EVANDRO RAPHALOSKI

# ESTUDO SOBRE A VARIAÇÃO DOS PARÂMETROS DO *TREE LOAD BALANCING ALGORITHM*

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Mestre em Engenharia.

**Área de Concentração:**
Engenharia Elétrica

**Orientadora:**
Professora Doutora
Maria Stela Veludo de Paiva

São Carlos
2006

Dedico este trabalho à memória dos meus pais, exemplos de vida pelo interminável apoio, incentivo e espírito de luta.

# AGRADECIMENTOS

À Professora Doutora Maria Stela Veludo de Paiva, pela orientação, discussões e atenção neste trabalho.

À Libânio Carlos de Souza e a Délcio Prizon da empresa SMAR Equipamentos Industriais Ltda, por compreenderem a relevância de um Mestrado e pelos conhecimentos transmitidos de vida pessoal e profissional, contribuindo para minha formação.

Ao Professor Doutor Rodrigo Fernandes de Mello, pelos valiosos esclarecimentos e discussões sobre o tema.

À minha irmã Maria Regina de Pila Raphaloski pelo contínuo apoio e suporte nos principais momentos de minha vida.

Ao companheiro e amigo Professor Doutor Marcelo Barros de Almeida pelo suporte, valiosos esclarecimentos e discussões no decorrer do trabalho e revisão final desta dissertação.

À Elena Luzia Palloni Gonçalves pela revisão das citações, chamadas e formas de apresentação das referências, equações, tabelas e figuras apresentadas na dissertação.

Aos meus colegas Anderson José de Almeida, Carlos Eduardo de Oliveira Chierici e Cleison Pereira de Melo, pelo suporte e auxílio no desenvolvimento do simulador.

Aos companheiros de pesquisa e desenvolvimento, principalmente, àqueles que compartilham seus conhecimentos no Codeproject, Codeguru e Sourceforge, para o aprendizado e aprimoramento de terceiros.

A todos os meus colegas de trabalho, pelo apoio e incentivo diário.

E a todos que de certa forma contribuíram para a conclusão desta dissertação.

*“Precisamos lembrar que não somos os únicos que estamos diante de um problema quase insolúvel. Da mesma maneira que uma pipa só consegue levantar vôo quando é colocada contra o vento, mesmo o pior de nossos problemas serve para nos elevar a um degrau mais alto”.*

**Dr. Rudolph Brasch**

# RESUMO

RAPHALOSKI, E. (2006). **Estudo sobre a variação dos parâmetros do *Tree Load Balancing Algorithm***. 174p. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2006.

Em um sistema localmente distribuído, quando diversas tarefas são aleatoriamente submetidas aos computadores de uma determinada rede, onde alguns ficam sobrecarregados e outros permaneçam ociosos, algoritmos de balanceamento de carga podem ser utilizados para homogeneizar e otimizar a alocação de recursos e, conseqüentemente, aumentar o desempenho computacional. Com a evolução dos sistemas distribuídos, houve a necessidade de aprimoramento desses algoritmos, a fim de suportarem sistemas distribuídos altamente escaláveis e possibilitarem o gerenciamento de ambientes heterogêneos. Visando suprir essas necessidades, recentemente foi proposto um algoritmo de balanceamento de carga denominado *Tree Load Balancing Algorithm* (TLBA). Para avaliar esse algoritmo foram desenvolvidos um simulador e um protótipo. Com as simulações foram comprovados seus benefícios para ambientes distribuídos heterogêneos altamente escaláveis e com o protótipo esses resultados foram validados. Este trabalho apresenta a implementação de um novo simulador, que possibilitou um estudo mais abrangente dos parâmetros de balanceamento de carga do TLBA, com base em amostras estatisticamente geradas e uma maior fidelidade em suas políticas de balanceamento. Suas novas características são simulação em tempo real, visualização da árvore lógica de acordo com as capacidades computacionais relativas, resultados em tabelas, gráficos gerados pelo simulador e estudos aplicados aos diferentes tipos de escalonamento e sistemas.

**Palavras-chave:** Parâmetros de balanceamento de carga. Alto desempenho. Simuladores. Técnicas de avaliação de desempenho.

# ABSTRACT

RAPHALOSKI, E. (2006). **Study about the variation of the Tree Load Balancing Algorithm parameters**. 174p. M.Sc. Dissertation – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2006.

In locally distributed systems, when plenty of tasks are randomically submitted to network computers where certain computers may be heavily loaded, while others lightly loaded, load balancing algorithms may be used on to homogenize and optimize the resources allocation, and hence improve the computer performance. Later on, the distributed systems evolution made necessary the upgrading of these algorithms in order to support the highly scalable distributed systems and handle with the management of the heterogeneous environments. To supply these needs, a new load balancing algorithm called Tree Load Balancing Algorithm (TLBA) has been recently proposed. The evaluation of this algorithm has been done by a simulator and a prototype which were developed for this purpose. The simulations have proved its benefits in highly scalable heterogeneous distributed environments, and with the prototype the results were validated. This work presents the implementation of a new simulator which allows detailed studies of the TLBA parameters, based on the statistically generated samples, and a higher fidelity on the implementation of its load balancing policies. Its new features are the real time simulations, visualization of the logical tree according to computers' relative capacities, results in tables, graphics generated by the simulator and studies applied to the different types of scheduling and systems.

**Keywords:** Load balancing parameters. High performance. Simulators. Performance evaluation techniques.

# LISTA DE FIGURAS

**Figura 2.1** – Um sistema distribuído sem balanceamento de carga ..... 7
**Figura 2.2 –** Classificação hierárquica da composição dos algoritmos de escalonamento ..... 10
**Figura 2.3** – Classificação Hierárquica Proposta por Casavant e Kuhl..... 11
**Figura 2.4 –** Escalonamento Centralizado ..... 15
**Figura 2.5** – Escalonamento descrito por Zhou e Ferrari..... 18
**Figura 2.6** – Política de Informação do TLBA, onde ocorre a propagação do índice de carga dos computadores nas "folhas" da árvore até chegar a raiz. ..... 27

**Figura 3.1** – Distribuição das Categorias de Computadores..... 39
**Figura 3.2** – Classificação de Flynn. ..... 41
**Figura 3.3** – Arquitetura SISD..... 41
**Figura 3.4** – Arquitetura MISD ..... 42
**Figura 3.5** – Arquitetura SIMD ..... 43
**Figura 3.6** – Arquitetura MIMD..... 43
**Figura 3.7** – Classificação da arquitetura MIMD quanto ao acoplamento ..... 44
**Figura 3.8** – UMA – Tempos de acesso uniformes (p0→m0 = p0→m1 = p0→m2)..... 45
**Figura 3.9** – NUMA – Tempo de acesso p0→m1 maior que p0→m0 ..... 46
**Figura 3.10** – Esquema de Alta Disponibilidade ..... 49
**Figura 3.11** – Exemplo de cluster para balanceamento de cargas ..... 50
**Figura 3.12** – Representação de NOW/COW ..... 53
**Figura 3.14** – Etapas do Processo de Modelagem ..... 61
**Figura 3.15** – Etapas para o Desenvolvimento de uma Simulação ..... 71

**Figura 4.1** – Fluxograma da Política de Informação do TLBA ..... 90
**Figura 4.2** – Fluxograma da Política de Localização do TLBA ..... 91
**Figura 4.3** – Fluxograma da Política de Transferência do TLBA ..... 93
**Figura 4.4** – Fluxograma da Política de Seleção do TLBA ..... 94
**Figura 4.5** – Fluxograma do Simulador..... 96

**Figura 5.1** – *Splash Window* do Simulador..... 108
**Figura 5.2** – Interface Gráfica inicial de apresentação do TLBASim..... 109
**Figura 5.3** – Quatro classes com as características do ambiente de simulação. ..... 110
**Figura 5.4** – Propriedades Individuais das quatro classes..... 110
**Figura 5.5** – Janela de *Log* do TLBASim ..... 112
**Figura 5.6** – Tabela com os Parâmetros de Entrada e Resultados ..... 113
**Figura 5.7** – Gráfico exemplo do tipo colunas..... 114
**Figura 5.8** – Gráfico exemplo do tipo linhas..... 114
**Figura 5.9** – Gráfico exemplo do tipo barras horizontais ..... 115
**Figura 5.10** – Propriedade Relacionadas aos Gráficos ..... 116
**Figura 5.11** – Janela de Apresentação do *Wizard*..... 117
**Figura 5.12** – Seleção do Tipo de Gráfico ..... 117
**Figura 5.13** – Passo 1: Seleção dos Dados..... 118
**Figura 5.14** – Passo 2: Seleção da Propriedades..... 119
**Figura 5.15** – Diagrama de árvores. ..... 120
**Figura 5.16** – Diagrama de árvores – duas iterações ..... 121
**Figura 5.17** – Janela para edição de textos..... 121
**Figura 5.18** – *Schema* para edição da Lista de Computadores em XML. ..... 122
**Figura 5.19** – Calculadora de fácil acesso e utilização no TLBASim..... 123
**Figura 5.20** – Variação do número de mensagens em função do número de processos ..... 127
**Figura 5.21** – Percentual do *switch* de rede utilizado em função do número de processos ..... 128

**Figura 5.22** – Tempo médio de resposta (segundos) dos processos em função do número de processos no submetidos ao sistema ..... 130
**Figura 5.23** – Número de mensagens por nível geradas em função da variação do número de níveis e computadores por nível para o TLBA V1.0 ..... 134
**Figura 5.24** – Número de mensagens por nível geradas em função da variação do número de níveis e computadores por nível para o TLBASim[1] ..... 135
**Figura 5.25** – Número de mensagens por nível geradas em função da variação do número de níveis e computadores por nível para o TLBASim[2] ..... 135
**Figura 5.26** – Número de mensagens geradas em função da variação do *Threshold* de transferência ..... 139
**Figura 5.27** – Número de mensagens por nível geradas em função da variação do *Threshold* de atualização de carga ..... 139
**Figura 5.28** – Número de migrações realizadas durante a variação dos *Thresholds* de transferência e atualização de carga. ..... 141
**Figura 5.29** – Número de mensagens por nível geradas em função da variação do número de máximo de computador por nível e na árvore ..... 143
**Figura 5.30** – Número de mensagens geradas por computadores sobrecarregados em função da variação do número de máximo de computadores por nível e na árvore ..... 144
**Figura 5.31** – Tempo médio de resposta (s) em função da variação do número de máximo de computador por nível e na árvore ..... 145
**Figura 5.32** – Número de mensagens de atualização geradas em função da variação do número de máximo de computador por nível e na árvore ..... 146

**Figura 5.33** – Número de mensagens de balanceamento de carga em função da variação do número de máximo de computador por nível e na árvore ..... 146
**Figura 5.34** – Número de mensagens geradas em função da variação do número de migrações permitidas..... 149
**Figura 5.35** – Tempo médio de resposta (s) em função da variação do número máximo de migrações permitidas ..... 149

