição dos recursos nos processos lógicos.

A configuração 2 apresentou o pior desempenho, em função do grande número de mensagens geradas. O aumento médio observado para a comunicação foi

de 98% em relação à configuração 1. Esse aumento ocorre pelo fato dos recursos envolvidos no *feedback* terem sido representados em processos lógicos distintos. O impacto gerado pela sobrecarga da comunicação, principalmente para as plataformas distribuídas, e a forma de configuração dos processos lógicos foram os fatores determinantes para o desempenho insatisfatório da simulação.

A configuração 3 apresentou um aumento médio de 15% na comunicação em relação à configuração 1. Apesar desse aumento, o desempenho obtido foi satisfatório para a plataforma *SP2/PVMe*: o aumento da comunicação provocou uma queda de apenas 2% no desempenho em relação à configuração 1. Para as demais plataformas o impacto do aumento da comunicação foi decisivo para a queda do desempenho.

Hipótese 2: Conclusão

Considerando modelos com recursos formando estruturas de *feedback* (inclusive modelos com estrutura em série) um melhor desempenho é obtido com a redução da comunicação, agrupando os recursos envolvidos no *feedback* em um mesmo processo lógico. Esse fato ocorre independente do tipo de plataforma utilizada, seja para arquitetura paralela com meio de comunicação de alto desempenho, seja para arquiteturas distribuídas com meio de comunicação de baixo desempenho. Como analisado no capítulo 6, a comunicação não representa um elemento determinante para o desempenho na plataforma *SP2/PVMe*. Porém, a redução na comunicação nessa plataforma pode implicar em pequeno aumento do *speedup* para alguns tipos de modelos.

Pode-se concluir que a **partição dos processos lógicos deve ser realizada de forma a agrupar em um mesmo processo lógico os recursos envolvidos em *feedback***.

7.2.1.3 Hipótese 3: Partição do Modelo em Processos Lógicos (3)

A forma de partição do modelo deve evitar agrupar em um mesmo processo lógico recursos que não se comunicam diretamente, "quebrando" o fluxo das tarefas no sistema, uma vez que isso torna os processos ociosos e aumenta a comunicação entre esses processos degradando o desempenho da simulação.

Exemplo 1: *Trice*

Como exemplo foram utilizadas duas configurações para o modelo *Trice*, ambas constituídas por dois processos lógicos e considerando 70% de probabilidade de uma tarefa requisitar a utilização da unidade de disco. A configuração 1 (*Trice* (1) - figura 5.4) agrupa os processadores *Front End* em um processo e as unidades de disco e processamento em outro processo. Na configuração 2 (*Trice* (3) - figura 5.6) o processador *PFEin* foi agrupado juntamente com a unidade de disco em um processo, e o processador *PFEout* foi colocado em outro processo com a unidade de processamento.

O *speedup* obtido e o número total de mensagens geradas para cada caso são mostrados na tabela 7.7, e a figura 7.7 apresenta uma representação gráfica do *speedup*.

Tabela 7.7 – *Speedup – Trice* (Partição dos Processos Lógicos).

Modelo \ Plataforma	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
***Trice* (1)**				
Speedup	1,12	0,65	0,44	1,04
Total Mensagens	196084	198490	206509	200800
***Trice* (3)**				
Speedup	1,04	0,28	0,13	0,50
Total Mensagens	674896	633128	640215	631425

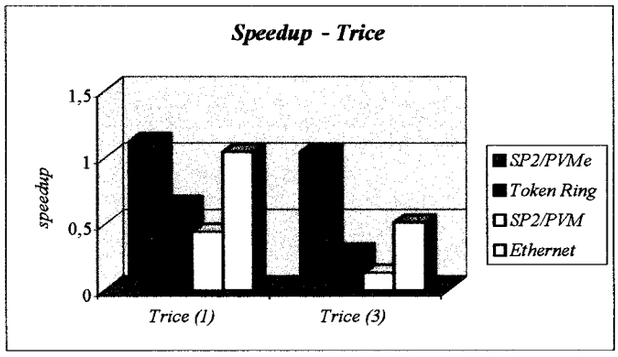

Figura 7.7 – *Speedup* - *Trice* (Partição dos Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo *Trice* com as configurações das figuras 5.4 (*Trice* (1)) e 5.6 (*Trice* (3)).

Considerando a simulação distribuída para o modelo *Trice*, a configuração 3 apresentou um desempenho insatisfatório para todas as plataformas utilizadas. Nessa configuração, o número de mensagens completas geradas foi praticamente seis vezes maior que na configuração 1, uma vez que o fluxo das tarefas no sistema foi dividido entre os processos. O aumento médio do número total de mensagens em relação à configuração 1 foi de 222%.

Hipótese 3: Conclusão

A divisão dos recursos de um modelo nos processos lógicos da simulação distribuída exerce influência no número de mensagens geradas pelos processos. A separação dos recursos pode gerar um grande aumento da comunicação e conseqüentemente, reduzir o desempenho da simulação. Observa-se para a plataforma *SP2/PVMe* a menor variação do *speedup* em função do aumento da comunicação.

Pode-se concluir então que **a forma de partição dos processos deve agrupar os recursos em uma seqüência lógica, evitando o aumento desnecessário da comunicação.**

7.2.1.4 Conclusão - Partição do Modelo em Processos Lógicos

A partição de um modelo em processos lógicos deve observar três pontos fundamentais: a comunicação e o balanceamento de carga dos processos, a presença de *feedback* e o relacionamento entre os recursos que compõem o modelo.

Em função da plataforma utilizada, deve-se optar por maximizar o balanceamento ou reduzir a comunicação. Outro ponto importante é a presença de recursos envolvidos em *feedback*. Nos modelos com essa característica a forma de configuração deve agrupar em um mesmo processo lógico os recursos relacionados ao *feedback* com o intuito de reduzir a comunicação e otimizar a execução da simulação. O terceiro ponto a ser considerado na partição dos processos consiste em agrupar os recursos em uma seqüência lógica, de acordo com o fluxo das tarefas no sistema.

7.2.2 Alteração da Granulosidade e Número de Processos Lógicos

No desenvolvimento de aplicações paralelas a granulosidade das unidades de trabalho submetidas aos elementos de processamento representa uma questão importante para o desempenho da aplicação. Esse desempenho pode variar em função do tipo de granulosidade adotada (fina, média ou grossa) e das características da plataforma utilizada (como a capacidade de processamento e do meio de comunicação).

7.2.2.1 Hipótese 4: Granulosidade e Número de Processos Lógicos

Plataformas baseadas em sistemas distribuídos (considerando máquinas paralelas virtuais com meio de comunicação de baixo desempenho) são adequadas para aplicações com granulosidade grossa. Já as arquiteturas paralelas permitem a obtenção de um bom desempenho com a utilização de granulosidade fina, mesmo que isso implique em aumento da comunicação. Assim, o número de processos e conseqüentemente a granulosidade desses processos devem ser determinados em função das características da plataforma utilizada.

Exemplo 1: Servidor Central

Para verificar a influência da granulosidade no desempenho da simulação distribuída utilizou-se o Modelo do Servidor Central[1], apresentado nas figuras 5.2 e 5.3.

[1] No *PVM* (domínio público) executando no *SP2* o paradigma de programação é *SPMD*, já que o *PVM* executa sobre o *MPI*. Devido às características do *ParSMPL*, que utiliza o paradigma *MPMD*, o modelo não foi executado nessa plataforma.

Na configuração 1 (Servidor Central (1) - figura 5.2) foram utilizados dois processos lógicos, um processo contendo a unidade de processamento e outro processo englobando as unidades de disco. Na configuração 2 (Servidor Central (2) - figura 5.3) foram utilizados três processos lógicos, resultando em uma granulosidade mais fina, quando comparada à configuração 1. Nessa configuração o elemento de processamento permaneceu em um processo lógico, e as unidades de disco foram agrupadas em dois processos lógicos (2 unidades de disco em cada processo).

A tabela 7.8 apresenta o *speedup* obtido com a execução da simulação distribuída do Modelo do Servidor Central e o número total de mensagens geradas pelos processos, considerando as duas configurações citadas anteriormente. A figura 7.8 apresenta esses dados dispostos graficamente (parte desses dados pode ser encontrada em [Bruschi et al. 1999]).

Tabela 7.8 – *Speedup* - Servidor Central (Granulosidade e Número de Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*Ethernet*
Servidor Central (1)			
Speedup	1,34	0,47	0,75
Total Mensagens	163080	166890	169889
Servidor Central (2)			
Speedup	1,46	0,43	0,68
Total Mensagens	198419	207071	202988

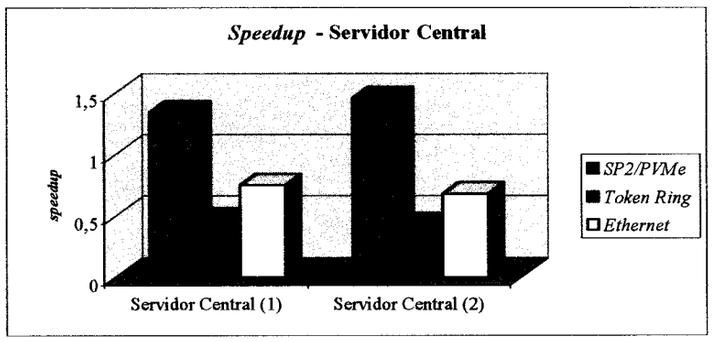

Figura 7.8 – *Speedup* - Servidor Central (Granulosidade e Número de Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo do Servidor Central, com as configurações dos processos lógicos das figuras 5.2 (Servidor Central 1) e 5.3 (Servidor Central 2).

Na plataforma *SP2/PVMe* verifica-se um aumento do *speedup*, quando comparadas as configurações 1 e 2. Na configuração 2, apesar do aumento da comunicação (observou-se um aumento aproximado de 22%), foi possível também um aumento do *speedup*, uma vez que a carga de trabalho foi melhor distribuída entre os processos.

Nas plataformas *Token Ring* e *Ethernet* a configuração 2 apresentou uma redução do desempenho, fato justificado pelo número de processos e o aumento da comunicação entre esses processos (houve um aumento médio de 25% no número total de mensagens).

Exemplo 2: Sistema Computacional

Para a realização das análises foram utilizadas as configurações para o Modelo do Sistema Computacional considerando 40% de probabilidade de uma tarefa deixar o sistema. Na configuração 1 (Sistema Computacional (1) - figura 5.8) foram utilizados

dois processos lógicos, um processo contendo a unidade de processamento e uma das unidades de disco, e outro processo com a outra unidade de disco.

Na configuração 2 (Sistema Computacional (3) - figura 5.10) foram utilizados três processos lógicos, um processo para cada um dos recursos do modelo. Nessa configuração obteve-se uma granulosidade menor quando comparada à configuração 1. A tabela 7.9 apresenta o *speedup* obtido para a execução da simulação distribuída do modelo considerando as configurações citadas, além do número total de mensagens geradas pelos processos. A figura 7.9 mostra a representação gráfica do *speedup*.

Tabela 7.9 – *Speedup* – Sistema Computacional (Granulosidade e Número de Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*Ethernet*
Sistema Computacional (1)			
Speedup	1,26	0,43	0,43
Total Mensagens	316023	341638	335820
Sistema Computacional (3)			
Speedup	1,28	0,38	0,40
Total Mensagens	407702	481892	431014

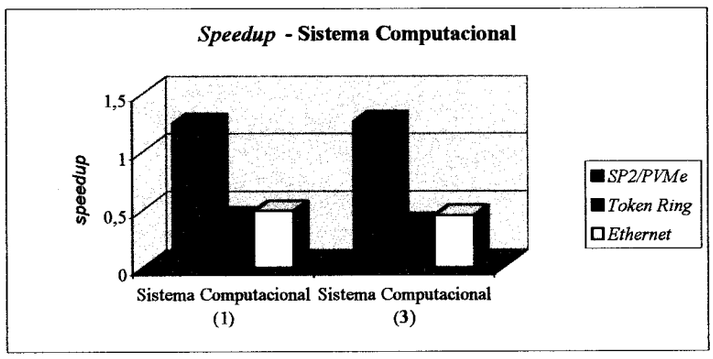

Figura 7.9 – *Speedup* – Sistema Computacional (Granulosidade e Número de Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo do Sistema Computacional, com as configurações dos processos das figuras 5.8 (Sistema Computacional (1)) e 5.10 (Sistema Computacional (3)).

Na plataforma *SP2/PVMe* a configuração 2 apresentou melhor desempenho para o Sistema Computacional em função das características da plataforma, apesar do aumento médio de 33% no número de mensagens em relação à configuração 1.

Nas plataformas *Token Ring* e *Ethernet* a configuração 2 apresentou uma redução do desempenho, fato justificado pelo aumento do número de processos e a sobrecarga da comunicação entre esses processos.

Hipótese 4: Conclusão

Na plataforma paralela o aumento do número de processos lógicos possibilita aumentar o desempenho da simulação distribuída, mesmo com o aumento da comunicação entre esses processos. Já nas plataformas distribuídas, como os *clusters* que utilizam meio de comunicação de baixo desempenho, a redução do número de processos e da comunicação é fundamental para a execução da simulação distribuída.

A figura 7.10 apresenta, como exemplo, o tempo de comunicação e processamento obtido com a execução da simulação distribuída dos modelos Servidor Central (1) e Servidor Central (2), nas plataformas *SP2/PVMe* e *Token Ring*. Na plataforma *SP2/PVMe*, a configuração 2, mesmo com o aumento da comunicação, possibilitou maior desempenho (figura 7.9). Para a plataforma *Token Ring* o aumento da comunicação, notadamente das mensagens nulas, foi responsável pela sobrecarga e conseqüentemente, pela redução do desempenho da simulação (figura 7.9).