# LISTA DE TABELAS

**Tabela 5.1** – Número de mensagens por nível e total dos simuladores em função da variação dos processos submetidos ao sistema ..... 125

**Tabela 5.2** – Percentual do *switch* utilizado (10 Mbits/s), em função da variação dos processos submetidos ao sistema ..... 127

**Tabela 5.3** – Tempo médio de resposta dos processos em função da variação do número de processos submetidos ao sistema..... 129

**Tabela 5.4** – Número de mensagens por nível geradas em função da variação do número de computadores por nível e do número de níveis do TLBA V1.0 ..... 132

**Tabela 5.5** – Número de mensagens por nível geradas em função da variação do número de computadores por nível e do número de níveis do TLBASim..... 132

**Tabela 5.6** – Número de mensagens por nível geradas em função da variação do número de computadores por nível e do número de níveis do TLBASim..... 133

**Tabela 5.7** – Número de mensagens geradas no sistema devido a variação do *Threshold* de viabilidade de transferência ..... 137

**Tabela 5.8** – Número de mensagens geradas no sistema devido a variação do *Threshold* de atualização de carga..... 138

**Tabela 5.9** – Número de migrações realizadas durante a variação dos *Thresholds* de transferência e atualização de carga .... 140

**Tabela 5.10** – Tempo médio de resposta (s) em função da variação dos *Thresholds* de transferência e atualização de carga .... 141

**Tabela 5.11** – Tempo médio de resposta (s), número de migrações realizadas e número de mensagens geradas no sistema devido à variação do número máximo de migrações permitidas ..... 148

**Tabela 5.12** – Tempo médio de resposta (s), variância e desvio padrão para ambientes homogêneos e heterogêneos..... 151

**Tabela 5.13** – Número de Mensagens geradas para sistemas homogêneos e heterogêneos ..... 152

# LISTA DE ABREVIATURAS E SIGLAS

ATM - *Asynchronous Transfer Mode*
CC-NUMA - *Cache Coherent Non-Uniform Memory Access*
COMA - *Cache-Only Memory Access*
COW - *Clusters of Workstations*
CPU - *Central Unit Processing*
CM - Cadeias de Markov
DE - Diagrama de Estados
E/S - Entradas e Saídas
FCFS - *First Come, First Served*
FIFO - *First In, First Out*
FTP - *File Transfer Protocol*
GPL - *General Public License*
GNU - GNU é um acrônimo recursivo de *GNU's not UNIX*
HA - *High Availability*
HPC - *High Performance Computing*
HTTP - *Hyper Text Transfer Protocol*
IP - *Internet Protocol*
LAN - *Local Area Network*
LCFS - *Last Come, First Served*
LBM - *Load Balance Manager*
LIM - *Load Information Manager*
MEM - Memória
MIMD - *Multiple Instruction Multiple Data*
MISD - *Multiple Instruction Single Data*
MPP - *Massively Parallel Processors*
NC-NUMA - *Non-Cache Non-Uniform Memory Access*
NCC-NUMA - *Non-Cache Coherent Non-Uniform Memory Access*

| | |
|---|---|
| NORMA | - *No Remote Memory Access* |
| NOW | - *Network Of Workstations* |
| NUMA | - *Non-Uniform Memory Access* |
| PE | - Processos Estocásticos |
| PROC | - Processador |
| REP | - Requisição de Execução de Processo |
| RF | - Redes de Filas |
| RP | - Redes de Petri |
| RR | - *Round Robin* |
| SC | - *Statecharts* |
| Sim | - Simulação |
| SIMD | - *Single Instruction Multiple Data* |
| SISD | - *Single Instruction Single Data* |
| SMP | - Sistema Múltiplo Processado |
| TCP | - *Transport Control Protocol* |
| TF | - Teoria de Filas |
| TLBA | - *Tree Load Balancing Algorithm* |
| TLBASim | - *Tree Load Balancing Algorithm Simulator* |
| UMA | - *Uniform Memory Access* |
| UC | - Unidade de Controle |
| XML | - *Extensible Markup Language* |
| XSD | - *Schema Definition Language* |

# LISTA DE SÍMBOLOS

$I_{comp-1}$ - Índice de carga anterior

$I_{comp}$ - Índice de carga

$\overline{I_{comp}}$ - Capacidade ociosa de um computador

$I_{comp-\alpha}$ - Somatório dos índices de carga de todos os computadores na sub-árvore do computador α

$\overline{I_{comp-\alpha}}$ - Complemento dos índices de carga de todos os computadores na sub-árvore do computador α

$\overline{I_{comp-médio-\alpha}}$ - Complemento médio dos índices de carga de toda a árvore do computador α

$L_{\alpha}$ - Limite superior do complemento médio dos índices de carga do computador α

$O_{cpu}$ - Ocupação percentual de CPU

$O_{mem}$ - Ocupação percentual de memória

$p_1, p_2 \ldots p_n$ - Processador número 1, 2 ... n

$m_1, m_2 \ldots m_n$ - Memória número 1, 2 ... n

$t$ - Tempos

V - Variação do índice de carga

$V_i$ - Variação inferior do índice de carga

$V_s$ - Variação superior do índice de carga

$w_1$ - Primeiro peso de multiplicação da $O_{cpu}$

$w_2$ - Segundo peso de multiplicação da $O_{mem}$

$\lambda_{i,p}$ - *Quantum* total de cada processo

$\mu_{i,p}$ - Tempo relativo do $\lambda_{i,p}$ que manteve a CPU ocupada

$\sigma_{i,p}$ - Quantidade de memória principal utilizada por um processo

$\theta_{i,p}$ - Quantidade total de memória principal

$\delta$ - *Threshold* de atualização

$\tau$ - Número total de computadores presentes na sub-árvore do computador α

$\delta t$ - *Threshold* de transferência

# SUMÁRIO


1 INTRODUÇÃO ..... 1

1.1 VISÃO GERAL ..... 1

1.2 MOTIVAÇÕES E CONTRIBUIÇÕES ..... 3

1.3 ORGANIZAÇÃO DO TRABALHO ..... 5

2 ALGORITMOS DE BALANCEAMENTO DE CARGAS ..... 7

2.1 INTRODUÇÃO ..... 7

2.2 TAXONOMIA DOS ALGORITMOS DE ESCALONAMENTO DE PROCESSOS ..... 11

2.3 PRINCIPAIS ALGORITMOS DE BALANCEAMENTO DE CARGA ..... 16

2.4 MODELO DE CARREGAMENTO DE CARGAS ..... 31

2.5 CONSIDERAÇÕES GERAIS ..... 32

3 AVALIAÇÃO DE DESEMPENHO E MODELAGEM DE SISTEMAS COMPUTACIONAIS ..... 35

3.1 INTRODUÇÃO ..... 35

3.2 CLUSTERS E A SUPERCOMPUTAÇÃO ..... 38

*3.2.1 Multiprocessadores e Multicomputadores ..... 40*

*3.2.2 Vantagens e Desvantagens ..... 51*

3.3 MODELAGEM DE SISTEMAS ..... 52

3.4 TÉCNICAS DE AVALIAÇÃO DE DESEMPENHO ..... 55

*3.4.1 Técnicas de Aferição ..... 57*

3.4.1.1 Coleta de Dados ..... 57

*3.4.1.2 Benchmarks ..... 59*

3.4.1.3 Construção de Protótipos ..... 59

*3.4.2 Técnicas de Modelagem ..... 60*

3.4.2.1 Técnicas para Especificação de Modelos ..... 62

3.4.2.1.1 Rede de Filas ..... 62

3.4.2.1.2 Redes de Petri ..... 63

3.4.2.1.3 Statecharts ..... 64

3.4.2.2 Soluções para o Modelo ..... 65

3.4.2.2.1 Modelagem Analítica ..... 65

3.4.2.2.2 Modelagem Baseada em Descrição ..... 67

3.4.2.2.3 Modelagem com Análise Estática ..... 67

3.4.2.2.4 Modelagem Híbrida ..... 68

3.4.2.2.5 Simulação ..... 68

3.5 PREDIÇÃO DE DESEMPENHO ..... 74

3.6 CONSIDERAÇÕES GERAIS ..... 75

4 ESTUDO SOBRE A VARIAÇÃO DOS PARÂMETROS DO *TREE LOAD BALANCING ALGORITHM* ..... 77
4.1 INTRODUÇÃO ..... 77
4.2 CARACTERÍSTICAS E LIMITAÇÕES DO TLBA ..... 78
4.3 PARÂMETROS DE BALANCEAMENTO DE CARGA DO TLBA ..... 79
*4.3.1 Índices de Carga da Memória e da CPU ..... 80*
*4.3.2 Threshold de Atualização de Carga ..... 83*
*4.3.3 Threshold para Analisar Viabilidade de Transferência ..... 84*
*4.3.4 Número de Níveis da Árvore Computacional ..... 86*
*4.3.5 Número de Computadores (nós) ..... 87*
*4.3.6 Ambientes Homogêneos e Heterogêneos ..... 87*
4.4 MÉTRICAS DE ANÁLISE E POLÍTICAS DE BALANCEAMENTO DO TLBA ..... 88
4.5 DESCRIÇÃO DO NOVO SIMULADOR ..... 97
*4.5.1 Objetivos da Simulação ..... 97*
*4.5.2 Diferenças entre os Simuladores ..... 98*
*4.5.3 Componentes do Simulador ..... 100*
*4.5.4 Lógica Implementada no Simulador ..... 103*
4.6 CONSIDERAÇÕES GERAIS ..... 106
5 RESULTADOS E DISCUSSÃO ..... 107
5.1 NOVO SIMULADOR E SEUS RECURSOS ..... 107
5.2 CASOS DE ESTUDO PARA ANÁLISE DO TLBA ..... 123
*5.2.1 Análise do TLBA V1.0 e o TLBASim em Função da Variação do Número de Processos Submetidos ao Sistema ..... 124*
*5.2.2 Análise da Variação do Número de Computadores por Nível e de Níveis da Árvore Computacional do TLBA V1.0 com o TLBASim ..... 131*
*5.2.3 Análise da Variação dos Thresholds de Viabilidade de Transferência e Atualização de Carga ..... 136*
*5.2.4 Análise da Variação de Computadores no Sistema ..... 142*
*5.2.5 Análise da Variação do Número de Migrações ..... 147*
*5.2.6 Análise de Sistemas Homogêneos e Heterogêneos ..... 150*
5.3 CONSIDERAÇÕES GERAIS ..... 153
6 CONCLUSÕES ..... 155
6.1 CONSIDERAÇÕES FINAIS ..... 155
6.2 TRABALHOS FUTUROS ..... 157
REFERÊNCIAS ..... 159

Capítulo

1

# 1 INTRODUÇÃO

## 1.1 Visão Geral

O estudo de métodos para avaliação de desempenho é uma área de pesquisa importante no desenvolvimento e análise de computadores. Por outro lado, o crescimento acelerado da tecnologia e o aumento da complexidade dos sistemas computacionais fazem da avaliação de desempenho uma questão não muito simples de ser resolvida.