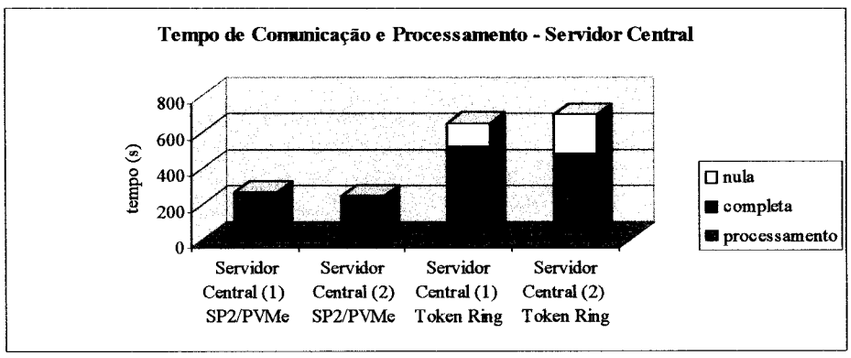

Figura 7.10 – Tempo de Comunicação e Processamento – Servidor Central. A figura apresenta o tempo de comunicação e processamento para o modelo nas plataformas *SP2/PVMe* e *Token Ring*.

Desta forma pode-se concluir que **o tipo de granulosidade e o número de processos lógicos adotados para o desenvolvimento da simulação distribuída devem ser determinados em função das características da plataforma utilizada.**

7.2.2.2 Hipótese 5: Granulosidade e Número de Processos Considerando Modelos com Estruturas em Série

Em modelos com estruturas em série o fluxo seqüencial das tarefas no sistema e as altas taxas de comunicação podem comprometer o desempenho da simulação distribuída. Nesse tipo de modelo a partição deve reduzir o número de processos e conseqüentemente, a comunicação entre os mesmos, para a obtenção de melhor desempenho da simulação distribuída.

Exemplo 1: Modelo Hipotético 3

Para analisar o comportamento de um modelo com estrutura em série utilizou-se o modelo cujas configurações são apresentadas nas figuras 5.17 (Modelo Hipotético 3 (1)) e 5.18 (Modelo Hipotético 3 (2)). A configuração 1 é constituída por três processos lógicos e na configuração 2 os recursos foram agrupados em dois processos lógicos.

O *speedup* obtido com o modelo e o número total de mensagens são mostrados na tabela 7.10, e a figura 7.11 apresenta uma representação gráfica do *speedup* obtido para o modelo.

Tabela 7.10 – *Speedup* – Modelo Hipotético 3 (Granulosidade e Número de Processos Lógicos).

Plataforma / Modelo	*SP2/ PVMe*	*Token Ring*	*Ethernet*
Modelo Hipotético 3 (1)			
Speedup	1,03	0,35	0,40
Total Mensagens	472854	490032	504992
Modelo Hipotético 3 (2)			
Speedup	1,10	0,48	0,65
Total Mensagens	242489	246582	246338

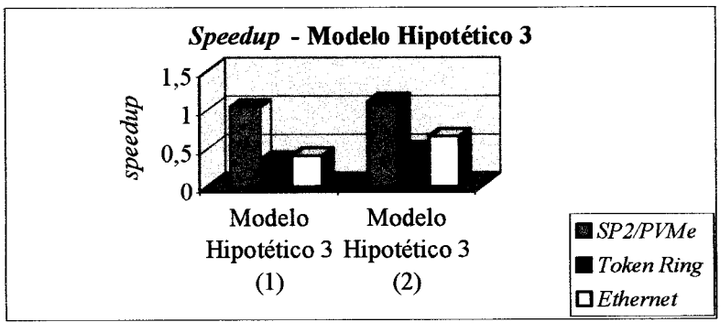

Figura 7.11 – *Speedup* – Modelo Hipotético 3 (Granulosidade e Número de Processos Lógicos). A figura apresenta o *speedup* obtido para o Modelo Hipotético 3 com as configurações dos processos das figuras 5.17 (Modelo Hipotético 3 (1)) e 5.18 (Modelo Hipotético 3 (2)).

Devido às características do modelo que impõe um fluxo seqüencial das tarefas no sistema, a comunicação entre os processos é intensa. Na configuração 1 as mensagens nulas correspondem em média a 80% do total de mensagens enviadas pelos processos. Na configuração 2 as mensagens nulas correspondem em média a 59% do total de mensagens geradas durante a simulação.

A configuração 2 apresentou melhor desempenho para todas as plataformas consideradas, devido ao menor número de processos e a redução na comunicação entre esses processos. Porém, apenas a plataforma *SP2/PVMe* apresentou um bom

desempenho para a simulação. Nas demais plataformas, o *speedup* obtido foi insatisfatório mesmo com a redução da comunicação e do número de processos lógicos.

Hipótese 5: Conclusão

Em modelos com estrutura em série as tarefas mantêm um fluxo seqüencial entre os recursos do sistema. Desta forma, o tráfego de mensagens entre os processos é intenso e compromete o desempenho da simulação distribuída. Deve-se observar que para a plataforma *SP2/PVMe* a comunicação não representa um fator determinante para o desempenho, porém a redução na comunicação possibilita um pequeno aumento do *speedup*.

Assim, para esse tipo de modelo, **a partição dos processos lógicos deve adotar granulosidade grossa, reduzindo o número de processos e a comunicação entre esses processos.**

7.2.2.3 Comentários - Granulosidade e Número de Processos Lógicos

Como demonstrado nos exemplos, a determinação do número de processos lógicos na simulação distribuída conservativa deve ser realizada em função do tipo de plataforma utilizada e das características do modelo a ser particionado.

Na arquitetura paralela com meio de comunicação de alto desempenho, o aumento do número de processos permite otimizar a execução da simulação mesmo com o aumento da comunicação. Considerando plataformas que utilizam meio de comunicação com baixo desempenho, a comunicação pode ser um ponto negativo

para a simulação, fator que justifica a utilização de um número reduzido de processos com granulosidade mais grossa.

Para modelos com estrutura em série, semelhantes ao exemplo apresentando, a partição do modelo deve ser realizada de forma a reduzir o número de processos e a comunicação entre esses processos.

7.2.3 Alteração nos Parâmetros da Aplicação

As variações nos parâmetros de uma aplicação, como taxas de probabilidade em pontos de decisão e tempo de chegada de tarefas, exercem influência no desempenho da simulação, refletindo no número de tarefas presentes no sistema e conseqüentemente no número de mensagens geradas pelos processos. Esta seção apresenta o comportamento da simulação distribuída quando alterações nas características dos modelos são consideradas.

7.2.3.1 Hipótese 6: Probabilidade nos Pontos de Decisão

As alterações das probabilidades nos pontos de decisão podem resultar em aumento da população do sistema e em aumento da comunicação entre alguns processos. Dependendo da probabilidade alterada, pode-se ter uma inversão do caminho mais seguido pelas mensagens e conseqüentemente a necessidade de se adaptar a divisão dos processos lógicos. Nos modelos em equilíbrio, essas alterações podem possibilitar um melhor desempenho para plataformas paralelas, uma vez que acarretam um aumento da comunicação. Quando essas alterações resultam em saturação do sistema, o desempenho obtido é insatisfatório para todas as plataformas.

Exemplo 1: Sistema Computacional

Para avaliar o impacto dos pontos de decisão do sistema no desempenho da simulação distribuída, o Modelo do Sistema Computacional foi avaliado utilizando-se diferentes probabilidades de uma tarefa deixar o sistema, considerando as configurações apresentadas nas figuras 5.8 (Sistema Computacional 1) e 5.9 (Sistema Computacional 2). A simulação foi executada considerando-se probabilidades de uma tarefa deixar o sistema variando entre 10% a 100% (a escolha das unidades de disco é realizada com igual probabilidade).

As figuras 7.12 e 7.13 apresentam respectivamente o *speedup* obtido e o total de mensagens completas e nulas geradas para a configuração 1 do modelo. Os dados correspondentes para a configuração 2 são apresentados nas figuras 7.14 e 7.15.

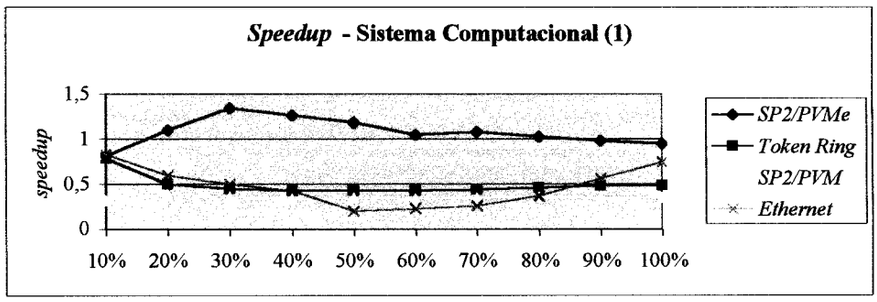

Figura 7.12 – *Speedup* – Sistema Computacional (1) (Probabilidade nos Pontos de Decisão). Para as análises foram utilizadas diferentes probabilidades de uma tarefa deixar o sistema. A escolha das unidades de disco é efetuada com igual probabilidade.

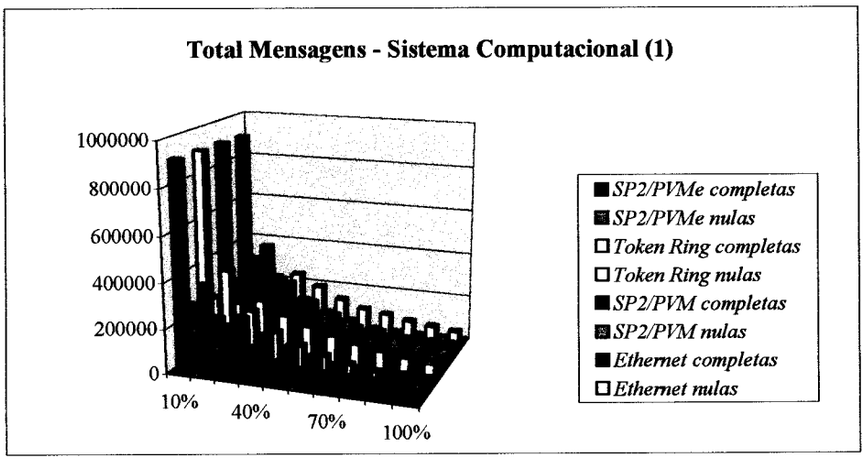

Figura 7.13 - Total de Mensagens– Sistema Computacional (1) (Probabilidade nos Pontos de Decisão). A figura apresenta o número total de mensagens nulas e completas geradas considerando diferentes probabilidades de uma tarefa deixar o sistema.

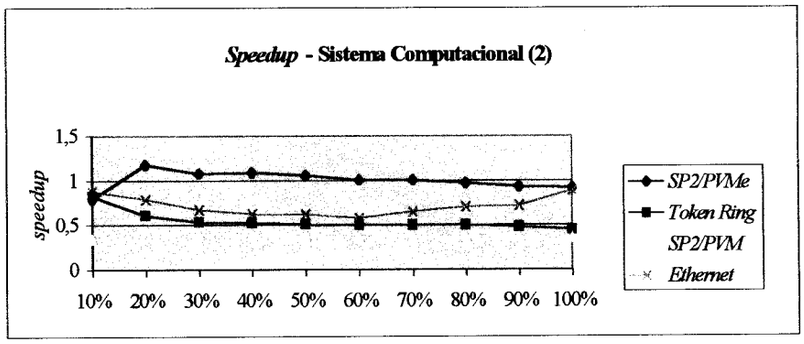

Figura 7.14 – *Speedup* – Sistema Computacional (2) (Probabilidade nos Pontos de Decisão). Para as análises foram utilizadas diferentes probabilidades de uma tarefa deixar o sistema. A escolha das unidades de disco é realizada com igual probabilidade.

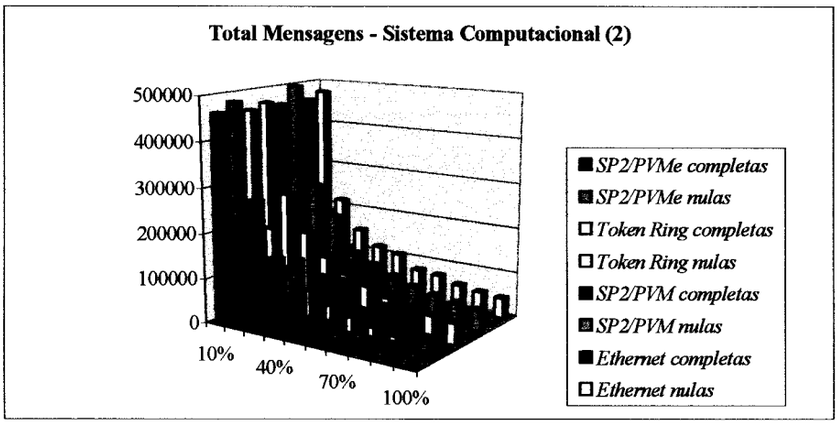

Figura 7.15 - Total de Mensagens– Sistema Computacional (2) (Probabilidade nos Pontos de Decisão). A figura apresenta o número total de mensagens considerando diferentes probabilidades de uma tarefa deixar o sistema.

A configuração 1 do Modelo do Sistema Computacional apresentou maior número de mensagens em relação à configuração 2, devido à configuração dos processos lógicos. Com as alterações nas probabilidades nos pontos de decisão do modelo o comportamento apresentado pela simulação considerando as duas configurações foi semelhante.

Com baixas taxas de probabilidade de uma tarefa deixar o sistema, a população presente no mesmo aumenta consideravelmente e o sistema entra em colapso. O número de mensagens geradas e o tempo de execução da simulação, tanto seqüencial quanto distribuída, também aumentam. Em todas as plataformas o desempenho obtido com 10% de probabilidade de uma tarefa deixar o sistema é justificado pelo aumento do tempo de execução da simulação seqüencial. Com 100% de probabilidade da tarefa deixar o sistema, o desempenho é justificado pela sobrecarga de envio das mensagens nulas. Nesses casos extremos o desempenho obtido pelas duas configurações é próximo.