O alto desempenho não depende exclusivamente da utilização de *hardwares* mais eficientes, mas inclui novas técnicas de processamento para solucionar um problema. Como exemplo dessas técnicas pode-se citar a programação paralela, técnicas de otimização, programação vetorial, entre outras.

O balanceamento de cargas e a otimização de seus parâmetros constituem um tema fundamental, principalmente, por influenciar diretamente no desempenho de um sistema.

Algoritmos de balanceamento de cargas podem ser aplicados em ambientes homogêneos, heterogêneos, distribuídos localmente e *grids*.

A heterogeneidade dos recursos existentes nas diversas máquinas de um sistema computacional leva à possibilidade de se ter uma grande variação na potência computacional de cada uma. Essa variação, aliada à possibilidade de utilização de diferentes arquiteturas, introduz dificuldades que precisam ser gerenciadas quando ocorre o balanceamento de cargas, visando a obtenção de um bom desempenho.

Os modelos com foco na avaliação de desempenho e os índices de desempenho tradicionalmente usados para sistemas homogêneos apresentam restrições quanto à aplicabilidade em sistemas heterogêneos. Os algoritmos de balanceamento de carga devem incorporar mecanismos eficazes para a gerência de recursos distintos, levando-se em consideração características de arquitetura, configuração e tempo (BRANCO, 2002).

Esse gerenciamento de recursos de uma determinada rede de computadores é possível através da implementação de políticas de balanceamento de cargas (KRUEGER e SINGHAL, 1992; SHIRAZI e HURSON, 1992; SHIVARATRI, SONG, CHOO e LEE, 1997).

Os resultados observados em termos do desempenho atingido na execução de uma dada classe de aplicações estão intimamente ligados à qualidade das informações utilizadas pelo mecanismo que implementa uma política de balanceamento de carga (BRANCO, 2002). A qualidade das informações, por sua vez, depende ainda da natureza, da quantidade e da validade das informações no instante em que são utilizadas pelo computador que decide aonde executar um dado processo (FEITELSON, 1996; FEITELSON e RUDOLPH, 1995; FEITELSON e JETTE, 1997).

Na aplicação de políticas de balanceamento de carga ou metodologias que visam à melhoria de um determinado sistema, podem-se destacar diversas técnicas para análise e avaliação de seu desempenho.

Essas técnicas fundamentam-se, principalmente, em duas abordagens: as técnicas de aferição, onde as informações requeridas por um estudo podem ser obtidas a partir do próprio sistema já existente e as técnicas de modelagem, onde se constrói um modelo representativo do sistema.

A escolha dentre os modelos de abordagens existentes depende do objetivo da avaliação, do sistema a ser avaliado e do nível de detalhe a ser considerado. Para a utilização de técnicas de aferição é necessário que o sistema a ser avaliado esteja pronto para que as informações possam ser

coletadas. Um exemplo de técnicas de aferição são os *benchmarks* utilizados, principalmente, na comparação de sistemas existentes (BRUSCHI, 2002).

As técnicas de modelagem baseiam-se na construção de um modelo que represente as características a serem analisadas no sistema real, de forma preditiva ou analítica. A solução desse modelo pode ser obtida através de soluções analíticas ou por simulação.

As soluções analíticas são atrativas devido à precisão dos resultados obtidos (BRUSCHI, 2002). Em geral, elas são complexas e de difícil construção, porém são atrativas devido a precisão de seus resultados.

Na simulação o sistema é abstraído em um modelo e a partir desse modelo, um programa de simulação é gerado. É uma técnica que pode ser utilizada na experimentação e avaliação de sistemas, de modo a prever as conseqüências causadas por mudanças de políticas, de condições ou métodos, sem a necessidade de implementá-las no sistema real.

## 1.2 Motivações e Contribuições

Uma das características principais para algoritmos de balanceamento de cargas é que sejam implantados de forma a não prejudicarem ou trazerem maior sobrecarga ao sistema já em execução. Além disso, o controle do tráfego de informações em uma rede deve ser o menor possível para que haja uma melhoria do desempenho e uma maior estabilidade do sistema.

A quantidade de informações que trafegam em uma determinada rede, quando em processo de balanceamento de cargas é determinado através do ajuste dos parâmetros de balanceamento de cargas.

Existem trabalhos em balanceamento dinâmico de carga que alteram os parâmetros dos algoritmos de balanceamento de carga em resposta às mudanças da carga do sistema em questão (CHENGJIANG e SANLI, 1994;

MELAB et al, 1996; XU e HWANG, 1993). Esse tipo de metodologia é denominado de adaptabilidade paramétrica.

Problemas de otimização constituem um conjunto diversificado e são caracterizados pela dificuldade inerente à solução. Dentre esses, a otimização do balanceamento de cargas através do ajuste de seus parâmetros têm recebido destacada atenção, devido à sua relevância no resultado final da distribuição das cargas e melhoria do desempenho do sistema.

No intuito de aplicar e colaborar com as pesquisas na área de balanceamento de cargas, este trabalho tem como objetivo o estudo de parâmetros de balanceamento de carga e a variação dos ambientes de aplicação – homogêneos ou heterogêneos – do *Tree Load Balancing Algorithm* (TLBA) proposto por Mello (2003).

Por meio dos resultados obtidos da aplicação dos casos (diferentes cargas de processos, número de computadores por nível, número de níveis etc) foi feito o estudo e a análise do comportamento deste algoritmo durante o processo de simulação.

Para esses estudos foi decidida a utilização da solução por simulação, devido à sua versatilidade (pode ser usada em diferentes ocasiões), flexibilidade (é adaptável a novas e diferentes situações) e baixo custo (com um mesmo programa pode-se simular diferentes situações) (PEGDEN, SHANNON e SADOWSKI, 1995).

O simulador desenvolvido neste trabalho foi utilizado para análise da variação dos parâmetros de balanceamento de carga e das capacidades computacionais, ampliando os estudos referentes ao TLBA e ao seu comportamento em diferentes configurações. Inicialmente, ele foi baseado no simulador original de Mello[1] (2005).

[1] Para informações adicionais sobre simuladores e novos projetos relacionados ao TLBA, visite o site: http://www.icmc.usp.br/~mello/proj.html.

A seguir, apresenta-se a organização deste trabalho com uma breve descrição dos temas que são abordados em cada capítulo.

## 1.3 Organização do Trabalho

O trabalho está estruturado segundo a seqüência de Capítulos relacionados a seguir.

- **Capítulo I:** Introdução ao Trabalho.

- **Capítulo II:** Abordagem geral dos principais trabalhos relacionados à área de balanceamento de carga.

- **Capítulo III:** Avaliação de Desempenho e Modelagem de Sistemas Computacionais, analisando-se os trabalhos e metodologias presentes na literatura, bem como suas métricas e estratégias de estudo.

- **Capítulo IV:** Nesse capítulo é abordado o estudo dos parâmetros de balanceamento de carga e da variação das capacidades computacionais de aplicação do algoritmo TLBA.

- **Capítulo V:** Apresentação dos Resultados e Discussão das simulações e variações dos parâmetros do TLBA.

- **Capítulo VI:** Apresentação das considerações finais do trabalho com as conclusões obtidas e proposta de trabalhos futuros.

- **Referências Bibliográficas:** Índice de Referências Bibliográficas utilizadas neste trabalho.

Capítulo

2

# 2 ALGORITMOS DE BALANCEAMENTO DE CARGAS

## 2.1 Introdução

Os algoritmos de balanceamento de cargas tentam distribuir uniformemente os recursos compartilhados, evitando a situação em que um elemento de processamento está sobrecarregado enquanto outro está com a carga de processamento leve (BRANCO, 2002). Na Figura 2.1 é apresentado um sistema distribuído sem balanceamento de carga (SHIVARATRI, KRUEGER e SINGHAL, 1992).

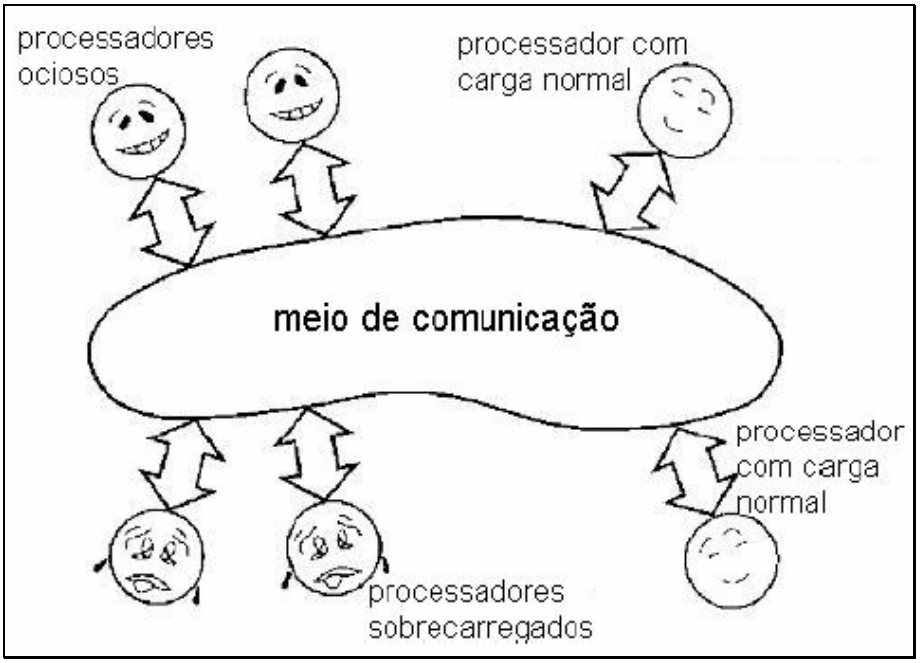


**Figura 2.1** – Um sistema distribuído sem balanceamento de carga
Fonte: Shivaratri, Krueger e Singhal (1992)

Os algoritmos de balanceamento de carga são aplicados em ambientes distribuídos, visando um melhor desempenho do sistema, a fim de distribuir de forma eqüitativa a carga de processos entre todos os computadores envolvidos na rede (EAGER, LAZOWSKA e ZAHORJAN, 1986a; LIVNY e MELMANN, 1982; MELLO, 2003; MELLO, TREVELIN e PAIVA, 2003b; SHIVARATRI, KRUEGER e SINGHAL, 1992; THEIMER e LANTZ, 1989; ZHOU e FERRARI, 1987).

Esses algoritmos são dotados de políticas de balanceamento e mecanismos que gerenciam e coordenam desde a localização dos computadores mais ociosos até a transferência do processo propriamente dita. Um computador é um servidor ou emissor quando oferece processos ao sistema, pois está carregado; um computador é receptor quando requer processos, pois está ocioso.