No *SP2/PVMe* as variações no número de tarefas presente no sistema exercem maior influência no desempenho da simulação com grande número de tarefas no sistema. Para essa plataforma a configuração 1 apresentou melhor desempenho.

Nas plataformas *Token Ring* e *Ethernet* o bom desempenho com pouca probabilidade de uma tarefa deixar o sistema ocorre devido ao alto tempo de execução da simulação seqüencial. Com o sistema em equilíbrio o desempenho tende a estabilizar. A redução da comunicação devido ao aumento das taxas de saída das tarefas do sistema permite um aumento do desempenho para as plataformas. A configuração 2 possibilitou melhor desempenho nessas plataformas. Na plataforma *SP2/PVM* o desempenho piora com o sistema saturado e com o sistema em equilíbrio, o desempenho tende a ser estável.

Hipótese 6: Conclusão

As variações de probabilidade nos pontos de decisão de um modelo podem levar a alterações no número de tarefas presentes no sistema e conseqüentemente no número de mensagens trocadas pelos processos. Portanto, **pode-se concluir que as alterações nas probabilidades nos pontos de decisão do modelo podem alterar o desempenho da simulação em função do aumento/redução das tarefas no sistema.**

7.2.3.2 Comentários – Probabilidade nos Pontos de Decisão

As influências das taxas de probabilidade no desempenho da simulação distribuída podem ser observadas com maior intensidade nos casos extremos, ou seja,

com o aumento do número de tarefas no sistema e com o sistema ocioso. Para o *SP2/PVMe* o desempenho tende a piorar com o sistema em saturação ou com poucas tarefas no sistema. Na plataforma *SP2/PVM* o comportamento da simulação é estável, ou seja, as alterações no desempenho são pequenas com as variações na população do sistema e tende a piorar com o sistema saturado.

Para as plataformas *Token Ring e Ethernet* o melhor desempenho é obtido com um grande número de tarefas no sistema (nesse caso devido ao aumento do tempo de execução da simulação seqüencial). O desempenho tende a melhor com a redução da população do sistema e, conseqüentemente, da comunicação.

7.2.3.3 Hipótese 7: Tempo entre Chegadas de Tarefas

As alterações no tempo entre chegadas de tarefas no sistema considerando modelos do tipo fechado podem implicar em variações no desempenho da simulação devido às alterações no número de mensagens geradas pelos processos.

Exemplo 1: Sistema Computacional Simplificado

O Modelo do Sistema Computacional Simplificado (figura 5.7) foi executado com as variações no tempo entre chegadas de clientes especificadas na tabela 7.11, que apresenta também o número total de mensagens para cada situação. A figura 7.16 apresenta a representação gráfica do *speedup* obtido com o modelo.

Tabela 7.11 – *Speedup* – Sistema Computacional Simplificado (Tempo Entre Chegadas de Tarefas).

Plataforma	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Tempo Chegada 50.0				
Speedup	1,21	0,45	0,37	0,69
Total Mensagens	228728	246716	253130	251413
Tempo Chegada 100.0				
Speedup	1,18	0,36	0,34	0,41
Total Mensagens	242616	260757	281359	267295
Tempo Chegada 500.0				
Speedup	1,15	0,32	0,30	0,38
Total Mensagens	262776	282675	325542	292151

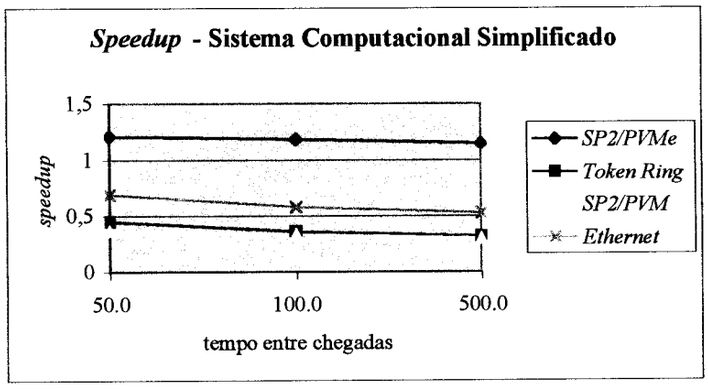

Figura 7.16 - *Speedup*– Sistema Computacional Simplificado (Tempo entre Chegadas de Tarefas). A figura apresenta o *speedup* obtido com o modelo considerando diferentes valores para o tempo entre chegadas de tarefas no sistema.

Considerando o exemplo apresentado as alterações no tempo entre chegadas de tarefas no sistema refletem alterações no número total de mensagens geradas pelos processos (verificou-se um aumento médio de 19% na comunicação entre os processos considerando os valores 50.0 e 500.0 no tempo entre chegada de tarefas), implicando então em diferenças no desempenho da simulação distribuída para todas as plataformas consideradas.

Na plataforma *SP2/PVMe* a alteração nos tempos entre chegadas de tarefas no sistema provocou uma redução máxima de 5% no desempenho da simulação. Esse fato é justificado pelas características da plataforma, que utiliza um meio de comunicação de alto desempenho. Nesse caso o impacto do aumento do número de mensagens não afetou de forma significativa o desempenho da simulação.

Nas demais plataformas o impacto causado pelas alterações no desempenho foi significativo. Ocorreu redução de 19% no *speedup* para a plataforma *SP2/PVM,* 29% de redução no *speedup* para a plataforma *Token Ring* e 45% de redução no *speedup* para a plataforma *Ethernet*. Para essas plataformas, que utilizam meio de comunicação de baixo desempenho, o aumento do número de mensagens exerce grande influência no desempenho da simulação distribuída.

Exemplo 2: Sistema de Computação com Processamento por Transação

O modelo do Sistema de Computação com Processamento por Transação (figura 5.11) foi executado com as variações no tempo entre chegadas de clientes especificadas na tabela 7.12, que apresenta também o número total de mensagens para cada situação. A figura 7.17 apresenta a representação gráfica do *speedup* obtido para o modelo.

Tabela 7.12 – *Speedup* – Sistema de Computação com Processamento por Transação (Tempo Entre Chegadas de Tarefas).

Plataforma	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Tempo Chegada 100.0				
Speedup	1,41	0,46	0,34	0,64
Total Mensagens	242418	240475	244703	257854
Tempo Chegada 300.0				
Speedup	1,35	0,39	0,32	0,55
Total Mensagens	338774	346109	384786	411104
Tempo Chegada 500.0				
Speedup	1,33	0,33	0,29	0,39
Total Mensagens	342605	360003	393230	428324

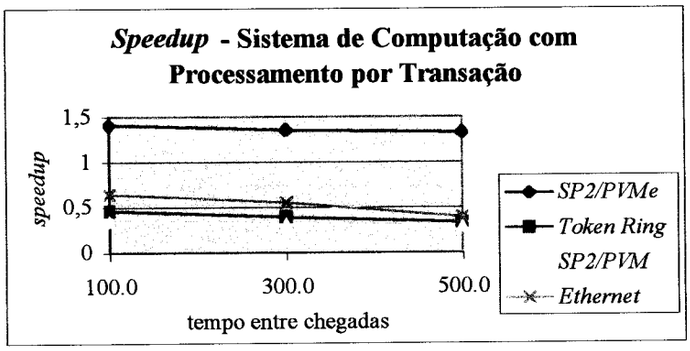

Figura 7.17 - *Speedup*– Sistema de Computação com Processamento por Transação (Tempo Entre Chegadas de Tarefas). A figura apresenta o *speedup* obtido para o modelo considerando diferentes valores para o tempo entre chegadas de tarefas no sistema.

Como observado na figura 7.17, o comportamento da simulação para o modelo em questão é semelhante ao comportamento apresentado pelo Modelo do Sistema Computacional Simplificado. Ou seja, as alterações no tempo entre chegadas das tarefas no sistema implicam em alterações do total de mensagens geradas pelos processos, resultando em diferenças no desempenho da simulação distribuída para todas as plataformas consideradas (o aumento médio no número de mensagens foi de

55%, considerando os valores 100.0 e 500.0 no tempo entre chegada de tarefas no sistema).

Na a plataforma *SP2/PVMe* a redução máxima do desempenho em função da alteração do tempo entre chegada de tarefas no sistema foi de 7%, em função das características da plataforma.

Já nas plataformas *SP2/PVM*, *Token Ring* e *Ethernet* a redução máxima verificada para o desempenho foi, respectivamente, de 15%, 28% e 39%. Nessas plataformas o aumento da comunicação implica na queda no desempenho da simulação distribuída.

Hipótese 7: Conclusão

As variações no tempo entre chegadas de tarefas no sistema implicam em alterações no número de mensagens geradas pelos processos lógicos na simulação. O aumento da comunicação decorrente dessas alterações implica em variações significativas no desempenho da simulação para as plataformas *Token Ring*, *Ethernet* e *SP2/PVM*. Pode-se concluir que as **alterações no tempo entre chegadas de clientes no sistema exercem maior influência no desempenho da simulação distribuída nas plataformas que utilizam meio de comunicação de baixo desempenho.**

7.2.3.4 Comentários – Tempo entre Chegadas de Tarefas

Modificações no tempo entre chegada de tarefas no sistema podem alterar o número de mensagens entre os processos. Com o aumento do número de mensagens

ocorrem variações significativas no desempenho da simulação distribuída em plataformas que utilizam meio de comunicação de baixo desempenho (incluindo o *SP2* utilizando uma biblioteca de comunicação não adaptada para a arquitetura).

Na arquitetura paralela que utiliza meio de comunicação de alto desempenho, o aumento da comunicação entre os processos lógicos e a sobrecarga decorrente desse aumento não constitui um fator determinante para o desempenho da simulação distribuída.

7.2.3.5 Hipótese 8: Tempo de Serviço dos Recursos

As alterações no tempo de serviço dos recursos podem implicar em alterações no desempenho da simulação quando essas alterações implicam em saturação no sistema, devido as modificações no número de mensagens geradas pelos processos.

Exemplo 1: Sistema Computacional Simplificado

A simulação do Modelo do Sistema Computacional Simplificado (figura 5.7) foi executada com variações no tempo de serviço dos recursos REC1 e REC2, para analisar o comportamento da simulação distribuída com essas alterações. O *speedup* obtido para cada situação e o número total de mensagens são apresentados na tabela 7.13 (a definição das classes C1, C2, C3 e C4 é apresentada na legenda da figura 7.18). A figura 7.18 apresenta uma representação gráfica do *speedup* (tsrec1 indica o tempo médio de serviço no recurso REC1 e tsrec2 indica o tempo médio de serviço no recurso REC2).

Tabela 7.13 – *Speedup* – Sistema Computacional Simplificado (Tempo de Serviço dos Recursos).

Plataforma	*SP2/ PVMe*	*Token Ring*	*SP2/ PVM*	*Ethernet*
Tempo Serviço C1				
Speedup	1,15	0,44	0,36	0,25
Total Mensagens	168631	184203	182546	184606
Tempo Serviço C2				
Speedup	1,11	0,90	0,48	0,85
Total Mensagens	287943	277776	291341	281688
Tempo Serviço C3				
Speedup	1,15	0,61	0,41	0,85
Total Mensagens	115021	116374	119158	118460
Tempo Serviço C4				
Speedup	0,99	0,90	0,46	0,92
Total Mensagens	288306	278259	293519	280210

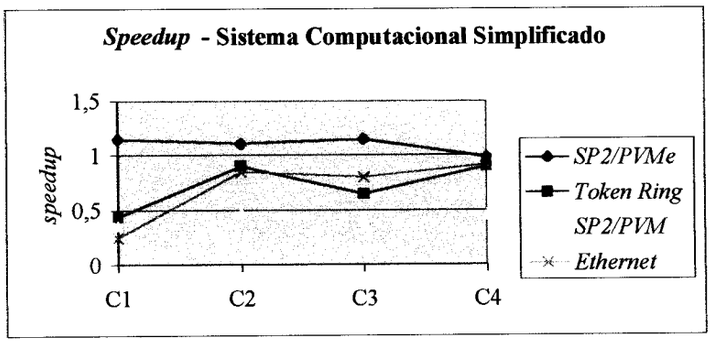

Figura 7.18 – *Speedup* – Sistema Computacional Simplificado (Tempo de Serviço dos Recursos). A figura apresenta o *speedup* obtido considerando os seguintes casos: C1: tsrec1 = 10.0 e tsrec2 = 20.0; C2: tsrec1 = 100.0 e tsrec2 = 20.0; C3: tsrec1 = 10.0 e tsrec2 = 100.0 e C4: tsrec1 = 100.0 e tsrec2 = 100.0. Em todos os casos o tempo de chegada de clientes foi 50.0.

Nos casos que provocam saturação no sistema (C2 e C4) o desempenho tende a cair para a plataforma *SP2/PVMe*, em decorrência do aumento do tempo de execução da simulação distribuída. Para as demais plataformas o *speedup* aumenta em função do aumento do tempo de execução da simulação seqüencial. Nesses casos,

o aumento do tempo de serviço dos recursos do sistema reflete na saturação do mesmo e no aumento do número de mensagens na execução da simulação distribuída.

Hipótese 8: Conclusão

As alterações nos tempos de serviços dos recursos do sistema podem levar a situações de saturação no mesmo, gerando um aumento do número de mensagens entre os processos. Para a plataforma paralela esse aumento resulta em uma pequena queda do desempenho, em função do aumento do tempo de execução da simulação distribuída. Nas plataformas distribuídas e o *SP2/PVM* o aumento do número de mensagens nas condições de saturação do sistema implica em aumento do desempenho devido ao aumento do tempo de execução da simulação seqüencial e distribuída.

Dessa forma pode-se concluir que **as modificações nos tempos de serviço dos recursos podem alterar de forma significativa o desempenho da simulação em situação de saturação do sistema para plataformas distribuídas**.