A ocupação dos recursos desses computadores é definida por índices de cargas que são calculados em função do comprimento da fila de processos de um computador (SHIVARATRI, KRUEGER e SINGHAL, 1992) ou em cálculos baseados na ocupação percentual da CPU e da memória (MELLO, 2003; MELLO, TREVELIN e PAIVA, 2003a).

Essas políticas são classificadas da seguinte forma:

1) **Política de Transferência**: determina se um computador é capaz de participar de um processo de transferência de tarefas ou não. Ela identifica computadores emissores e/ou receptores de processos, transferindo-os em seguida. São normalmente consideradas políticas de *Thresholds* (EAGER, LAZOWSKA e ZAHORJAN, 1986a).

2) **Política de Seleção**: uma vez viabilizada a transferência de um processo, ela tem como objetivo selecionar o processo com características de maior ocupação da CPU e tempo de execução

longo, que ocasionará a mínima sobrecarga no sistema em virtude da transferência e que tenha poucas dependências e chamadas relacionadas ao sistema local (SHIVARATRI, KRUEGER e SINGHAL, 1992);

3) **Política de Localização**: visa encontrar o melhor computador receptor de processos para o melhor emissor e vice-versa. O melhor emissor é definido como o computador mais sobrecarregado do sistema. O melhor receptor, como o computador mais ocioso do sistema (MELLO, 2003; SHIVARATRI, KRUEGER e SINGHAL, 1992);

4) **Política de Informação**: define quando, de onde e que tipo de informações relacionadas aos demais computadores da rede devem ser coletadas. Essas informações são utilizadas para o cálculo do índice de carga que define a ocupação de cada processador ou computador no ambiente. Existem três tipos de políticas de informação:

*a)* *Política Orientada à demanda*: uma máquina coleta o estado das outras máquinas somente quando ela se torna emissora ou receptora;

*b)* *Política Periódica*: as informações são coletadas de tempos em tempos;

*c)* *Política Orientada à Mudança de Estado*: as informações das máquinas são divulgadas conforme a mudança de seu estado.

Já os mecanismos empregados no balanceamento de cargas, por outro lado, podem ser divididos em:

1) **Mecanismo de métrica da carga**: indica o método utilizado para medir a carga em cada uma das máquinas;

2) **Mecanismo de comunicação da carga**: define o método através do qual será efetuada a comunicação das informações de carga entre as diversas máquinas; e

3) **Mecanismo de migração**: define o protocolo a ser utilizado quando ocorre migração de processos entre as máquinas.

Políticas de balanceamento de cargas consideradas como viáveis para utilização são aquelas que proporcionam um melhor desempenho das aplicações e não influenciam, ou pouco influenciam na carga do sistema.

Já bons algoritmos são aqueles que utilizam poucos recursos computacionais do ambiente e mesmo nessas condições conseguem alocar recursos de forma otimizada e homogênea oferecendo, conseqüentemente, um alto desempenho para as aplicações.

Funcionalmente e hierarquicamente falando, a atividade de algoritmos de escalonamento pode ser disposta como é apresentado na Figura 2.2 (BRANCO, 2002).

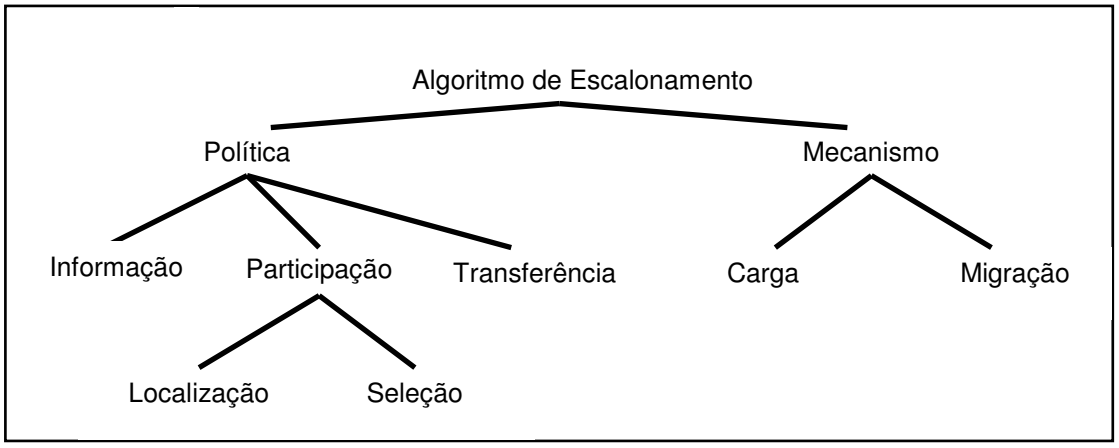


**Figura 2.2 –** Classificação hierárquica da composição dos algoritmos de escalonamento
Fonte: Branco (2002)

## 2.2 Taxonomia dos Algoritmos de Escalonamento de Processos

Para compreender melhor o termo balanceamento de cargas e outros utilizados para classificar algoritmos de escalonamento de processos foram propostas algumas taxonomias (CASAVANT e KUHL, 1988; LÜLING e MONIEN, 1993; LÜLING, MONIEN e RAMME, 1993; SHIRAZI, HURSON e KAVI, 1995; XU e LAU, 1997). Nenhuma dessas taxonomias abrange todas as características que um escalonador pode possuir, devido a isso existem características que são descritas à parte. Dentre as taxonomias estudadas, a proposta por Casavant e Kuhl (1988) é a mais aceita e completa, sendo apresentada na Figura 2.3.

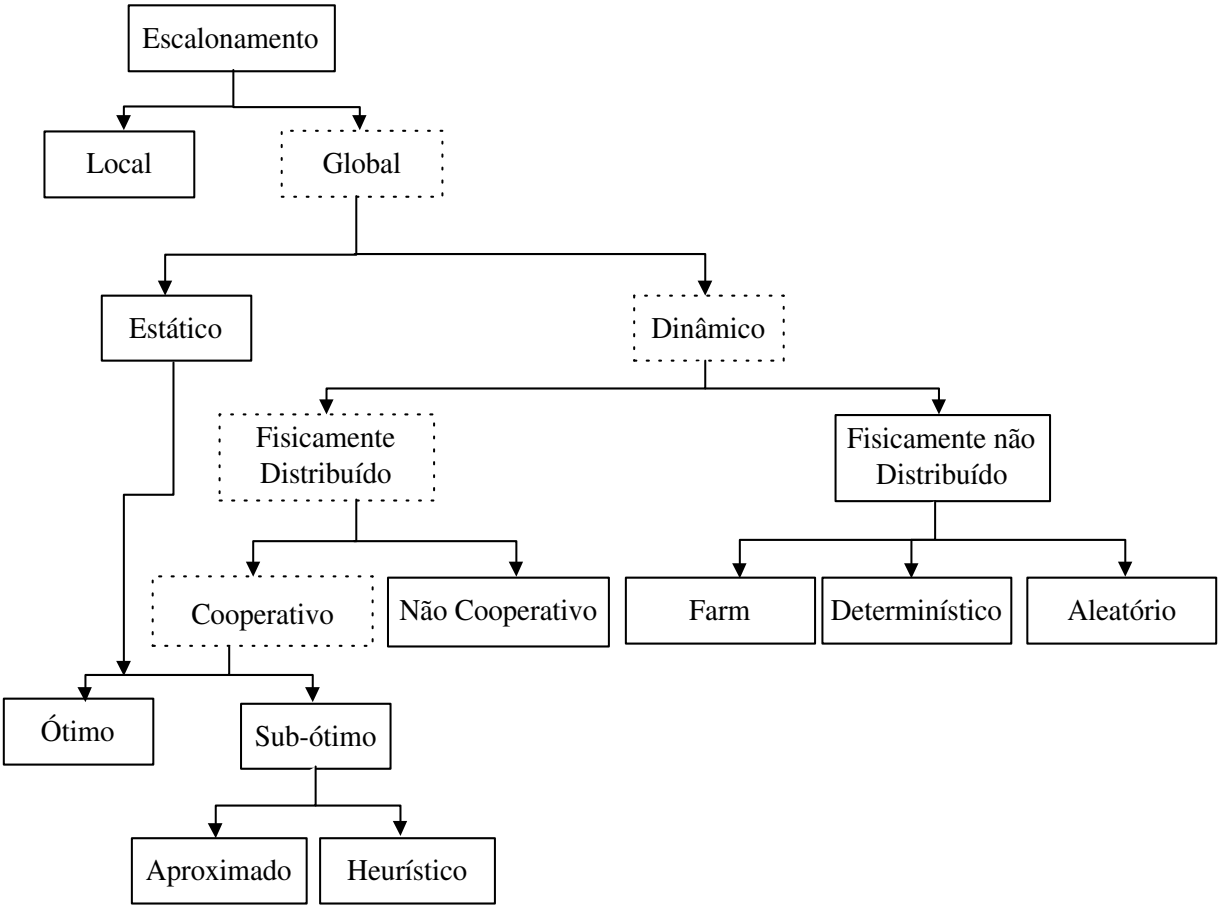


**Figura 2.3** – Classificação Hierárquica Proposta por Casavant e Kuhl
Fonte: Branco (2002)

Na taxonomia apresentada, o escalonamento local está relacionado com a ordenação de processos em um único processador. Esse trabalho tem como foco o escalonamento global, o qual consiste em decidir num ambiente com um número elevado de computadores qual destes irá executar a tarefa.

O escalonamento estático é aquele que é resolvido antes da execução das tarefas de maneira determinística. Quando é possível determinar todas as informações destas tarefas (dependência entre tarefas, custo de comunicação, custo de execução, etc) o escalonamento pode ser ótimo, em caso contrário sub-ótimo. Nesta última categoria, é possível optar pela busca de uma "boa" solução ao invés de procurar no espaço de soluções inteiro (aproximado). Os modelos heurísticos representam a categoria de algoritmos estáticos que assumem um conhecimento *a priori* sobre os processos e as informações do sistema.

Muitas vezes as informações a respeito das tarefas e do sistema não podem ser obtidas antes da execução, levando à utilização de um escalonamento dinâmico.

Dependendo se o escalonador deve residir fisicamente em um único computador ou deve estar fisicamente distribuído entre computadores, ele pode ser classificado, respectivamente, em distribuídos e não-distribuídos (ou centralizados).

Os escalonadores distribuídos são divididos entre aqueles que cooperam entre si e aqueles nos quais cada computador toma decisões independentemente das ações tomadas pelos outros. Os escalonadores cooperativos são classificados de forma análoga aos estáticos.