7.2.3.6 Comentários: Tempo de Serviço dos Recursos

As alterações nos tempos de serviço dos recursos do sistema podem levar esse sistema a situações de saturação, como por exemplo recursos com grandes filas de tarefas para serem executadas e recursos ociosos. Nessas situações, o número de mensagens na simulação distribuída aumenta consideravelmente e o desempenho da simulação na plataforma paralela tende a diminuir, em função do aumento do tempo de execução da simulação distribuída.

Nas demais plataformas, o aumento desproporcional do tempo de execução da simulação seqüencial em relação ao tempo de execução da simulação distribuída leva ao aumento do *speedup*.

7.2.3.7 Hipótese 9: Número de Tarefas em Sistema do Tipo Fechado

As alterações no número de tarefas escalonadas para execução em sistema do tipo fechado podem gerar variações no desempenho da simulação devido às alterações no número de mensagens geradas pelos processos lógicos.

Exemplo 1: Servidor Central

O Modelo do Servidor Central, considerando as configurações apresentadas nas figuras 5.2 e 5.3, foi executado com as variações no número de tarefas apresentadas na tabela 7.14 e 7.15. As tabelas apresentam também o número total de mensagens para cada situação. As figuras 7.19 e 7.20 apresentam a representação gráfica do *speedup* obtido para o modelo.

Tabela 7.14 – *Speedup* – Servidor Central (1) (Número de Tarefas).

Plataforma	*SP2/ PVMe*	*Token Ring*	*Ethernet*
Número Tarefas 5			
Speedup	1,34	0,47	0,75
Total Mensagens	163080	166890	169889
Número Tarefas 50			
Speedup	1,41	0,52	0,80
Total Mensagens	156624	159959	163959
Número Tarefas 100			
Speedup	1,43	0,52	0,81
Total Mensagens	154219	159818	162812
Número Tarefas 500			
Speedup	1,51	0,55	0,85
Total Mensagens	154302	159775	159389

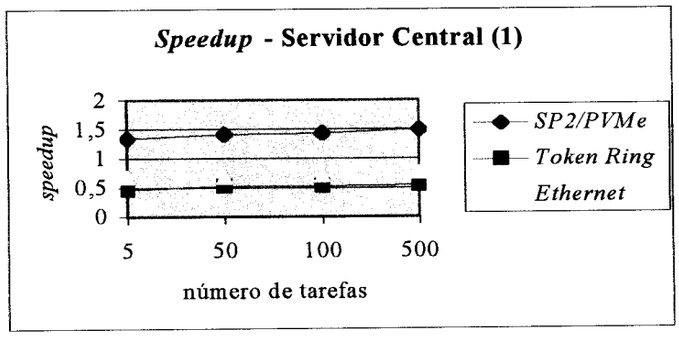

Figura 7.19 – *Speedup* – Servidor Central (1) (Número de Tarefas). A figura apresenta o *speedup* obtido para o modelo considerando variações no número de tarefas escalonadas inicialmente para execução (figura 5.2).

Para a configuração 1 do Modelo do Servidor Central o aumento do número de tarefas no sistema implicou na redução do número de mensagens geradas pelos processos e conseqüentemente, no aumento do desempenho da simulação. O aumento para as plataformas *SP2/PVMe*, *Token Ring* e *Ethernet* foi, respectivamente, de 13%, 17% e 14% (considerando desempenho obtido para 5 e 500 tarefas no sistema).

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Tabela 7.15 – *Speedup* – Servidor Central (2) (Número de Tarefas).

Plataforma	*SP2/ PVMe*	*Token Ring*	*Ethernet*
Número Tarefas 5			
Speedup	1,46	0,43	0,68
Total Mensagens	198419	207071	202988
Número Tarefas 50			
Speedup	1,48	0,50	0,75
Total Mensagens	161557	165348	163297
Número Tarefas 100			
Speedup	1,50	0,52	0,78
Total Mensagens	160255	164213	162601
Número Tarefas 500			
Speedup	1,52	0,54	0,80
Total Mensagens	158035	162565	159923

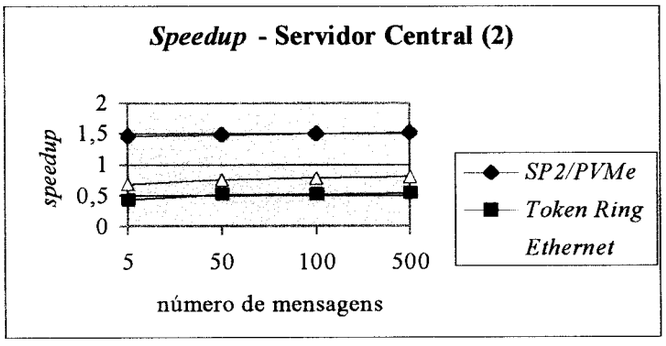

Figura 7.20 – *Speedup* – Servidor Central (2) (Número de Tarefas). A figura apresenta o *speedup* obtido para o modelo considerando variações no número de tarefas escalonadas inicialmente para execução (figura 5.3).

Analisando a configuração 2 pode-se observar que o aumento da população no sistema provoca alterações no desempenho da simulação para todas as plataformas consideradas, devido a variação na comunicação entre os processos. Na a plataforma *SP2/PVMe* verificam-se as menores variações no desempenho da simulação, em função das características do *switch* de alto desempenho (aumento de 4% no *speedup* considerando 5 e 500 tarefas no sistema). Já para as demais plataformas a redução do

número de mensagens possibilitou um maior aumento do desempenho: 18% para *Ethernet* e 26% para *Token Ring*, também considerando 5 e 500 tarefas no sistema.

Considerando as configurações 1 e 2, o comportamento observado para o desempenho, da simulação foi semelhante, apresentando aumento do desempenho em função do aumento do número de tarefas no sistema. Para a configuração 1, com menor número de processos lógicos, a variação no desempenho foi menor para as plataformas distribuídas. Considerando o *SP2/PVMe* a menor variação ocorreu com a configuração 2.

Hipótese 9: Conclusão

O aumento do número de tarefas no sistema provoca alterações no número de mensagens geradas pelos processos. Com a redução do número de mensagens, o desempenho da simulação tende a aumentar, e esse aumento ocorre com maior magnitude nas plataformas que utilizam meio de comunicação de baixo desempenho e configuração com menor número de processos lógicos. Pode-se concluir que **o aumento do número de tarefas em um sistema do tipo fechado possibilita o aumento do desempenho da simulação distribuída e o grau de variação do desempenho ocorre em função do número de processos adotados para a representação do modelo e a plataforma utilizada**.

7.2.3.8 Comentários – Número de Tarefas em Sistemas do Tipo Fechado

As variações no número de tarefas escalonadas para execução em modelos do tipo fechado provocam modificações no número de mensagens geradas pelos processos e conseqüentemente, alterações no *speedup* obtido para o modelo, para todas as plataformas consideradas. Essas alterações resultam em maior variação do desempenho para as plataformas que utilizam meio de comunicação de baixo desempenho, como *Ethernet* e *Token Ring*.

7.3 Simulação Distribuída sem Sincronismo

Apesar dos benefícios que a simulação distribuída pode oferecer, a execução da simulação com a utilização desse paradigma exige uma sincronização entre os processos paralelos para garantir o relacionamento de causa e efeito e a execução correta da simulação. Essa sincronização introduz uma sobrecarga que pode degradar o desempenho da simulação distribuída, em função de diferentes fatores, como as características do modelo e da plataforma utilizados.

Uma alternativa que vem sendo estudada visando melhorar o desempenho da simulação distribuída é a introdução de tolerância na execução da simulação, ou seja, a simulação é executada ignorando-se o sincronismo e permitindo a ocorrência de erros de causa e efeito [Martini et al 1997, Rao et al. 1998, Thondugulam et al. 1999].

A principal motivação para essa nova abordagem é a possibilidade de redução no tempo de execução, permitindo assim a obtenção de maior desempenho para a simulação distribuída. A redução da comunicação pode implicar em melhor

desempenho principalmente em plataformas com meio de comunicação de baixo desempenho, como observado no capítulo 6. Contudo, a execução da simulação sem o sincronismo imposto pelos protocolos introduz erros nos resultados obtidos, o que vem incentivando o estudo de técnicas estatísticas para a recuperação desses erros [Thondugulam et al. 1999].

Para verificar o comportamento da simulação distribuída sem o sincronismo do protocolo *CMB* foi utilizado o Modelo Hipotético 4, descrito no capítulo 5.

7.3.1 Modelo Hipotético 4

A simulação do Modelo Hipotético 4, com a configuração apresentada na figura 5.19, foi executada considerando diferentes valores para o tempo entre chegada de tarefas no sistema, o tempo de serviço de um recurso e a taxa de probabilidade no ponto de decisão do sistema.

A tabela 7.16 apresenta o *speedup* obtido para cada caso, considerado a execução da simulação distribuída conservativa e a execução da simulação sem sincronismo na plataforma *SP2/PVMe*. Os parâmetros utilizados para cada caso são descritos na tabela (tarec1 indica o tempo médio de chegada de tarefas no sistema; tsrec1 indica o tempo médio de serviço para o recurso REC1; tsrec2 indica o tempo médio de serviço para o recurso REC2; tsrec3 indica o tempo médio de serviço para o recurso REC3 e R indica a probabilidade de uma tarefa solicitar a utilização do recurso REC1).

Tabela 7.16 – *Speedup* – Modelo Hipotético 4.

Parâmetros	Modelo Hipotético 4 (1) (*CMB*)	Modelo Hipotético 4 (1) (sem sincronismo)
Caso 1 (tarec1 = 50.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 15.0 R = 30%)	1,20	1,27
Caso 2 (tarec1 = 50.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 15.0 R = 70%)	1,09	1,13
Caso 3 (tarec1 = 50.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 60.0 R = 30%)	1,26	1,34
Caso 4 (tarec1 = 50.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 60.0 R = 70%)	1,23	1,31
Caso 5 (tarec1 = 50.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 15.0 R = 0%)	1,28	1,37
Caso 6 (tarec1 = 5.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 15.0 R = 30%)	1,13	1,15
Caso 7 (tarec1 = 5.0 tsrec1 = 10.0 tsrec2 = 15.0 tsrec3 = 15.0 R = 70%)	1,03	1,06

Como pode ser verificado na tabela 7.16, a execução da simulação distribuída sem a utilização do sincronismo na plataforma *SP2/PVMe* possibilitou a obtenção de melhor desempenho, em decorrência da inexistência do sincronismo e da redução da comunicação entre os processos (apesar da comunicação não representar um fator determinante para o desempenho da simulação nessa plataforma, a ausência das mensagens nulas contribui para um aumento do desempenho) . Porém, a falta de sincronismo introduz erro nos dados estatísticos obtidos.

A tabela 7.17 apresenta os resultados obtidos para a utilização, comprimento médio de fila e período médio ocupado dos recursos representados no modelo, nas versões seqüencial, distribuída (*CMB*) e sem sincronismo, para alguns dos casos apresentados na tabela 7.16.

Tabela 7.17 – Estatísticas - Modelo Hipotético 4.

	Seqüencial			*CMB*			Sem sincronismo		
	Util.	Fila	P.M.O.	Util.	Fila	P.M.O.	Util.	Fila	P.M.O.
Caso 1									
Rec1	0,2868	0,1138	10,0	0,2867	0,1134	10,0	0,2901	0,1123	10,1
Rec2	0,4309	0,3170	15,0	0,4302	0,3233	15,0	0,4321	0,3176	15,0
Rec3	0,2959	0,2115	14,8	0,2959	0,3203	15,0	1,0	0,0567	14,9
Caso 2									
Rec1	0,6642	1,3110	9,9	0,6638	1,2855	9,9	0,6710	1,3187	10,0
Rec2	0,9994	650,58	15,0	0,9997	648,85	15,0	0,9996	648,24	15,0
Rec3	0,2921	0,1189	14,8	0,2899	0,1179	14,9	1,0	0,0046	15,0
Caso 3									
Rec1	0,2868	0,1138	10,0	0,2869	0,1140	10,1	0,2910	0,1132	10,0
Rec2	0,4291	0,3279	15,0	0,4289	0,3277	14,9	0,4320	0,3176	15,0
Rec3	0,9978	4735,1	59,6	0,9976	4734,9	59,8	1,0	0,0475	58,9
Caso 4									
Rec1	0,6675	1,3304	10,0	0,6679	1,3308	10,0	0,6710	1,3690	10,0
Rec2	0,9986	273,2	14,9	0,9980	275,0	15,0	0,9979	280,3	15,0
Rec3	0,9996	4426,1	59,4	0,9993	4430,1	60,2	1,0	0,0017	59,8

Considerando a execução da simulação distribuída sem sincronismo pode-se observar que as estatísticas dos recursos REC1 e REC2 para a utilização e o comprimento médio de fila sofrem pequenas alterações em relação aos dados obtidos com a execução da simulação seqüencial. Já para o recurso REC3 as alterações são significativas, tanto para a utilização do recurso quanto para o comprimento médio de fila. Em algumas situações, como nos casos 3 e 4, a estatística obtida para a utilização do recurso REC3 é bem próxima dos valores obtidos com a execução seqüencial e distribuída da simulação. Em todos os casos considerados as estatísticas para o período médio ocupado não sofrem alterações, quando comparadas as diferentes versões para a execução da simulação.

Observa-se que essa abordagem pode representar uma alternativa para a execução da simulação distribuída, principalmente com a utilização de plataformas com meio de comunicação de baixo desempenho. Nessas plataformas a redução da

comunicação pode evitar a sobrecarga de comunicação e possibilitar melhor desempenho da simulação (como verificado na seção 6.4, a comunicação pode degradar o desempenho da simulação em função das características do modelo). Um aspecto importante a ser considerado é o estudo de métodos estatísticos para a correção dos erros introduzidos devido à ausência da sincronização.