Existem várias características que não estão presentes na classificação proposta anteriormente. A seguir estão as principais encontradas na literatura:

- **adaptável e não-adaptável:** a diferença entre elas é o fato de que os adaptáveis mudam dinamicamente seu mecanismo de controle através do uso

de parâmetros do sistema, enquanto que os não-adaptáveis não alteram seu algoritmo;

- **alocação única e realocação dinâmica:** na primeira opção os trabalhos são alocados uma única vez a determinado processador, enquanto que na segunda podem migrar várias vezes antes de serem executados. Claramente, a segunda opção traz uma sobrecarga maior ao sistema;

- **preemptivo e não-preemptivo:** em um sistema preemptivo uma tarefa pode ser interrompida uma vez que iniciou sua execução, enquanto que no não-preemptivo isto não é possível;

- **sistema de várias aplicações e de única aplicação:** existem sistemas onde o escalonador está trabalhando com uma aplicação específica e, portanto, seu objetivo é minimizar o tempo total para realizar esta aplicação. Por outro lado, podem existir casos onde o escalonamento é para diversas aplicações e deve ser minimizado o tempo médio para resolução de cada aplicação envolvida;

- **balanceamento de carga:** esta política de escalonamento usa a filosofia de que ser justo com o uso dos recursos de *hardware* do sistema é bom para os usuários. A idéia básica é tentar balancear a carga em todos os processadores de maneira que executem a uma mesma taxa. Os computadores agem em conjunto, removendo trabalho de computadores muito carregados para computadores pouco carregados. Esta solução é mais eficiente quando os computadores são homogêneos, pois isso permite que todos os computadores saibam a estrutura do outro;

- **compartilhamento de carga:** esta política de escalonamento tem por objetivo distribuir a carga do sistema de computadores muito carregados para computadores pouco carregados no sistema. A idéia básica é aumentar o desempenho do sistema distribuindo a carga através dos computadores. Não existe a necessidade de que a taxa de execução dos computadores seja semelhante, e sim, de que não existam simultaneamente computadores

sobrecarregados e computadores pouco carregados. Esta solução é importante para eficiência tanto em sistemas homogêneos como heterogêneos.

No projeto de um escalonador de tarefas uma das decisões é a escolha entre uma política estática ou dinâmica. As políticas estáticas não utilizam informações sobre o estado do sistema para escalonar tarefas. Algoritmos estáticos assumem informações sobre os tempos médios de execução de tarefas e necessidades de intercomunicação entre todas as tarefas (EAGER, LAZOWSKA, ZAHORJAN, 1986b). Algoritmos dinâmicos, entretanto, decidem a distribuição de carga no sistema através do estado atual do sistema. Como o estado do sistema sofre alterações dinamicamente, políticas dinâmicas costumam possibilitar melhoras significantes de desempenho (DANDAMUDI et al, 1996). Existem dois componentes importantes em um escalonamento dinâmico, a política de transferência e a política de localização.

A escolha de um bom índice de carga tem um efeito considerável no desempenho do sistema e, por isso, alguns índices de carga foram propostos (EAGER, LAZOWSKA e ZAHORJAN, 1986a; MELLO, 2003; SHIVARATRI, KRUEGER e SINGHAL, 1992; THEIMER e LANTZ, 1989; ZHOU e FERRARI, 1987). Índices como tamanho da fila de execução da CPU, tempo médio da fila da CPU, utilização da CPU e quantidade de memória disponível são comuns. Apesar de intuitivamente parecer óbvio que o uso de índices sofisticados melhora o desempenho, via de regra não é o que ocorre. Quanto mais cálculos e parâmetros são utilizados, maior é a sobrecarga do sistema e mais difícil é a obtenção e a distribuição do índice (KUNZ, 1991).

A maior parte dos sistemas dinâmicos são das categorias fisicamente distribuído, cooperativo, sub-ótimo e heurístico (CASAVANT e KUHL, 1988; EVANS et al, 1993). Entretanto, o escalonamento dinâmico fisicamente não-distribuído, também denominado centralizado, apresenta importantes vantagens para multicomputadores. Na Figura 2.4 é apresentado o modelo de escalonamento centralizado.

As políticas centralizadas são mais eficientes que as distribuídas para ambientes com um baixo número de computadores (até 100 computadores) em uma rede local (velocidades de comunicação idênticas entre computadores). Por outro lado, elas apresentam problemas de escalabilidade, não tolerância a falhas, perda de paralelismo e imprecisão das informações globais do sistema (COSTA, 1998).

A escalabilidade pode ser definida como a capacidade de aumentar a quantidade de processamento que pode ser realizada em um sistema através da adição de mais recursos. Note que a escalabilidade difere de desempenho pois a escalabilidade não aumenta o desempenho, mas sim mantém o desempenho no caso de uma maior carga.

Normalmente, políticas centralizadas apresentam problemas de "gargalo" quando são submetidas a um elevado número de processadores (KUCHEN e WAGENER, 1990). Sistemas centralizados não possuem tolerância a falhas, pois caso ocorra algum problema no processador onde a tarefa escalonada está sendo executada ocasionará a interrupção total da aplicação (DANDAMUDI et al, 1996).

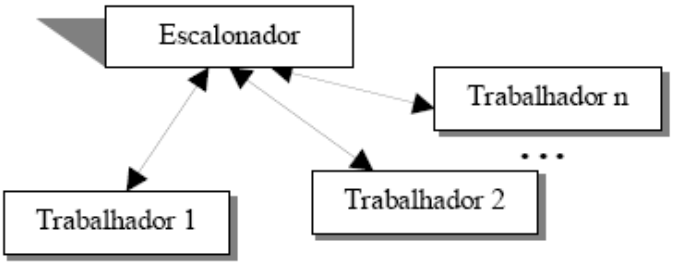


**Figura 2.4 –** Escalonamento Centralizado
Fonte: Costa (1998)

Como vantagens dos sistemas centralizados pode-se citar a simplicidade, maior eficiência para um baixo número de computadores, maior eficiência para sistemas com memória compartilhada e menor custo de comunicação (DANDAMUDI et al, 1996; YAMIN, 1994).

## 2.3 Principais Algoritmos de Balanceamento de Carga

Nesta Seção é apresentada uma breve descrição dos principais trabalhos relacionados à área de balanceamento de carga, solidificando conceitos novos e necessários para o entendimento do TLBA.

Os principais trabalhos que abordam as políticas de balanceamento de carga e apresentam contribuições relevantes para esta dissertação são descritos a seguir.

Zhou e Ferrari (1987) analisaram cinco algoritmos de balanceamento de carga em sistemas distribuídos fracamente acoplados que são iniciados pelo servidor, ou seja, pelo computador mais carregado.

O balanceamento de carga é dado no nível de tarefas. A coleta e gerenciamento das informações de carga e a decisão da transferência de tarefas são realizadas pelo *Load Information Manager* (LIM), um módulo de *software* que constantemente monitora esses parâmetros. Já o mecanismo de balanceamento de carga, por outro lado, é dado pelo *Load Balance Manager* (LBM), que inicia o processo de execução remota e estabelece uma conexão entre os computadores (ou nós) da rede.

Esses algoritmos são:

1) **Disted**: periodicamente, o LIM de cada computador verifica o índice de carga local e, caso haja uma diferença significativa entre o valor atual e o seu anterior, é enviada uma mensagem *broadcast* para os demais LIMs, para atualizarem suas tabelas de índices de carga com relação àquele computador. Quando um LIM recebe um pedido de transferência devido à execução de um processo, primeiramente é analisado se o índice de carga local está acima de um certo *Threshold*. Se for o caso, o LBM analisa os índices de carga dos demais computadores e, então, seleciona e transfere o processo para o computador que apresentou o

menor índice de carga. Caso todos os computadores estejam com um alto índice de carga, o processo é executado localmente.

2) **Global**: visa diminuir o número de mensagens geradas no *Disted* com a implantação de um LIM mestre que recebe os índices de carga dos demais LIMs e, periodicamente, envia um vetor contendo todos os índices de carga. A política de transferência permanece igual à do *Disted*.

3) **Central**: as informações de carga e os pedidos de transferência são enviados diretamente a um LIM e um LBM mestres, respectivamente. O LBM utiliza as informações do LIM mestre para alocar os processos no sistema.

4) **Random**: devido à sua simplicidade, não é necessária nenhuma informação de carga dos demais computadores. Quando uma requisição para execução de um processo chega ao sistema, é definido aleatoriamente um computador para executá-lo.

5) **Lowest**: esse algoritmo obtém informações de carga sob demanda. Quando uma requisição de um novo processo chega ao sistema, o LIM do computador que iniciou o processo define um conjunto limitado de LIMs remotos, captura sua carga e seleciona o mais ocioso para receber o processo iniciado. O *Lowest* difere dos anteriores, pois independe do tamanho do sistema ao qual está sendo aplicado. Esse algoritmo e o *Random* foram propostos por Eager et al (1986b).

Uma vez apresentados os algoritmos de Zhou e Ferrari (1987), na Figura 2.5, são representados os computadores receptores e emissores e a troca de informações envolvidas nesse sistema.

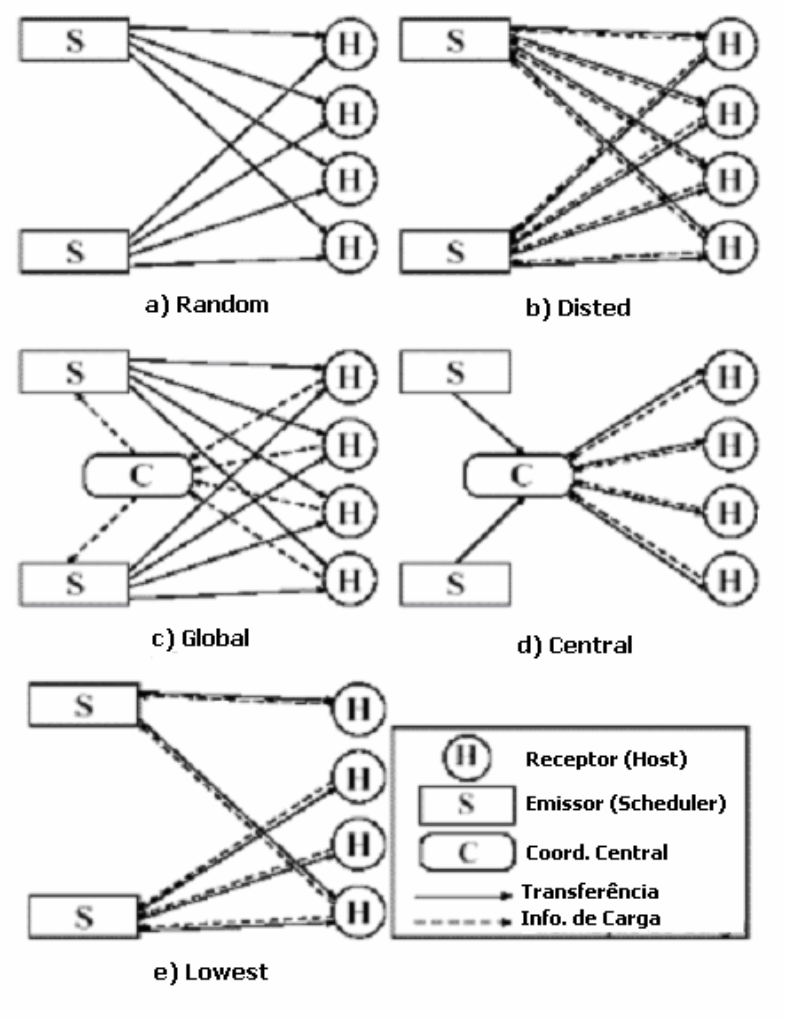


**Figura 2.5** – Escalonamento descrito por Zhou e Ferrari

Como medida comparativa desse estudo foi utilizado o tempo médio de resposta dos processos, juntamente com seus respectivos desvios padrão.

O índice de carga do ambiente foi calculado com base no comprimento da fila de processos em espera pela CPU. Computadores com muitos processos na fila estariam, segundo essa técnica, mais sobrecarregados.

A partir dos resultados apresentados por Zhou e Ferrari (1987), concluiu-se que todos os algoritmos apresentaram melhorias significativas de desempenho quando aplicados em sistemas sobrecarregados e com uma única CPU.

O *Random* apresentou um desempenho superior com relação aos ambientes onde não havia balanceamento de cargas e inferior a todos os outros algoritmos com relação ao tempo médio de resposta dos processos. Sua política de balanceamento de carga é, entretanto, a mais simples de todos.

O *Random*, devido à sua arbitrariedade, e o *Disted*, com sua estrutura de comunicação $N^2$, apresentaram um desempenho consideravelmente pior em relação aos outros três algoritmos.

O método de busca utilizado no *Lowest* foi o mais eficiente, tendo um menor fluxo de mensagens na rede. Com relação à alocação de processos, os algoritmos *Global*, *Central* e *Disted* mostraram-se melhores para a seleção de computadores mais ociosos na rede.

Foi possível também concluir que algoritmos centralizados, tal como o *Global* e o *Central*, respondem melhores a ambientes cuja carga é constantemente alterada, enquanto que algoritmos distribuídos, tal como o *Lowest*, impõem menor sobrecarga ao sistema e atingem maior escalabilidade. Já o desempenho do *Random e o Disted* deterioram consideravelmente com o aumento do sistema.

Finalmente, concluiu-se que o algoritmo *Lowest* é o mais indicado para ambientes distribuídos.

Já Theimer e Lantz (1989) estudaram um conjunto de algoritmos similares ao de Zhou e Ferrari (1987), mas aplicados a sistemas distribuídos compostos por aproximadamente 70 *workstations* com o Sistema Operacional V (BERGLUND, 1986) e seus algoritmos eram baseados em operações de *multicast* e execução remota transparente, facilidades fornecidas por esse sistema operacional.

Os algoritmos nesse estudo são muito similares aos *Disted*, *Central* e *Lowest* de Zhou e Ferrari (1987). Suas descrições são apresentadas a seguir.

1) **Algoritmo de Escalonamento Centralizado**: caracteriza-se pelo envio periódico de informações de atualização dos estados dos computadores e de receptores solicitando execuções remotas ao computador central. Esse algoritmo corresponde ao *Central* de Zhou e Ferrari (1987).

2) **Algoritmo de Escalonamento Descentralizado**: é totalmente descentralizado e requer que cada máquina realize a seleção do seu próprio receptor. É possível ter-se um algoritmo de escalonamento descentralizado se cada computador tem uma visão global do sistema continuamente ou requisitando quando necessário, a informação dos estados de cada computador através da geração de mensagens de atualização, como no algoritmo centralizado e envio de mensagens *broadcast* para todos os computadores. Essa metodologia não é eficiente, pois mesmo os computadores que já estão ocupados deverão capturar as mensagens de atualização dos outros computadores, bem como gerarem suas próprias mensagens. Além disto, caso ocorram problemas com um computador é necessária, de alguma forma, a reconstrução das informações perdidas dos estados dos demais computadores do sistema. Esse algoritmo é similar ao *Disted* e *Lowest* de Zhou e Ferrari (1987).

Em um algoritmo baseado no envio de mensagens direcionadas, apenas aquelas máquinas interessadas na seleção do receptor precisam se preocupar com o estado da informação. Se as mensagens são enviadas para um grupo *multicast* contendo somente máquinas no estado ocioso, as demais nunca participam do processo.

Esse modelo é aquele no qual um cliente interessado em obter a seleção de um receptor envia uma mensagem para um grupo *multicast,* requisita a informação atual dos estados dos computadores pertencentes a esse grupo e aguarda algumas das primeiras respostas.

De posse da resposta dos possíveis receptores é selecionado o mais ocioso, que receberá o processo iniciado.

No trabalho proposto por Shivaratri, Krueger e Singhal (1992), o balanceamento de cargas em sistemas de redes locais é vantajoso, pois proporciona um aumento do desempenho, escalabilidade e extensibilidade a um baixo custo. Os autores propuseram o estudo de cinco algoritmos de balanceamento de carga e a comparação de seus desempenhos. Esses algoritmos são descritos a seguir.

1) **Algoritmos Iniciados pelo Emissor**: um computador é considerado emissor se é o precursor de uma nova tarefa que proporciona um aumento de sua carga acima de um determinado limiar. Por outro lado, o computador que recebe esta tarefa e mantém seu índice de carga abaixo de um limiar é chamado de receptor.

Há três tipos de algoritmos iniciados pelo emissor:

a) *Random*: sua política de localização é muito simples, pois não utiliza nenhuma informação remota. Simplesmente, um computador é escolhido, aleatoriamente, e recebe o processo do emissor. Para que não ocorram migrações freqüentes de processos, é definido um limite para as transferências. Apesar de sua simplicidade e de não possuir balanceamento de cargas, sua política de localização fornece um desempenho médio sobre o sistema.

b) *Threshold*: esse algoritmo evita a transferência desnecessária de processos através do monitoramento de computadores selecionados aleatoriamente, determinando se uma possível migração a um suposto receptor aumentará sua carga acima de um certo limiar. Caso isto não aconteça, o computador recebe o processo, se não, outro computador é selecionado aleatoriamente e monitorado. Se nenhum receptor for encontrado, o processo é executado localmente. Para evitar uma

sobrecarga do sistema, é definido um número limitado de monitoramento através de um parâmetro chamado *poll limit*.

c) *Shortest*: nesse caso, um número de computadores é selecionado aleatoriamente e suas cargas são monitoradas. Após a comparação entre elas, o computador mais ocioso, ou seja, o que tem menor carga, é definido como o receptor. Caso esse receptor esteja mais sobrecarregado que o emissor, o processo é executado localmente.

2) **Algoritmos Iniciados pelo Receptor**: o pedido de migração é iniciado pelos computadores mais ociosos. No trabalho realizado por Livny e Melmann (1982) e Eager, Lazowska e Zahorjan (1986a), é proposto esse tipo de algoritmo.

A política de transferência dá-se a partir do momento em que um processo é iniciado. Se o índice de carga do computador local cai abaixo de um certo limiar, ele é identificado como receptor e, aleatoriamente, seleciona computadores (emissores) a serem monitorados. Caso a carga de um deles seja maior que um determinado limite, o processo é transferido ao receptor.

Se nenhum dos computadores monitorados estiver sobrecarregado, nenhuma atitude é tomada. Após um certo tempo, essas operações são reiniciadas.

3) **Algoritmos Iniciados Simetricamente**: a transferência de processos se dá tanto por parte do emissor quanto do receptor. Esses algoritmos apresentam vantagens com relação aos anteriores, nos quais algoritmos iniciados pelo emissor apresentam melhores resultados em baixas cargas para localizar computadores ociosos e os iniciados pelos receptores em altas cargas para localizar os computadores sobrecarregados. Por outro lado, apresentam desvantagens com relação aos posteriores. Algoritmos iniciados por emissores que monitoram computadores em altas cargas podem trazer instabilidade ao sistema,

enquanto que os iniciados por receptores apresentam uma maior complexidade nas implementações por preempção e migração de processos, sendo necessário um aumento no número de mensagens para sincronismo das operações.

4) **Algoritmos Adaptativos Estáveis Simetricamente Iniciados**: a instabilidade dos algoritmos gerados anteriormente deve-se ao grande monitoramento do sistema por computadores susceptíveis ao envio de informações. Esse algoritmo armazena as informações monitoradas de cada computador (descartadas nos algoritmos anteriores) em listas estruturadas de dados, classificando-as em: sobrecarregados (emissores), ociosos (receptores) e computadores com cargas médias.

Inicialmente, cada computador leva em consideração que os outros são receptores. Esse estado é representado em cada computador por uma lista de receptores, contendo todos os computadores (exceto ele mesmo) e listas de emissores e computadores com carga média.

O monitoramento cessa quando um emissor é encontrado ou um computador deixa de ser receptor ou é atingido o *poll limit*.

Um computador é considerado emissor se estiver com carga acima de um limite superior. Por outro lado, é considerado receptor se estiver abaixo de um limite inferior e com carga média se estiver entre tais limites.

Em sistemas sobrecarregados é pouco provável encontrar computadores ociosos, tendo-se, então, a lista de receptores vazia e vice-versa. Assim sendo, a melhor política de localização é, respectivamente, a iniciada pelo receptor e a iniciada pelo emissor. Para cargas médias, o algoritmo é iniciado simetricamente.

5) **Algoritmos Adaptativos Estáveis Iniciados pelo Emissor**: esses algoritmos são similares aos algoritmos adaptativos estáveis iniciados simetricamente, porém a política de transferência do iniciado pelo receptor foi modificada para atrair tarefas não-preemptivas de emissores. Uma característica importante é a realização de balanceamento de carga somente com tarefas não-preemptivas, com um menor custo gerencial para o sistema.

Nesses algoritmos as estruturas de dados (de cada computador) dos algoritmos adaptativos estáveis simetricamente iniciados são substituídas por vetores de estados. Cada computador usa esses vetores para saber em quais listas (emissor, receptor ou carga média) é enquadrado em comparação com os demais nós do sistema. A sobrecarga para manter esses vetores de estados é pequena e constante, independentemente do número de nós no sistema.

No algoritmo iniciado pelo emissor, um computador é selecionado e adicionado no vetor de estados, mostrando que agora aquele emissor faz parte da lista de emissores daquele computador. O computador monitorado é avisado sobre essa ação.

O algoritmo iniciado pelo receptor foi substituído pelo seguinte protocolo: quando um computador torna-se receptor, ele informa apenas aos computadores que não possuem a informação de seu novo estado. Assim, o vetor de estados daquele computador é atualizado para informar que ele agora pertence à lista de receptores de todos os outros computadores que estavam desatualizados sobre o seu estado atual.

Essa pesquisa analisou os tempos médios de resposta de processos submetidos aos algoritmos, concluindo que os estáveis simetricamente iniciados apresentam os melhores resultados de desempenho. Deve-se destacar que o índice de carga adotado é calculado com base no número de processos em espera pela CPU.

Baseado nos estudos apresentados anteriormente foi proposto por Mello (2003) um algoritmo de balanceamento de cargas para ambientes distribuídos heterogêneos escaláveis, denominado *Tree Load Balancing Algorithm* (TLBA), projetado para executar na forma de módulos sobre cada um dos computadores do sistema. A origem do nome deve-se à criação da topologia lógica no formato de árvore, que interconecta os computadores do sistema.