7.4 Considerações Finais

Neste capítulo foram apresentadas as análises realizadas sobre os dados obtidos com a simulação distribuída dos modelos descritos no capítulo 5. A abordagem empregada possibilitou analisar o comportamento da simulação distribuída conservativa perante as características do modelo e a configuração dos processos lógicos.

Em função das análises realizadas considerando diferentes tipos de modelos de sistemas computacionais, foram estabelecidas diretrizes para auxiliar o usuário no processo de desenvolvimento de uma simulação distribuída conservativa. Com o auxílio dessas diretrizes é possível minimizar a necessidade do usuário em conhecer as particularidades e o comportamento do protocolo de sincronização e possibilitar que esse usuário desenvolva programas de simulação com um bom desempenho em uma determinada plataforma. Essas diretrizes são sintetizadas na tabela 7.18 (em função dos resultados obtidos para a plataforma *SP2/PVM*, a tabela apresenta as diretrizes considerando plataformas paralelas e distribuídas). Essas diretrizes são explicadas após a tabela 7.18.

Tabela 7.18 – Diretrizes para Desenvolvimento da Simulação Distribuída Conservativa.

	Arquitetura Paralela	**Arquitetura Distribuída (*cluster*)**
Partição do Modelo em Processos Lógicos		
Comunicação	Pequeno impacto no desempenho; O balanceamento tem prioridade.	Grande impacto no desempenho; Minimizar comunicação.
Balanceamento	Fundamental para o desempenho; Maximizar balanceamento em detrimento da comunicação.	Minimizar comunicação em detrimento do balanceamento.
Fluxo de tarefas	Partição do modelo deve ser realizada em função do fluxo de tarefas no sistema.	Partição do modelo deve ser realizada em função do fluxo de tarefas no sistema.
Modelo com *feedback*	A forma de partição deve agrupar no mesmo processo os recursos envolvidos no *feedback*.	A forma de partição deve agrupar no mesmo processo os recursos envolvidos no *feedback*.
Granulosidade e Número de Processos Lógicos		
Número de Processos	Melhor desempenho com maior número de processos (granulosidade fina).	Melhor desempenho com menor número de processos (granulosidade grossa).
Granulosidade	Melhor desempenho para granulosidade fina.	Melhor desempenho para granulosidade grossa.
Modelo com Estrutura em Série	Reduzir o número de processos; Agrupar os recursos que constituem *feedback* em um mesmo processo.	Reduzir o número de processos; Agrupar os recursos que constituem *feedback* em um mesmo processo.
Alteração nas Características do Modelo		
Probabilidade nos pontos de decisão	Queda do desempenho com sistema ocioso ou com grande número de tarefas.	Aumento do desempenho com grande número de tarefas no sistema; Pode ocorrer aumento do desempenho com sistema ocioso.
Número de tarefas em sistema do tipo fechado	Menor desempenho com poucas tarefas no sistema; Pequeno aumento no desempenho com o aumento do número de tarefas.	Menor desempenho com poucas tarefas no sistema; Aumento no desempenho com o aumento do número de tarefas.
Alteração de parâmetros dos recursos – aumento do tempo de chegada de tarefas	Pequena redução do desempenho em função do aumento do número de mensagens.	Redução significativa do desempenho em função do aumento do número de mensagens.
Alteração de parâmetros dos recursos – aumento do tempo de serviço	Redução do desempenho se as alterações resultam em saturação do sistema.	Aumento do desempenho se as alterações resultam em saturação do sistema.

- **Partição dos Processos Lógicos**: para determinar a configuração dos processos na simulação distribuída deve-se considerar três pontos principais: as características da plataforma (*hardware/software*) utilizada, as características do modelo (presença de *feedback*) e o relacionamento entre os recursos que compõem o modelo:
  - Em plataformas onde o meio de comunicação não apresenta alto desempenho, a partição do modelo deve ser feita de forma a minimizar a comunicação entre os processos. Com plataformas que utilizam meios de comunicação de alto desempenho (e com ambientes de *software* que permitam explorar essa característica), o ponto fundamental é maximizar o balanceamento, em detrimento da comunicação;
  - Para modelos com *feedback* a partição do mesmo deve ser efetuada de forma a agrupar em um mesmo processo os recursos que constituem esse *feedback*, mesmo que isso implique em aumento da comunicação;
  - A partição dos processos deve evitar "quebrar" o fluxo das tarefas, agrupando em um mesmo processo recursos que não se comunicam diretamente;
- **Granulosidade e Número de Processos Lógicos**: a granulosidade adotada para os processos está diretamente ligada às características da plataforma (*hardware/software*) utilizada, e também com as características do modelo:
  - A granulosidade fina (maior número de processos) é indicada para plataforma com alto desempenho na comunicação, como o *SP2/PVMe*.

Para arquiteturas com meio de comunicação que não apresenta um bom desempenho, como os sistemas distribuídos, a granulosidade grossa (menor número de processos) deve ser utilizada na configuração dos processos;

- Para modelos com estrutura em série a partição deve ser feita de forma a minimizar a comunicação, ou seja, reduzindo o número de processos lógicos;

- **Probabilidade nos Pontos de Decisão**: alterações nas probabilidades nos pontos de decisão exercem pouca influência no desempenho da simulação quando o sistema está em equilíbrio. As maiores alterações no desempenho ocorrem com o sistema ocioso (sobrecarga das mensagens nulas) ou saturado (altas taxas de comunicação e tempo de execução da simulação seqüencial) para a plataforma *SP2/PVMe*. Para as plataformas distribuídas ocorrem grandes alterações com o sistema saturado e ocioso;

- **Alteração no Número de Tarefas em Sistemas do Tipo Fechado**: o aumento do número de tarefas permite o aumento do desempenho da simulação, em função da redução do número de mensagens geradas pelos processos. O aumento no desempenho é influenciado também pelo número de processos lógicos que constituem a simulação;

- **Alteração no Tempo de Chegada de Clientes no Sistema:** o aumento no tempo de chegada de clientes no sistema provoca uma redução significativa do desempenho para plataformas com meio de comunicação de baixo desempenho;

- **Alteração nos Tempos de Serviço dos Recursos**: quando as alterações nos tempos de serviço resultam em saturação do sistema, provocam uma pequena queda no desempenho na plataforma paralela. Nas plataformas distribuídas essas alterações implicam em aumento do desempenho em situações de saturação.

Além de estabelecer as condições que possibilitam guiar o usuário para o desenvolvimento de aplicações distribuídas com um bom desempenho, as diretrizes permitem agilizar o processo de desenvolvimento, uma vez que o usuário não necessita executar diferentes testes para determinar qual situação melhor se adapta à simulação.

Este capítulo apresentou também uma análise sobre o comportamento da simulação distribuída sem sincronismo, um novo paradigma que vem sendo estudado com o objetivo de reduzir o tempo de execução da simulação distribuída. Apesar da falta de sincronismo implicar em erros nas análises estatísticas, essa nova abordagem pode consistir em uma alternativa viável para o desenvolvimento da simulação distribuída, principalmente para as classes de plataformas nas quais a comunicação representa o fator determinante para o desempenho.

O capítulo 8 apresenta as conclusões obtidas, as contribuições e as propostas para continuidade do trabalho.

CAPÍTULO 8

CONCLUSÕES, CONTRIBUIÇÕES E PROPOSTA DE TRABALHOS FUTUROS

Este capítulo apresenta uma visão geral sobre o trabalho desenvolvido, as conclusões e contribuições resultantes e as propostas para a continuidade do trabalho.

8.1 Introdução

O paradigma da simulação distribuída constitui uma alternativa para o desenvolvimento de aplicações complexas e que demandam grande poder computacional. Contudo, devido à necessidade de protocolos para impor a sincronização e garantir a execução correta dos eventos da simulação distribuída, o desenvolvimento de um programa de simulação utilizando esse paradigma não constitui uma atividade trivial. Além de conhecimentos sobre aspectos relativos à computação paralela, o usuário deve estar familiarizado com as características do

modelo a ser desenvolvido, do protocolo de sincronização e da plataforma utilizada para que seja alcançado um bom desempenho em uma simulação distribuída.

Desta forma, as análises realizadas neste trabalho têm por objetivo fornecer aos usuários as informações sobre o comportamento do protocolo conservativo *CMB*, determinando os fatores que podem influenciar no desempenho da simulação nas diferentes plataformas consideradas. Com base nessas análises, é estabelecido um conjunto de diretrizes, as quais visam auxiliar o usuário no processo de desenvolvimento de uma simulação distribuída conservativa.

Com o auxílio dessas diretrizes e a disponibilidade da extensão funcional *ParSMPL* pretende-se auxiliar o usuário a desenvolver aplicações eficientes utilizando o paradigma da simulação distribuída, sem exigir desse usuário o conhecimento das características e particularidades do protocolo de sincronização utilizado e com conhecimento apenas de conceitos básicos de programação paralela.

A revisão apresentada nos capítulos 2 e 3 baseia-se em uma vasta bibliografia enfatizando os conceitos fundamentais dentro do paradigma de computação paralela e simulação distribuída. O capítulo 2 faz uma revisão sobre os conceitos básicos de computação paralela, classificação das arquiteturas e paradigmas de programação. O paradigma de programação utilizando a troca de mensagens é detalhado, apresentando exemplos dos ambientes mais utilizados. As principais características dos ambientes *PVM* e *PVMe*, empregados no desenvolvimento deste trabalho, são destacadas.

O capítulo 3 descreve os conceitos básicos sobre simulação e os tipos de protocolos utilizados para a sincronização dos processos paralelos na simulação distribuída. Neste capítulo são apresentadas a classificação desses protocolos e as

principais características das classes de protocolos conservativos e otimistas. O protocolo *CMB*, objetivo de estudo deste trabalho, é apresentado em detalhes. Esse capítulo apresenta também o estado da arte na simulação distribuída, destacando as principais linguagens e bibliotecas utilizadas atualmente, além de relacionar um grande número de protocolos que vêm sendo desenvolvidos, incluindo novos protocolos e alterações e otimizações dos protocolos existentes.

Diferentes técnicas para avaliação de desempenho foram analisadas. A escolha das técnicas de aferição, mais especificamente a construção de protótipos para executar a simulação distribuída, para a avaliação de desempenho do protocolo *CMB* é justificada pela precisão dos resultados obtidos quando comparados com aqueles obtidos por meio de técnicas de modelagem. Desta forma, as extensões funcionais *SMPLX* e *ParSMPL* foram desenvolvidas para servirem de base para a execução da simulação distribuída. O capítulo 4 descreve as principais características dessas ferramentas. Para a realização das análises foi definido um conjunto de modelos representativos de uma grande variedade de sistemas computacionais, descrito no capítulo 5.

Com a disponibilidade das extensões funcionais e a definição do conjunto de modelos foi possível executar a simulação seqüencial e distribuída dos mesmos e realizar a coleta das informações fundamentais para a análise de desempenho da simulação distribuída. Os capítulos 6 e 7 apresentam a avaliação do protocolo *CMB* considerando a influência de diferentes fatores no desempenho da simulação distribuída, de forma a estabelecer o conjunto de diretrizes que permitem auxiliar o processo de desenvolvimento de aplicações eficientes.

8.2 Conclusões

Por intermédio das análises dos dados obtidos com a simulação de diversos modelos e o desenvolvimento das extensões *SMPLX* e *ParSMPL*, pode-se chegar a diversas conclusões que são descritas a seguir.

1. Desempenho da Simulação Distribuída nas Diferentes Plataformas

As implementações da simulação distribuída para os modelos desenvolvidos neste trabalho apresentaram melhor desempenho quando executadas na plataforma *SP2* utilizando o ambiente de passagem de mensagens *PVMe*. As características da plataforma, como a eficiência de comunicação e de processamento, contribuíram para o desempenho obtido.

Nas demais plataformas a baixa eficiência da comunicação e/ou do processamento foram responsáveis pela degradação no desempenho da simulação distribuída. Na plataforma *SP2/PVM* o desempenho foi afetado pela baixa eficiência da comunicação (pelo fato do *PVM* executar sobre outro ambiente de passagem de mensagem) e a eficiência dos elementos de processamento. A plataforma *Token Ring* apresenta eficiência de processamento, porém a comunicação impõe restrições ao desempenho da simulação.

A plataforma *Ethernet* apresenta baixa eficiência de comunicação e processamento, fato que justifica o desempenho obtido para a simulação, superior ao desempenho obtido nas plataformas *Token Ring* e *SP2/PVM*.

Por meio das análises apresentadas no capítulo 6 foi possível observar que o *speedup* da simulação distribuída é determinado pelo o tempo gasto com a

comunicação entre os processos paralelos e a potência computacional dos elementos de processamento utilizados. Desta forma, foi possível estabelecer uma classificação para as plataformas utilizadas, em função da eficiência de comunicação e de processamento de cada uma dessas plataformas. Com base nessa classificação, e considerando o desempenho obtido para a simulação distribuída dos modelos analisados, foi realizado um estudo que possibilita prever a tendência do comportamento da simulação em função das características da plataforma utilizada.

2. Utilização de Ambientes Distribuídos

Apesar do desempenho insatisfatório obtido com a execução da simulação nas plataformas distribuídas, a utilização dessas plataformas com o uso de ambientes de passagem de mensagens constitui uma alternativa viável para o desenvolvimento de aplicações com o paradigma da simulação distribuída. É possível explorar as características do *hardware* existente, desenvolvendo aplicações com um custo relativamente baixo, e muitas vezes com *software* familiar à equipe de desenvolvimento. Um ponto forte na utilização desses ambientes é a possibilidade de migrar as aplicações para diferentes plataformas sem a necessidade de alterações nos códigos fontes dessas aplicações.

3. Fatores Determinantes para o Desempenho da Simulação Distribuída

Características da Plataforma

A plataforma utilizada na execução de uma simulação distribuída apresenta uma influência marcante no desempenho obtido. Com a execução da simulação de

diferentes modelos, observa-se que o *speedup* obtido em cada plataforma tende a estar relacionado com uma determinada faixa de valores. Essas faixas podem ser definidas considerando as características da comunicação e dos elementos de processamento da plataforma em questão.

Na plataforma *SP2* as características do modelo e o sincronismo imposto pelo protocolo constituem os fatores determinantes para o desempenho da aplicação. Nessa plataforma a comunicação não representa um elemento que afeta de forma decisiva o desempenho da simulação.

Nas demais plataformas consideradas no trabalho tanto os aspectos relativos à comunicação quanto às características do modelo e o sincronismo imposto pelo protocolo constituem fatores que podem afetar de forma significativa o desempenho da aplicação.

As variações de *speedup* que ocorrem nas diferentes faixas de valores são determinadas pelas características de cada modelo, que podem ser responsáveis por degradar o desempenho da simulação.

Mensagens Nulas

A sobrecarga gerada pelas mensagens nulas, utilizadas no protocolo *CMB* para evitar o *deadlock* da simulação, nem sempre constitui um obstáculo para o desempenho da simulação distribuída. O grau de influência dessas mensagens no desempenho da simulação depende diretamente das características da plataforma utilizada e do modelo desenvolvido.

Na plataforma *SP2/PVMe*, que apresenta eficiência na comunicação, a sobrecarga das mensagens nulas não interfere no desempenho da simulação

distribuída. Nas plataformas com baixa eficiência de comunicação (*Token Ring, Ethernet e SP2/PVM*) a sobrecarga das mensagens nulas pode exercer influência no desempenho da simulação, de acordo com as características do modelo considerado.

Partição dos Processos e Características do Modelo

Diferentes fatores, como por exemplo os aspectos relacionados à partição e configuração dos processos lógicos e as características do modelo, exercem influência no desempenho da simulação distribuída. O grau de influência das características do modelo no desempenho é determinado pelas características de cada plataforma.

Considerando a configuração dos processos lógicos, na plataforma paralela deve-se procurar maximizar o balanceamento. Nas plataformas distribuídas a redução da comunicação é prioridade para o desempenho, em função das características do modelo. A forma de partição deve agrupar no mesmo processo lógico os recursos envolvidos em *feedback*, e deve ser realizada considerando o fluxo de tarefas no sistema, independente do tipo de plataforma utilizada. Modelos com estrutura em série obtêm melhor desempenho utilizando número reduzido de processos.

Na plataforma paralela obtém-se melhor desempenho utilizando processos lógicos com granulosidade fina. Já para as demais plataformas, a granulosidade grossa possibilita melhor desempenho para a simulação distribuída.

Os parâmetros do modelo, tais como as taxas de probabilidade nos pontos de decisão e os tempos de serviço dos recursos, podem implicar em saturação do sistema, reduzindo o desempenho na plataforma paralela e aumentando o desempenho nas plataformas distribuídas. Alterações nos tempos entre chegadas de tarefas podem implicar em redução do desempenho para todas as plataformas consideradas.

4. Viabilidade da Utilização de Extensões Funcionais e Ambientes de Passagem de Mensagens

O desenvolvimento deste trabalho demonstra que a utilização de extensão funcional de linguagens de programação tradicionais (como a linguagem C) no desenvolvimento de aplicações em simulação distribuída é viável, devido às vantagens que tal abordagem oferece, tal como a possibilidade de se utilizar uma linguagem de programação já conhecida e os recursos disponíveis nessa linguagem.

Além disso, a utilização das plataformas de portabilidade permite ao usuário o desenvolvimento de aplicações que podem ser facilmente migradas para diferentes tipos de plataformas, sem a necessidade de alteração no código fonte da aplicação.

5. Otimização do Desempenho da Simulação

O desempenho da simulação conservativa pode ser otimizado em função das particularidades do modelo simulado e da plataforma utilizada. Assim, atenção especial deve ser dada à forma de partição e configuração dos processos lógicos e aos parâmetros do modelo, pois esses fatores podem exercer maior ou menor influência no desempenho da simulação de acordo com a plataforma considerada.

6. Viabilidade do Paradigma da Simulação Distribuída Conservativa

O paradigma da simulação distribuída possibilita o desenvolvimento de aplicações eficientes, explorando os benefícios oferecidos pela computação paralela.

A análise minuciosa dos resultados obtidos demonstra que o desempenho da simulação distribuída é afetado diretamente pelas características da plataforma utilizada e pode ser agravado pelas características do modelo desenvolvido.

A implementação do protocolo *CMB* com o *ParSMPL* demonstra a viabilidade da utilização de extensões funcionais e ambientes de passagem de mensagens para o desenvolvimento da simulação distribuída conservativa. O processo de desenvolvimento de uma simulação distribuída exige do usuário o conhecimento não somente das características do protocolo, mas também das características do modelo e do paradigma de programação paralela. Esse processo complexo dificulta a utilização da simulação distribuída. Com a utilização da extensão *ParSMPL*, mesmo para um usuário com poucos conhecimentos em simulação distribuída, é possível desenvolver aplicações eficientes, uma vez que todas as características do protocolo e a comunicação entre os processos paralelos são implementadas de forma transparente ao usuário.

8.3 Contribuições

As principais contribuições deste trabalho são apresentadas a seguir.

1. Extensões Funcionais para Simulação

As extensões funcionais desenvolvidas neste trabalho constituem um conjunto de ferramentas flexíveis que possibilitam a avaliação de desempenho de sistemas computacionais. A extensão *SMPLX*, versão para *UNIX* da extensão funcional *SMPL*, oferece ao usuário a possibilidade de utilizar uma ferramenta com

características semelhantes a uma extensão já conhecida e também a possibilidade de migrar as aplicações desenvolvidas para uma grande variedade de plataformas.

O *ParSMPL* oferece ao usuário as primitivas necessárias para a implementação de programas de simulação distribuída utilizando o protocolo conservativo *CMB*. A transparência da comunicação e sincronização entre os processos e a possibilidade de migrar as aplicações para diferentes tipos de plataformas constituem as principais características dessa extensão.

Além disso, o processo de desenvolvimento de uma aplicação utilizando o *ParSMPL* possibilita que até mesmo usuários com pouca experiência em simulação distribuída e programação paralela consigam desenvolver suas aplicações. A estrutura dos processos lógicos pode ser utilizada como um gabarito, simplificando e agilizando o trabalho do usuário.

2. Metodologia para Avaliação de Desempenho

A análise de desempenho do protocolo conservativo *CMB* utilizando técnicas de aferição e o conjunto de modelos desenvolvidos constituem casos de testes que podem ser empregados na avaliação de diferentes protocolos, possibilitando uma comparação com os resultados obtidos por intermédio das análises efetuadas com o protocolo *CMB*.

3. Diretrizes Para o Desenvolvimento de uma Simulação Distribuída

Neste trabalho foi definido de um conjunto de diretrizes para orientar o usuário no processo de desenvolvimento da simulação distribuída com o protocolo

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

conservativo *CMB*. Com o auxílio dessas diretrizes é possível o desenvolvimento de aplicações eficientes, mesmo no caso de usuários com pouca experiência na utilização dos paradigmas de simulação distribuída e computação paralela.

4. Estudo do Comportamento da Simulação Distribuída

O trabalho apresentou um estudo que possibilita a previsão da tendência do comportamento da simulação distribuída em função das características arquiteturais da plataforma utilizada. Considerando as características de comunicação e de processamento dessas plataformas e o *speedup* da simulação distribuída, foi possível estabelecer um diagrama para analisar o comportamento da simulação. Utilizando esse diagrama e tendo conhecimento das características da plataforma utilizada, pode-se verificar a tendência do comportamento da simulação distribuída.

Desta forma, é possível para o usuário verificar a relação custo/benefício da utilização do paradigma distribuído e determinar se a utilização desse paradigma é vantajosa.

8.4 Propostas para Trabalhos Futuros

As alternativas apresentadas visam a continuidade deste trabalho.

1. Alterações na Extensão *ParSMPL*

A extensão *ParSMPL* pode ser alterada visando oferecer ao usuário opções na utilização de ambientes de passagem de mensagens e interfaces para agilizar o processo de desenvolvimento das aplicações:

- Migração do *ParSMPL* para outros ambientes de passagem de mensagens, como por exemplo o *MPI*, permitindo assim uma comparação do desempenho obtido, além de oferecer ao usuário maior flexibilidade no desenvolvimento das aplicações;
- Desenvolvimento de uma interface gráfica para auxiliar o usuário no processo de implementação da simulação distribuída com o *ParSMPL*;
- Criação de um ambiente para simulação distribuída que permita a geração automática do programa de simulação a partir da especificação fornecida pelo usuário;
- Alteração no *ParSMPL* para utilização de técnicas de modelagem mais flexíveis para a representação dos sistemas computacionais, como por exemplo as Redes de *Petri*.

2. Definição de Classes de Protocolos

O desenvolvimento de um núcleo com a implementação de diferentes tipos de protocolos, como, por exemplo, o protocolo conservativo *Conservative Time Windows*, que forneceria ao usuário a possibilidade de analisar qual protocolo oferece o melhor desempenho para sua aplicação.

A disponibilidade desse núcleo permitiria a análise do comportamento da simulação distribuída e a definição de classes de protocolos mais eficientes em função do tipo de sistema a ser simulado.

3. Utilização de Técnicas de Modelagem

A utilização das técnicas de aferição para a avaliação do protocolo *CMB* possibilitaram a obtenção de resultados precisos e a realização de uma análise detalhada sobre o comportamento do protocolo considerando diferentes modelos e plataformas computacionais. Porém, apesar da precisão dos dados obtidos, a realização das análises fica restrita às plataformas disponíveis para a execução da extensão funcional e a realização da coleta de dados.

Desta forma, o desenvolvimento de uma ferramenta que utiliza técnicas de modelagem, e a simulação para resolução do modelo, possibilitaria ao usuário maior flexibilidade e possibilidade de avaliar o comportamento do protocolo considerando diferentes tipos de plataformas. A utilização das técnicas de modelagem permitiria agilizar o processo de avaliação de desempenho do protocolo e não restringiria o trabalho do usuário às plataformas disponíveis para análise.

4. Estudo de Técnicas para Balanceamento Dinâmico de Carga

Desenvolver um estudo para verificar a necessidade do balanceamento dinâmico de carga na simulação distribuída e a viabilidade da inclusão dessas técnicas na extensão *ParSMPL*.

REFERÊNCIAS BIBLIOGRÁFICAS

AKYILDIZ, L. F.; CHEN, L.; DAS, S. R.; FUJIMOTO, R. M.; SERFOZO, R. F. Performance analysis of Time Warp with Limited Memory. In: PROCEEDINGS OF THE 1992 ACM SIGMETRICS CONFERENCE ON MEASUREMENT AND MODELING COMPUTER SYSTEMS, p.213-224, 1992.

ALMASI, G. S; GOTTLIEB, A. *Highly Parallel Computing*. 2a. ed., The Benjamin Cummings Publishing Company, Inc. 1994.

ALONSO, J. M; ARRUABARRENA, A.; BEIVIDE, R. Parallel Simulation of Message Routing Networks. In: PROCEEDINGS OF THE EUROMICRO WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING. IEEE Computer Press, 1995.

ALONSO, J. M.; ARRUABARRENA, A.; BEIVIDE, R. Conservative Parallel Simulation of a Message-passing Network: a Performance Study. In: PROCEEDINGS OF THE 1995 SUMMER COMPUTER SIMULATION CONFERENCE. The Society for Computer Simulation International, 1995.

ALONSO, J. M.; ARRUABARRENA, A.; BEIVIDE, A.; FORTES, J. A. B. An Evaluation of Implementations of the CMB Parallel Simulation Algorithm on Distributed Memory Multicomputers. *Journal of Systems Architecture*, n.44, p.519-545, 1998.

AMORIM, C. L.; BARBOSA, V. C.; FERNANDES, E. S. T. Uma Introdução à Computação Paralela e Distribuída. VI Escola de Computação, 1988.

ANDREWS, G. R., SCHNEIDER, F. B. Concepts and Notations for Concurrent Programming. *ACM Computing Survey*, v.15, n.1, p.3-43, 1983.

AYANI, R.; RAJAEI, H. Parallel Simulation Using Conservative Time Windows. In: PROCEEDINGS OF THE 1992 WINTER SIMULATION CONFERENCE, p.709-717. The Society for Computer Simulation International, 1992.

AYANI, R.; BERKMAN, B. Parallel Discrete Event Simulation on SIMD Computers. *Journal of Parallel and Distributed Computing*, v.18, p.501-508, 1993.

AYANI, R. Parallel Simulation. *Lecture Notes in Computer Science*, 1993.

AYANI, R.; RAJAEI, H. Parallel Simulation Based on Conservative Time Windows: a Performance Study. *Concurrency: Practice and Experience*, v.6, n.2, 1994.

BAGRODIA, R.; MEYER, R.; TAKAI, M.; CHEN, Y.; ZENG, X.; MARTIN, J.; SONG, H. Y. Parsec: a Parallel Simulation Environment for Complex Systems. *IEEE Computer*, v.31, n.10, p.77-85, 1998.

BALAKRISHNAN, V.; FREY, P.; ABU-GHAZALEH, N. B.; WILSEY, P. A. A Framework for Performance Analysis of Parallel Discrete Event Simulations. In: PROCEEDINGS OF THE 1997 WINTER SIMULATION CONFERENCE, p.429-436. The Society for Computer Simulation International, USA, 1997.

BLECH, R. A. *An Overview of Parallel Processing*. Slides. Parallel Computing with PVM Workshop, Nasa Lewis Research Center, 1994. (http://www/lerc.nasa.gov/Other_Groups/IFMD/2620/tutorialPP.html).

BRYANT, R. E. *Simulation of Packet Communications Architecture Computer Systems*. Relatório Técnico, MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

BRUSCHI, S. M.; ULSON, R. S.; SANTANA, R. H. C.; SANTANA, M. J. Simulation as a Tool for Computer Architecture Teaching. In: PROCEEDINGS OF THE 1999 SUMMER COMPUTER SIMULATION CONFERENCE. The Society for Computer Simulation International, USA, 1999..