Existem dois componentes fundamentais envolvidos no aspecto físico do algoritmo. Eles são apresentados a seguir.

1) ***Broker:*** é o componente responsável pela montagem de toda a árvore de computadores interconectados. O *broker* define em que nível da árvore um computador será inserido, qual seu antecessor e quais seus sucessores. Essa inserção é dada de tal forma a manter a árvore balanceada. Para isso, um *benchmark* é executado pelo computador durante a etapa que requer sua participação no sistema[2].

2) ***Peer:*** esse componente é responsável por requisitar a adição de um computador na árvore e implementa as políticas de balanceamento de carga do algoritmo TLBA.

Para participar da árvore lógica de interconexões, cada computador deve negociar com o *Broker*. Ele define em que nível da árvore um computador será inserido, qual seu antecessor e quais seus sucessores. Em seguida, é enviada uma mensagem contendo tal informação. Nessa etapa, o computador inicia conexões com seu antecessor e aguarda a conexão de seus sucessores. O *Broker* contém toda a estrutura hierárquica dessa árvore lógica. Ele pode ser replicado em outros computadores para garantir alta disponibilidade.

[2] Para maiores informações sobre esse *benchmark*, consulte o site: http://www.cs.wisc.edu/~zandy/ckpt.

O *Peer*, periodicamente, analisa a ocupação do recurso de CPU e memória principal e em seguida calcula o índice de carga. Esse evento periódico tem intervalos definidos pelo administrador do sistema.

Mello (2003), propôs um novo índice de carga para o TLBA, baseado na ocupação de CPU e de memória. Esse novo índice contribui para o aumento de desempenho e estabilidade no balanceamento de cargas, comprovado através de sua aplicação no *Lowest Modificado* (MELLO, TREVELIN e PAIVA, 2003b) e comparado aos Algoritmos Simetricamente Iniciados (SHIVARATRI, KRUEGER e SINGHAL, 1992).

As políticas de balanceamento de cargas implementadas por esse algoritmo seguem descrições e metodologias de gerenciamento apresentadas a seguir.

A política de informação implementada pelo TLBA utiliza índices de carga baseado na ocupação da CPU e de memória. Com auxílio desses índices é possível determinar os computadores sobrecarregados e ociosos no sistema, de forma a distribuir eqüitativamente a carga dos computadores do sistema.

Na Figura 2.6 é representada a propagação de índices de carga calculados por um computador $\alpha$ (D), localizado no último nível *Nk* de uma árvore, para seu antecessor $\beta$ (B), localizado no nível *Nk-1*. O computador $\beta$ armazena em memória seu índice de carga e de seus sucessores. O computador $\beta$, devido ao fato de apresentar sucessores realiza um somatório de seu índice e dos sucessores, submetendo o resultado ao seu antecessor na árvore. Essa operação é propagada até chegar à raiz da árvore (A), atualizando as informações do sistema (MELLO, 2003).

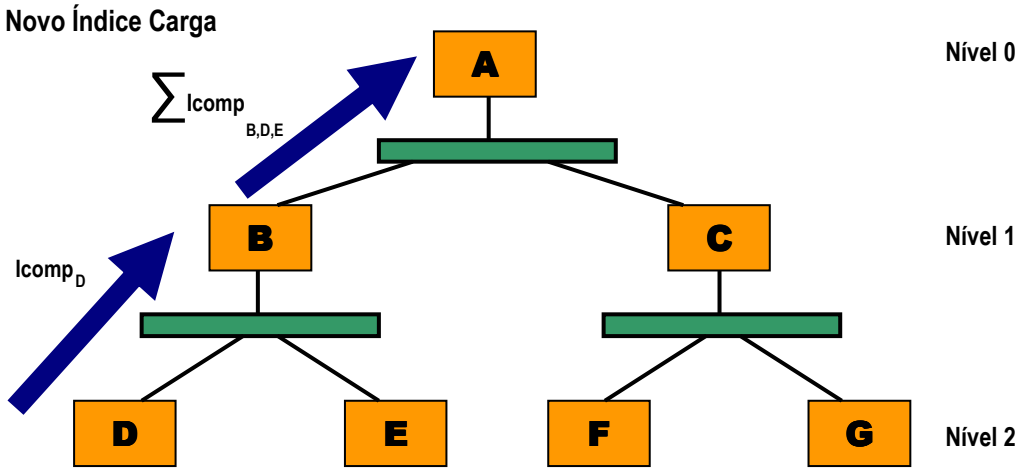


**Figura 2.6** – Política de Informação do TLBA, onde ocorre a propagação do índice de carga dos computadores nas "folhas" da árvore até chegar a raiz.

A política de localização do TLBA utiliza índices de carga calculados em cada computador do sistema e que são propagados para seus respectivos antecessores até chegar no nível do computador raiz. Para evitar que essa transferência de informações sobrecarregue o sistema, a propagação de dados é feita se houver mudanças significativas na ocupação dos recursos. Essas mudanças são baseadas em um parâmetro denominado *Threshold* de Atualização de Cargas, que é adicionado ou removido de índices de carga e cuja variação superior ou inferior definirá se é necessária a propagação de índices de carga para os antecessores.

No algoritmo TLBA, a política de localização busca somente por computadores receptores. Essa busca é realizada da forma *top-down*, iniciando-se pelo computador raiz e propagando-se até encontrar o computador mais ocioso, ou seja, aquele que tem menor carga.

A política de localização do algoritmo TLBA privilegia a localização de computadores mais próximos das folhas, ou seja, computadores presentes no último nível *Nr* da árvore. Assim sendo, para casos onde o sistema apresenta baixa carga, as mensagens de busca em profundidade tendem a percorrer todos os níveis e localizar computadores receptores próximos do nível *Nr*. Já

em altas cargas, os receptores de processos são localizados mais próximos do primeiro nível da árvore, portanto, nesses casos, há menor consumo de recursos do ambiente (MELLO, 2003).

Uma vez localizado o melhor receptor de processos, seu endereço é enviado para o *broker* ou computador sobrecarregado:

- **é enviado para o *Broker***, caso esse tenha recebido uma requisição para executar um novo processo no ambiente e tenha ativado a política de localização;
- **é enviado para o computador sobrecarregado**, caso esse tenha ativado a política de localização para transferir parte de sua carga.

A política de seleção de um algoritmo de balanceamento de carga é responsável por selecionar processos para transferência. No algoritmo TLBA essa política busca pelo processo que ocupa maior quantidade dos recursos de CPU e memória em um computador sobrecarregado $\alpha$.

Uma vez localizado o processo de maior ocupação, seu identificador é submetido para a política de transferência que cuida das etapas de salvar seu contexto, transferi-lo para seu distino e reiniciá-lo (ZANDY, 2004).

O *checkpointing* salva o contexto de um determinado processo, para que seja reiniciado em um outro computador (CHANCHIO e SUN, 2001). A última etapa é a da transferência propriamente dita.

No entanto, todas as etapas geram um grande custo computacional e são realizadas nos computadores que apresentarem processos ocupando a maior quantidade de CPU e memória.

A validação desse algoritmo foi realizada através de um simulador e um protótipo e seus resultados comprovaram um baixo número de mensagens

geradas em operações de balanceamento de carga, maior estabilidade em altas cargas e baixos tempos médios de resposta em comparação ao algoritmo *Lowest* modificado, exceto em sistemas de baixas cargas para os tempos de resposta médio e mediano (MELLO, 2003; MELLO, TREVELIN e PAIVA, 2003b).

Apesar de o TLBA ter apresentado contribuições relevantes à área em estudo, as simulações realizadas anteriormente apresentam algumas limitações que são descritas a seguir.

A função de distribuição de probabilidade adotada por Mello (2003), foi a distribuição de Poisson. Atualmente, a maioria dos trabalhos apresentados utilizam-se do método de Feitelson (ver Seção 2.4), como modelo de carregamento de cargas em sistema distribuído (FEITELSON, 1996; FEITELSON, 2001; FEITELSON e JETTE, 1997; FEITELSON e RUDOLPH, 1998; FEITELSON e WEIL, 1998; GHARE e LEUTENEGGER, 1999; LO, MACHE e WINDISCH, 1998; MELLO e SENGER, 2004; MU´ALEM e FEITELSON, 2001; TALBY, FEITELSON e RAVEH, 1999);

Foram analisadas apenas duas capacidades de processamento heterogêneo. Para melhor verificar o comportamento desse algorimo em ambientes heterogêneos é necessário diversificar ainda mais as capacidades computacionais aonde ele é aplicado.

O número de repetições (iterações) realizadas pelo simulador para a análise de precisão do tempo médio de resposta foi de 100 vezes. Esse número não necessariamente garante a significância estatística dos resultados.

Na realidade não é possível encontrar uma estimativa perfeita para a média a partir de um número finito de amostras de tamanho finito. Neste caso, a melhor opção é obter limites probabilísticos de forma que seja alta a probabilidade de a média exata pertencer a um determinado intervalo, também conhecido como intervalo de confiança para a média.

Dessa forma, neste trabalho foram utilizados intervalos de confiança e o número de repetições ficou condicionado à obtenção de um resultado dentro do intervalo desejado.

No trabalho de Mello (2003) não foi feita nenhuma análise com relação as característas do comportamento da aplicação em *CPU Bound*, *I/O Bound, Memory Bound, Network Bound.*

Já outros trabalhos introduzem conceitos interessantes com relação aos custos de migração de um determinado processo. Esses custos estão relacionados à operação de transferência do contexto de um processo em uma rede de computadores e ao restabelecimento dos canais de comunicação existente entre os processos (MELLO e SENGER, 2004).

No trabalho apresentado por Halchor-Balter e Downey (1997) é proposto um modelo de custos de migração baseado na idade dos processos. Esse estudo conclui que existem benefícios na migração de processos que demandam um longo período de execução desde o seu início até o fim da sua execução. Assim sendo, esse estudo propõe um modelo de migração baseado na idade dos processos em execução.

De acordo como Mello e Senger (2004), esse estudo está incompleto em suas avaliações, pois além de se analisar o tempo de execução dos processos é necessário avaliar a carga de tais processos no sistema. Logo, processos com longo tempo de execução, mas com baixa taxa de ocupação, quando migrados não agregam benefícios ao computador emissor. Ao contrário, esses processos acumulam um custo de migração que aumenta os seus tempos de execução. Mello e Senger (2004) propõem um novo modelo de migração baseado no conhecimento do tempo de execução dos processos e de suas cargas.

## 2.4 Modelo de Carregamento de Cargas

Uma das formas de se avaliar algoritmos de balanceamento de cargas sem a implementação total de um protótipo é através da simulação. Naturalmente, a qualidade de seus resultados está relacionada aos parâmetros de entrada utilizados na simulação. Assim sendo, um dos fatores fundamentos a serem considerados é a escolha do modelo de carregamento de cargas (FEITELSON, 1996).