CAI, W.; LETERTRE, E.; TURNER, S. J. Dag Consistent Parallel Simulation: a Predictable and Robust Conservative Algorithm. In: PROCEEDINGS OF THE 11TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p.178-181, 1997.

CHANDY, K. M.; MISRA, J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*. SE-5, n.5, p.440-452, 1979.

CHANDY, K. M.; MISRA, J. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *Communications of the ACM*, v.24, n.4, p.198-205, 1981.

CHANDY, K. M.; SHERMAN, R. The Conditional Event Approach Distributed Simulation. In: PROCEEDINGS OF THE SCS MULTICONFERENCE ON DISTRIBUTED SIMULATION, v.21, n.2, p.93-99. The Society for Computer Simulation International, 1989.

CAVITT, D. B.; OVERSTREET, C. M.; MALY, K. J. A Performance Analysis Model for Distributed Simulations. In: PROCEEDINGS OF WINTER SIMULATION CONFERENCE. The Society for Computer Simulation International, USA, 1996.

COTA, B. A.; SARGENT, R. G. A Framework for Automatic Lookahead Computation in Conservative Distributed Simulations. In: PROCEEDINGS OF THE SCS MULTICONFERENCE ON DISTRIBUTED SIMULATION, p. 56-59. The Society for Computer Simulation International, 1989.

DAS, R. S.; FUJIMOTO, R. M. A performance Study of the Cancellback Protocol for Time Warp. In: PROCEEDINGS OF THE 7TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p.135-142,1993.

DAS, S. R. Estimating the Cost of Throttled Execution in Time Warp. In: PROCEEDINGS OF THE 10TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p.186-189, 1996.

DEELMAN, E.; SZYMANSKI, B. K. Continuously Monitored Global Virtual Time in Parallel Discrete Event Simulation. In: PROCEEDINGS FOR THE INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, 1997.

DICKENS, P. M., NICOL, D. M. The Impact of Adding Aggressiveness to a Non-Aggressive Windowing Protocol. In: PROCEEDINGS OF THE 1993 WINTER SIMULATION CONFERENCE. The Society for Computer Simulation International, USA, p.731-739 1993.

DILLON, E., SANTOS, C. G. D., GUYARD, J. Homogeneous and Heterogeneous Networks of Workstations: Message Passing Overhead. In: MPI DEVELOPERS CONFERENCE, University of Notre Dame, 1995.

D'SOUZA, L. M.; FAN, X.; WILSEY, P. A. Modifications to the pGVT Algorithm to Eliminate Acknowledgement Messages and Improve the GVT Broadcast Frequency. In: PROCEEDINGS OF THE WORLD CONGRESS ON SYSTEM SIMULATION, 1997.

DUNCAN, R. A Survey of Parallel Computer Architectures. *IEEE Computer*, p.5-15, 1990.

FERSCHA, A.; TRIPATHI, S. K. Parallel and Distributed Simulation of Discrete Event Systems. *Computer Simulation -Technical Report - 3336*, Dept. of Computer Science, University of Maryland, College Park, MD,1994.

FERSCHA, A. Adaptative Model Parallelism Exploration in Parallel Simulation. In: PROCEEDINGS OF EUROSIM'95, p.241-248,1995.

FERSCHA, A.; JOHNSON, J. A Testbed for Parallel Simulation Performance Prediction. In: PROCEEDINGS OF THE 1996 WINTER SIMULATION CONFERENCE, USA, 1996.

FOSTER, Y. *Designing and Bulding Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Publishing Company Inc., 1995.

FUJIMOTO, R. M. Performance Measurements of Distributed Simulation Strategies. *Transactions of the Society for Computer Simulation* (The Society for Computer Simulation International), v.6, n.2, p.89-132, 1989.

FUJIMOTO, R. M. Parallel Discrete Event Simulation. *Communications of the ACM*, v.33, n.10, p.31-53, 1990.

FUJIMOTO, R.; NICOL, D. State of The Art in Parallel Simulation. In: PROCEEDINGS OF THE 1992 WINTER SIMULATION CONFERENCE, p.246-254, 1992.

GALLUSCIO, A. P.; DOUGLASS, J. T.; MALLOY, B. A.; TURNER, A. J. A Comparison of Two Methods for Advancing Time in Parallel Discrete Event Simulation. In: PROCEEDINGS OF THE 1995 WINTER SIMULATION CONFERENCE, p.650-667. The Society for Computer Simulation International, USA, 1995.

GEIST, A. Advanced Programming in PVM. In: PROCEEDINGS OF THE 3TH EUROPEAN PVM CONFERENCE, Lecture Notes in Computer Science, n. 1156, p. 1-6,1996.

GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAN, V. *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.

GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAN, V. *PVM 3 User's Guide and Reference Manual*. Oak Ridge National Laboratory. 1994.

GOGF, T. J.; MOTT, J. A. *Improve Quality & Productivity With Simulation*. JMI Consulting Group, 1995.

JEFFERSON, D. R. Virtual Time. *ACM Transaction Programming Language and System*, v.7, n.3, p.404-425, 1985.

HAMILTON JR., J.; NASH, D. A.; POOCH, U. W. *Distributed Simulation*. CRC Press, 1997.

HOEGER, H. R.; JONES, D. W. Integrating Concurrent and Conservative Distributed Discrete-Event Simulators. *Simulation*, v.5, n.67, p.303-314, 1996.

HWANG, K.; BRIGGS, F. A. *Computer Architecture and Parallel Processing*. McGraw-Hill International Editions, 1984.

HWANG, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill International Editions, 1993.

HWANG, K.; XU, Z. *Scalable Parallel Computing – Technology, Architecture, Programming*. McGraw-Hill Companies, 1998.

IBM Corporation. *IBM PVMe for AIX – User's Guide and Subroutine Reference*. Version 2, Release 2, 1996.

KACPRZAK, M. Distributed Simulation of Sets of Asynchronous Automata. In: PROCEEDINGS 13TH EUROPEAN SIMULATION MULTICONFERENCE, p. 215-219. The Society for Computer Simulation International, Polonia, 1999.

KALANTERY, N. Tentative Time Warp. In: PROCEEDINGS OF THE 3TH INTERNATIONAL EURO-PAR CONFERENCE, p.458-467, 1997.

LIM, C.; LOW, Y.; GOI, B. A Conservative Parallel Simulation Algorithm for Entity-Oriented Modeling. In: PROCEEDINGS OF THE 5TH INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING, 1998.

LIM, C.; LOW, Y.; CAI, W.; HSU, W.; JUANG, S.; TURNER, S. J. An Empirical Comparison for Runtime Systems for Conservative Parallel Simulation. In: PROCEEDINGS OF THE 2TH WORKSHOP ON RUNTIME SYSTEMS FOR PARALLEL PROGRAMMING, 1998.

LIN, Y.; LAZOWSKA, E. D.. A Time-Division Algorithm for Parallel Simulation. *ACM Transactions on Modeling and Computer Simulations*, v.1, n.1, p.73-83, 1991.

LOW, Y.; LIM, C.; CAI, W.; HUANG, S.; HSU, W.; JAIN, S.; TURNER, S. J. Survey of Languages and Runtime Libraries for Parallel Discrete Event Simulation. *Simulation*, v.72, n.3, 1999.

LUBACHEVSKY, B. D. Efficient Distributed Event-driven Simulations of Multiple-loop Networks. *Communications of the ACM*, v.32, p.111-125, 1989.

MACDOUGALL, M. H. *Simulating Computing Systems - Techniques and Tools.* The MIT Press, 1987.

MADISETI, V. K.; HARDAKER, D. A.; FUJIMOTO, R. M. et al. The MIMDIX Operating System for Parallel Simulation. In: PROCEEDINGS OF THE SCS MULTICONFERENCE ON PARALLEL AND DISTRIBUTED SIMULATION, v.24, n.3, p.65-74. The Society for Computer Simulation International, 1992.

MARTINI, P.; RÜMEKASTEN, M.; TOLLE, J. Tolerant Synchronization for Distributed Simulations of Interconnected Computer Networks. In: PROCEEDINGS OF THE 11TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p. 138-141, 1997.

MARYANSKY, F.J. *Digital Computer Simulation.* Hayden Book Company, 1980

MASCARENHAS, E.; KNOP, F.; PASQUINI, R.; REGO, V. Checkpoint and Recovery Methods in the ParaSol Simulation System. In: PROCEEDINGS OF THE 1997 WINTER SIMULATION CONFERENCE, USA, 1997.

MCBRYAN, O. A. An Overview of Message Passing Environments. *Parallel Computing*, v.20, n.4, p.416-444, 1994.

MEYER, R. A.; BAGRODIA, R. L. Path Lookahead: a Data Flow View of PDES Models. In: PROCEEDINGS OF THE 13TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION (PADS'99), 1999.

MISRA, J. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, v.18, n.1, p.39-65, 1986.

MORSELLI JR., J. C. M. *Simulação Distribuída em uma Rede de Transputers Utilizando o Método CMB.* Dissertação (Mestrado). ICMSC-Universidade de São Paulo,1995.

NAROSKA, E.; SCHWIEGELSHOHN, U. Conservative Parallel Simulation of a Large Number of Processes. *Simulation*, v.72, n.3, p. 150-162, 1999.

NAVAUX, P. O. A. Introdução ao Processamento Paralelo. *Revista Brasileira de Computação*, v.5, n.2, p.31-43, 1989.

NICOL, D. High Performance Parallelized Discrete-Event Simulation of Stochastic Queueing Networks. In: PROCEEDINGS OF 1988 WINTER SIMULATION CONFERENCE. The Society for Computer Simulation International, 1988.

NICOL, D.; FUJIMOTO, R. *Parallel Simulation Today.* Technical Report, College of William and Mary. 1993.

NICOL, D. M. The Cost of Conservative Synchronization in Parallel Discrete Event Simulations. *Journal of the Association for Computing Machinery*, v.30, n.2, p.304-333, 1993.

NICOL, D. M. Principles of Conservative Parallel Simulation. In: PROCEEDINGS OF THE 1996 WINTER SIMULATION CONFERENCE, p.128-135, USA, 1996.

ORLANDI, R. C. G. S. *Ferramentas para Análise de Desempenho de Sistemas Computacionais Distribuídos*. Dissertação (Mestrado). ICMSC-Universidade de São Paulo, 1995.

OVEREINDER, B.; HERTZBERGER, B.; SLOOT, P. Parallel Discret Event Simulation. (httpp://www.fwi.uva.nl/research/pwrs/papers/archieve/Overeinder91), 1991.

PETERSON, G. D.; CHAMBERLAIN, R. D. Exploiting Lookahead In Synchronous Parallel Simulation. In: PROCEEDINGS OF THE 1993 WINTER SIMULATION CONFERENCE, p.706-712. The Society for Computer Simulation International, USA, 1993.

PEYTCHEV, E. T.; BARGIELA, A. Parallel Simulation of City Traffic Flows Using "PADSIM". In: PROCEEDINGS OF MODELING AND SIMULATION CONFERENCE ESM'95, p.330-334, 1995.

PHAM, C. D.; FDIDA, F. Length-Based Blocking and Local Estimations in Distributed Simulation: A Case Study. In: PROCEEDINGS OF THE 29TH SYMPOSIUM ON SIMULATION, p.97-106, 1996.

PHAM, C. D.; ESSMEYER, J.; FDIDA, S. Simulation of a Routing Algorithm using Distributed Simulation Techniques. In: PROCEEDINGS OF THE INTERNATIONAL EUROPAR'97 CONFERENCE, p.1001-1008, 1997. Lecture Notes in Computer Science, v. 1300.

PHAM, C. D.; BRUNST, H.; FDIDA, S. Conservative Simulation of Load-Balanced Routing in a Large ATM Network Model. In: PROCEEDINGS OF THE 12TH PARALLEL AND DISTRIBUTED SIMULATION WORKSHOP, 1998.

POPLAWSKI, A. L.; NICOL, D. M. NOPS: a Conservative Parallel Simulation Engine for TeD. In: PROCEEDINGS OF THE 12TH WORKSHOP ON PARALLEL AND DISTRIBUTED SYSTEMS, p.180-187, 1998.

PORRAS, J.; HARA, V.; HARJU, J.; IKONEN, J. Improving the Performance of the Chandy-Misra Parallel Simulation Algorithm in a Distributed Workstation Environment. In: PROCEEDINGS OF THE 1997 SUMMER COMPUTER SIMULATION CONFERENCE, p.657-662. The Society for Computer Simulation International, USA, 1997.

PORRAS, J.; HARA, V.; HARJU, J.; IKONEN, J. Applyng a Modified Chandy-Misra Algorithm to the Distributed Simulation of a Cellular Network. In: PROCEEDINGS OF THE 12TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p.188-195, 1998.

PRAKASH, A.; SUBRAMARIAN, R. Optimistic Distributed Simulation. In: PROCEEDINGS OF THE 24TH ANNUAL SIMULATION SYMPOSIUM, p.123-132, 1991.

PREISS, B. R.; LOUCKS, W. M. Prediction and Lookahead in Distributed Simulation. Technical Report, University of Waterloo, 1989.

PREISS, B. R.; LOUCKS, W. M. The Impact of Lookahead on the Performance of Conservative Distributed Simulation. In: PROCEEDINGS OF 1990 EUROPEAN MULTICONFERENCE – SIMULATION METHODOLOGIES, LANGUAGES AND ARCHITECTURES, p.204-209, Nuremberg, FRG, June 1990.

PREISS, B.; LOUCKS, W. M.; MACINTYRE, I. D.; FIELD, J. A. Null Message Cancellation in Conservative Distributed Simulation. In: PROCEEDINGS OF THE 1991 WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p. 33-38, 1991.

PREISS, B. R.; WAN, K. W. C. The Parsimony Project: A Distributed Simulation Testbed in Java. In: PROCEEDINGS OF 1999 INTERNATIONAL CONFERENCE ON WEB-BASED MODELLING & SIMULATION, p. 89-94, 1999.

QUEALY, A. An introduction to Message Passing. Slides, NASA Lewis Research Center, (http://www.lerc.nasa.gov/Other_Groups/IFMD/2620/tutorialPP.html), 1994.

QUINN, M. J. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill series in Supercomputing and Artificial Intelligence. McGraw-Hill Inc., 1987.

RADHARAMANAM, R. ABU-GHAZALEH, N. On-line Configuration of a Time Warp Parallel Discrete Event Simulation. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, ICPP-1998, p.28-35, 1998.

RAO, D. M.; THONDUGULAM, N. V.; RADHAKRISHANAN, R.; WILSEY, P. A. Unsynchronized Parallel Discrete Event Simulation. In: PROCEEDINGS OF THE 1998 WINTER SIMULATION CONFERENCE, p. 1563-1570, USA, 1998.

REIHER, P. L.; JEFFERSON, D. Limitation of the Optimism in the Time Warp Operating System. In: PROCEEDINGS OF THE 1989 WINTER SIMULATION CONFERENCE, p.765-770, 1989.

SÁNCHEZ, V.; BAUTISTA, A.; TIRADO, F. Deblocking Event Algorithm: a New Approach to Conservative Parallel Discrete Event Simulation. In: PROCEEDINGS OF THE 4TH EUROMICRO WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING, 1996. (http://pc-bilbo.dia.ucm.es/papers/simul/)

SANTANA, M.J. *An Advanced Filestore Architecture for a Multiple-LAN Distributed Computing System*. Tese (Doutorado), University of Southampton, England, 1990.

SANTANA, M. J. SANTANA, R.H. C.; SPOLON, RENATA; SPOLON, ROBERTA. A Graphical User-Interface for an Automatic Simulation System. In: PROCEEDINGS OF THE 1996 SUMMER COMPUTER SIMULATION

CONFERECE, p. 139-143. The Society for Computer Simulation International, USA, 1996.

SANTANA, M. J.; SANTANA, R. H. C.; SPOLON, ROBERTA; SPOLON, RENATA. Automatic Generation of Discrete-System Simulation Programs. In: PROCEEDINGS OF THE 1996 SUMMER COMPUTER SIMULATION CONFERENCE, p. 133-138. The Society for Computer Simulation International, USA, 1996.

SANTANA, R. H. C.; SANTANA, M. J.; SOUZA, P. S. L.; SOUZA, M. A.; PIERKARSKI, A. E. T. *Computação Paralela.* NOTAS DE AULA. ICMC-USP, 1997.

SENGER, L. J. *Avaliação de Desempenho do PVM-W95.* Dissertação (Mestrado), ICMC-Universidade de São Paulo, 1997.

SCHOF, S. Efficient Data Structures for Time Warp Simulation Queues. *Journal of Systems Architecture*, n. 44, p. 497-517, 1998.

SOARES, L. F. G. *Modelagem e Simulação Discreta de Sistemas.* Editora Campus Ltda., 1992.

SOLIMAN, H. M. On the Selection of the State Saving Strategy In Time Warp Parallel Simulations. *Transactions of The Society for Computer Simulation International*, v. 16, n. 1, p. 32-36, 1998.

SOUZA, M. A. *Avaliação das Rotinas de Comunicação Ponto-a-Ponto do MPI.* Dissertação (Mestrado), ICMC-Universidade de São Paulo, 1996.

SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C.. Distributed Simulation, Time Warp and is Variants: Taxonomy and Performance Evaluation Issues. In: PROCEEDINGS 13TH EUROPEAN SIMULATION MULTICONFERENCE, p. 220-227. The Society for Computer Simulation International, Polonia, 1999.

SRINIVASAN, S.; REYNOLDS JR., P. NPSI Adaptative Synchronization Algorithms for PDES. IN: PROCEEDINGS OF THE 1995 WINTER SIMULATION CONFERENCE, p.658-665. The Society for Computer Simulation International, USA, 1995.

SUNDERAN, V. S.; GEIST, G. A.; DONGARRA, J.; MANCHEK, R. The PVM Concurrent System: Evolution, Experiences, and trends. *Parallel Computing*, v.20, n.4, p.531-545, 1994.

SWAIN, J. J. Simulation Survey: Tools for Process Understanding and Improvement. *ORMS Today*, 1995.

TANIR, O; SULEYMAN, S. Defining Requirements for a Standard Simulation Environment. *IEEE Computer*, v.27, n.2, p.28-34, 1994.

TANENBAUM, A. S. *Distributed Operating Systems.* Prentice-Hall, Inc. 1995.

TAY, S. C.; TEO, Y. M., AYANI, R. Speculative Parallel Simulation with an Adaptive Throttle Scheme. In: PROCEEDINGS OF THE 11TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, p.116-123, 1997.

THONDUGULAM, N. V.; RAO, D. M.; RADHAKNSHNAN, R.; WILSEY, P. A. Relaxing Causal Constraints in PDES. In: PROCEEDINGS OF THE 13TH

INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM, p. 696-700, 1999.

TURNER, S. J. Models of Computation for Parallel Discrete Event Simulation. *Journal of Systems Architecture*, n. 44, p. 395-409, 1998.

ULSON, R. S.; SANTANA, R. H. C.; SANTANA, M. J. A Distributed Simulation Environment for Computing Systems Performance Evaluation. Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'97). In: PROCEEDINGS OF THE 1997 SUMMER COMPUTER SIMULATION CONFERENCE, p.85-90. The Society for Computer Simulation International, 1997.

ULSON, R. S.; MORSELLI Jr., J. C. M.; SANTANA, R. H. C; SANTANA, M. J. Conservative Distributed Simulation On Portability Platforms: The CMB Protocol Behavior. In: PROCEEDINGS OF 1999 APPLIED MODELLING SIMULATION, p. 495-500. IASTED, Austrália, 1999.

VARGHESE, G.; CHAMBERLAIN, R.; WEIHL, W. E. The Pessimism behind Optimistic Simulation. In: PROCEEDINGS OF THE 8TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 1994.

VRIES, R. C. Reducing Null Messages in Misra's Distributed Discrete Event Simulation Method. *IEEE Transactions on Software Engineering*, v.16, n.1, 1990.

WILSEY, P. A. Feedback Control in Time Warp Synchronized Parallel Simulators. In: PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON DISTRIBUTED INTERACTIVE SIMULATION AND REAL TIME APPLICATIONS, p.31-38, 1997.

XIAO, Z.; UNGER, B.; SIMMONDS, R. Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation. In: PROCEEDINGS OF THE 13TH WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 1999.

ZALUSKA, E. J. Research Lines in Distributed Computing Systems and Concurrent Computation. In: ANAIS DO WORKSHOP EM PROGRAMAÇÃO CONCORRENTE, SISTEMAS DISTRIBUÍDOS E ENGENHARIA DE SOFTWARE, ICMC-USP, p.132-155, 1991.

APÊNDICE A

ANÁLISES ESTATÍSTICAS

A.1 Introdução

A inferência estatística tem como objetivo tirar conclusões sobre parâmetros populacionais a partir de análise dos dados amostrais. Alguns pontos importantes são o tamanho amostral e o modo de selecionar a amostra, a natureza da inferência desejada e a precisão das conclusões. Existem dois tipos de inferências, a estimação dos parâmetros, que pode ser realizada pela determinação de intervalos de confiança, e o teste de hipóteses. A seção a seguir demonstra a realização do teste de hipóteses utilizado no trabalho.

A.2 Teste de Hipóteses para Amostras Pequenas

O teste de hipóteses realizado tem por objetivo verificar se as diferenças observadas para os tempos de execução dos modelos de simulação analisados são estatisticamente significativas.

Foram definidas hipóteses para a comparação dos dados obtidos, e a partir da aceitação ou rejeição dessas hipóteses, pode-se concluir sobre a validade de uma determinada afirmação. H_0 é chamada hipótese de nulidade, e H_1 é a hipótese alternativa. Para provar qual hipótese é correta, tenta-se provar a hipótese nulidade. Se não é possível provar a hipótese nulidade, a hipótese alternativa é considerada verdadeira (para manter uniformidade o resultado final é dado em termos da hipótese H_0, ou seja, aceitar ou rejeitar H_0).

Considerando-se que as amostras n_1 e n_2 (relativas às populações[1] 1 e 2, respectivamente) são pequenas, ambas as populações são normais e as variâncias das populações 1 e 2 são distintas, isto é, $\delta_1^2 \neq \delta_2^2$ (nesse caso, $n_1 \neq n_2$), a estatística para os testes das hipótese relacionadas anteriormente é dada pelo teste apresentado na

[1] Neste trabalho a população 1 será indicada por *Tdist* e a população 2, por *Tseq*.

figura A.1, utilizando a distribuição *t* de Student (as amostras das populações 1 e 2 são independentes, ou seja, as médias dos dois tratamentos não são relacionadas entre si). As hipóteses elaboradas para a realização dos testes são:

- **Hipótese A**: a simulação distribuída apresenta um tempo de execução menor em relação à simulação seqüencial.
- **Hipótese B**: a simulação distribuída apresenta um tempo de execução maior em relação à simulação seqüencial.

Teste Estatístico ($\delta^2_{Tpar} \neq \delta^2_{Tseq}$, $n_{Tpar} \neq n_{Tseq}$)

$$t = \frac{\left(\overline{y_{Tpar}} - \overline{y_{Tseq}}\right) - D_0}{\sqrt{\frac{s^2_{Tpar}}{n_{Tpar}} - \frac{s^2_{Tseq}}{n_{Tseq}}}}$$

Graus de liberdade:

$$v = \frac{\left(s^2_{Tpar}/n_{Tpar} + s^2_{Tseq}/n_{Tseq}\right)}{\left[\frac{\left(s^2_{Tpar}/n_{Tpar}\right)^2}{n_{Tpar}-1} + \frac{\left(s^2_{Tseq}/n_{Teq}\right)^2}{n_{Tseq}-1}\right]}$$

	A) Tdist > Tseq	***B) Tdist < Tseq***
Hipóteses:	H_{A0}: $\mu_{Tdist} = \mu_{Tseq}$ H_{A1}: $\mu_{Tdist} > \mu_{Tseq}$	H_{B0}: $\mu_{Tdist} = \mu_{Tseq}$ H_{B1}: $\mu_{Tdist} < \mu_{Tseq}$
Região de rejeição:	$t > t_\alpha$	$t < -t_\alpha$

Onde:

- $\overline{y_{Tpar}}, \overline{y_{Tseq}}$: são as médias amostrais
- s^2_{Tpar}, s^2_{Tseq} : indicam as variâncias amostrais
- n_{Tdist}, n_{Tseq}: indicam os tamanhos da amostras
- *Tdist*: indica amostras de execução da simulação distribuída
- *Tseq*: indica amostras de execução da simulação seqüencial

Figura A.1 – Teste Estatístico para os Modelos de Simulação.

Nas tabelas A.1, A.2, A.3 e A.4 são analisados estatisticamente os dados obtidos para o Modelo do Sistema Computacional Simplificado (figura 5.7), utilizando as hipóteses apresentadas na figura A.1. Para cada dupla de valores médios comparados, são calculados os valores de ***t***. Se a diferença estatística é verificada, no campo relacionado ao **Resultado** é colocado a palavra *rejeita*, significando que a hipótese alternativa foi aceita e existe diferença estatística entre os valores medidos. Caso contrário, no campo **Resultado** é colocado o valor *aceita*.

Tabela A.1 – Análise Estatística – Sistema Computacional Simplificado (*SP2/PVMe*).

Sistema Computacional Simplificado Plataforma: ***SP2/PVMe***		
Tempo_chegada	**Estatística**	
50.0	Hipótese	A
	Resultado	Rejeita H_{A0}
100.0	**Estatística**	
	Hipótese	A
	Resultado	Rejeita H_{A0}
500.0	**Estatística**	
	Hipótese	A
	Resultado	Rejeita H_{A0}

Tabela A.2 – Análise Estatística – Sistema Computacional Simplificado (*Token Ring*).

Sistema Computacional Simplificado Plataforma: **SP2/Token Ring**		
Tempo_chegada	**Estatística**	
50.0	Hipótese	B
	Resultado	Rejeita H_{B0}
100.0	**Estatística**	
	Hipótese	B
	Resultado	Rejeita H_{B0}
500.0	**Estatística**	
	Hipótese	B
	Resultado	Rejeita H_{B0}

Tabela A.3 – Análise Estatística – Sistema Computacional Simplificado (*SP2/PVM*).

Sistema Computacional Simplificado Plataforma: ***SP2/PVM***		
Tempo_chegada	**Estatística**	
50.0	Hipótese	B
	Resultado	Rejeita H_{B0}
100.0	**Estatística**	
	Hipótese	B
	Resultado	Rejeita H_{B0}
500.0	**Estatística**	
	Hipótese	B
	Resultado	Rejeita H_{B0}

Tabela A.4 – Análise Estatística – Sistema Computacional Simplificado (*Ethernet*).

Sistema Computacional Simplificado Plataforma: ***Ethernet***		
Tempo_chegada	**Estatística**	
50.0	Hipótese	B
	Resultado	Rejeita H_{B0}
100.0	**Estatística**	
	Hipótese	B
	Resultado	Rejeita H_{B0}
500.0	**Estatística**	
	Hipótese	B
	Resultado	Rejeita H_{B0}

Bibliografia Utilizada

Achcar, J. A.; Rodrigues, J. Introdução à Estatística para Ciências e Tecnologia. Notas do ICMC-USP. 1995

Costa Neto, P. L. O. Estatística. Edgard Blûcher Ltda, 1997.

Mendenhall, W.; Sincich, T. Statistic for Engineering and the Sciences. 1992.