Alguns autores afirmam que não existe informação confiável sobre carregamento de cargas em máquinas paralelas (CHIANG, MANSHARAMANI e VERNON, 1994; KRUEGER, LAI e RADIYA, 1991; LEUTENEGGER e VERNON, 1990; MAJUMDAR, EAGER, BUNT, 1988). No entanto, de acordo com Feitelson (1996) isso não é verdade. Assim como sistemas uniprocessados, a maioria dos sistemas paralelos mantém o rastreamento de todas as tarefas em execução em um sistema. A análise deste rastreio fornece informações significantes com relação a distribuição de cargas no sistema.

Nos estudos realizados por Feitelson (1996) são utilizados as informações obtidas de 6 diferentes tipos de sistemas apresentados a seguir e com distribuição real de cargas.

- 128-nós do iPSC/860 na NASA Ames;
- 128-nós do IBM SP1 em Argonne;
- 400-nós do Paragon no SDSC;
- 126-nós do Butterfly no LLNL;
- 512-nós do IBM SP2 no CTC;
- 96-nós do Paragon no ETH, Zurich;
- 512-nós do IBM SP2 no CTC;
- 96-nós do Paragon no ETH, Zurich;

Nos 3 primeiros casos as informações foram obtidas diretamente do sistema e nos demais de observações publicados.

Esse estudo teve como finalidade produzir um modelo de carregamento de cargas que refletisse a de um sistema real e pudesse ser utilizado em simulações com escalonamento paralelo (FEITELSON, 2006).

De forma a complementar esses estudos Feitelson e Jette (1997) observaram que execuções repetidas da mesma aplicação tendem a apresentar padrões similares de consumo de recursos e que muitas informações, relativas ao comportamento das aplicações, podem ser descobertas sem a cooperação explícita do usuário, através da observação do histórico de execuções passadas.

## 2.5 Considerações Gerais

Esse capítulo apresentou os principais trabalhos relacionados à área de balanceamento de carga e alocação de recursos computacionais, visando o aumento de desempenho na execução de aplicações.

Dentre eles está o TLBA, com uma proposta de utilização de um índice de carga baseado em CPU e memória. Neste algoritmo foram comprovados a queda nos tempos médios de resposta dos processos e um baixo número de mensagens geradas no meio de comunicação.

Trabalhos relacionados da área de balanceamento de cargas propõem uma grande variedade de objetivos como o compartilhamento de carga, melhor distribuição de recursos entre os computadores, redução do tempo de resposta, redução do tempo de execução, entre outros (FEITELSON, 1996; FEITELSON e JETTE, 1997; FEITELSON e RUDOLPH, 1995, 1996, 1998; MÜLLENDER, 1993; SHIRAZI e HURSON, 1992; XU e LAU, 1997).

A utilização dos algoritmos de balanceamento de cargas, visando uma melhor distribuição dos recursos de um determinado sistema, pode prover um aumento considerável no seu desempenho uma vez que a otimização e a eficiência do sistema estão diretamente relacionadas com a melhor utilização da capacidade de processamento existente (BRANCO, 2002).

Por outro lado, qualquer que seja o algoritmo de balanceamento de cargas utilizado é preciso que administre da melhor forma possível às informações obtidas dos computadores da rede. Desta forma, deve ser selecionado o processo que trará menor sobrecarga ao sistema na sua transferência, que tenha um elevado tempo de execução e poucas dependências relacionadas ao sistema local.

O algoritmo de balnaceamento de cargas também deve verificar a viabilidade de transferência do processo para um determinado computador (receptor), a fim de não sobrecarregar sua fila de espera e evitar a necessidade de uma nova transferência dessa tarefa para um outro receptor (SHIVARATRI, KRUEGER e SINGHAL, 1992).

Esse controle de informações submetidas em um determinado sistema ou uma melhor adequação da utilização do algoritmo em função da homogeneidade ou heterogeneidade do sistema é realizado através dos parâmetros de balanceamento de cargas.

Levando-se em consideração a importância da otimização dos parâmetros de balanceamento de cargas e das limitações dos estudos realizados por Mello (2003), essa dissertação contribui de forma a complementar esses estudos, empregando o modelo de carregamento de carga baseado nos estudos realizados por Feitelson (1996).

Capítulo

3

# 3 AVALIAÇÃO DE DESEMPENHO E MODELAGEM DE SISTEMAS COMPUTACIONAIS

## 3.1 Introdução

O desenvolvimento tecnológico nas diversas áreas de conhecimento vem sendo apoiado por sistemas computacionais cada vez mais rápidos e com maior capacidade. Por sua vez, grande parte dos avanços tecnológicos tem contribuído para a evolução dos sistemas computacionais, melhorando suas características, diminuindo seus custos e proporcionando um alto desempenho computacional.

A computação com alto desempenho é a integração de conceitos de *hardwares* e *softwares,* no intuito de resolver desafios computacionais em um menor período de tempo do que o realizado por computadores seqüenciais (arquitetura de Von Neumann), tendo-se em uma única máquina diversos processadores, cooperando entre si. Devido às limitações tecnológicas uma melhora expressiva em arquiteturas de Von Neumann é uma árdua tarefa. Assim sendo, a computação paralela tem sido tratada como uma alternativa atraente, particularmente quando os problemas a serem solucionados são essencialmente paralelos (BRANCO, 2002; CENTRO DE COMPUTAÇÃO DA UNICAMP e FINEP, 2002).

Os altos custos dos equipamentos com arquiteturas paralelas e implantação desses sistemas dificultaram sua disseminação. Por outro lado, o desempenho dos computadores pessoais e das estações de trabalho tem apresentado um aumento significativo ao longo das últimas décadas.

Ao mesmo tempo o custo relativamente baixo dessas máquinas, favoreceu a sua ampla utilização levando à interconexão entre elas, o que permitiu o estabelecimento de sistemas computacionais distribuídos, constituindo uma área amplamente difundida e pesquisada nas últimas três décadas (COULORIS, DOLLIMORE e KINDBERG, 1994; TANENBAUM, 1996).

Os sistemas distribuídos mostraram ao longo dos anos suas vantagens sobre os sistemas centralizados, conquistando em pouco tempo um lugar de destaque no cenário computacional (BRANCO, 2002).

Embora não haja um consenso quanto quais características devam ser consideradas formalmente em sistemas computacionais distribuídos, transparência é um requisito comum em todas classificações propostas por diversos autores (COULORIS, DOLLIMORE e KINDBERG, 1994; MÜLLENDER, 1993; TANENBAUM, 1996).

Sistemas computacionais distribuídos são potencialmente mais confiáveis do que os tradicionais, uma vez que possuem a propriedade de falha parcial. Visto que os processadores são autônomos, uma falha em um deles não afetará o funcionalmente normal dos demais processadores. Dessa forma, a confiabilidade pode ser incrementada pela replicação de funções ou dados críticos da aplicação nos processadores (SCOFIELD, 2002).

No decorrer do tempo, estudos foram realizados aliando-se os conhecimentos da computação paralela com o potencial dos sistemas distribuídos. A união dos custos relativamente baixos oferecidos pelos sistemas computacionais distribuídos ao alto desempenho fornecido pelo processamento paralelo, deu origem ao surgimento da “computação paralela distribuída” (BRANCO, 2002).

O processamento paralelo e distribuído tem se popularizado nos últimos anos, motivado pela possibilidade de aumento do desempenho na execução de diferentes tipos de aplicações quando comparadas com o desempenho obtido através da computação seqüencial.

Uma das áreas da computação que tem se tornado essencial com essa grande evolução dos sistemas computacionais é a avaliação de desempenho. Indicando a qualidade do trabalho executado por um sistema, ela pode ser utilizada na seleção de sistemas alternativos, no projeto de novos sistemas e na análise comparativa de sistemas existentes (SPOLON, 2001).

Nos sistemas existentes, a análise de desempenho refere-se à otimização da eficiência e da utilização dos recursos, processamento a um custo mínimo e diminuição do tempo de resposta. Em sistemas inexistentes, a avaliação de desempenho deve fazer previsões sobre o comportamento do novo sistema.

Na abordagem de avaliação de desempenho existem várias técnicas que vão desde *benchmarks* até metodologias por predição de desempenho (AIZA, 2003; BRUSCHI, 1997, 2002; NAIK, 1985; SANTANA et al, 1994; SPOLON, 2001; WEICKER, 1990; WILLEBEEK-LEMAIR e REEVES, 1993; WU, 1997; LAINE, 2003). A escolha de qual metodologia utilizar depende, fundamentalmente, do objetivo do estudo em andamento e do tipo de avaliação desejada.

Neste trabalho, o método de avaliação adotado foi a simulação, devido ao fato de oferecer uma grande flexibilidade aliada ao baixo custo. Isso possibilitou praticidade e facilidade na hora do estudo dos diferentes casos avaliados e na obtenção dos resultados.

Nas Seções seguintes será apresentada a evolução das arquiteturas em busca de um melhor desempenho e metodologias para a avaliação de um sistema computacional.

## 3.2 Clusters e a Supercomputação

O desenvolvimento das arquiteturas paralelas levou à construção de máquinas especializadas em realizar grandes quantidades de operações por segundos. Esses computadores que proporcionavam um maior desempenho na resolução de problemas eram denominados de Supercomputadores.

De acordo com Omêna (2004), a IBM tem uma visão bastante particular sobre o que é supercomputação e qual sua função atualmente e cita a seguinte definição: "[...] Supercomputadores são máquinas construídas sob encomenda, com a capacidade de processamento grande o suficiente para resolver complexos problemas referentes às aplicações para as quais foram desenvolvidos".

Em geral, os supercomputadores são construídos para obter alto desempenho, são máquinas que possuem alto poder de processamento e sua arquitetura totalmente voltada para a obtenção de resultados específicos ao tipo de tarefa a ser realizada.

No entanto, devido aos altos custos dos supercomputadores, formou-se uma nova organização de arquiteturas computacionais, composta por várias categorias (microcomputadores, estações de trabalho, *mainframes* e supercomputadores) e que segue uma estrutura piramidal (FOX e FURMANSKI, 1997; HWANG e XU, 1998), de acordo com duas variáveis custo/desempenho e volume de vendas, conforme apresentado na Figura 3.1 (adaptada de HWANG e XU, 1998). O que se pode observar é que a quantidade de computadores topo de linha é relativamente pequena, devido basicamente aos seus elevados custos.

Na Figura 3.1, à medida que se desce na estrutura piramidal, a partir do topo (onde ficam os chamados "supercomputadores"), encontram-se bases cada vez maiores de *mainframes*, seguidos de estações servidoras e, finalmente, milhões de microcomputadores. Uma observação importante é a de que, embora se percebam sensíveis diferenças nos custos, a tecnologia atual

pôde prover microcomputadores com desempenhos cada vez melhores. Isso é uma das principais motivações para a utilização das redes de computadores, onde existe uma relação custo/desempenho extremamente favorável (SOUSA, 2001).

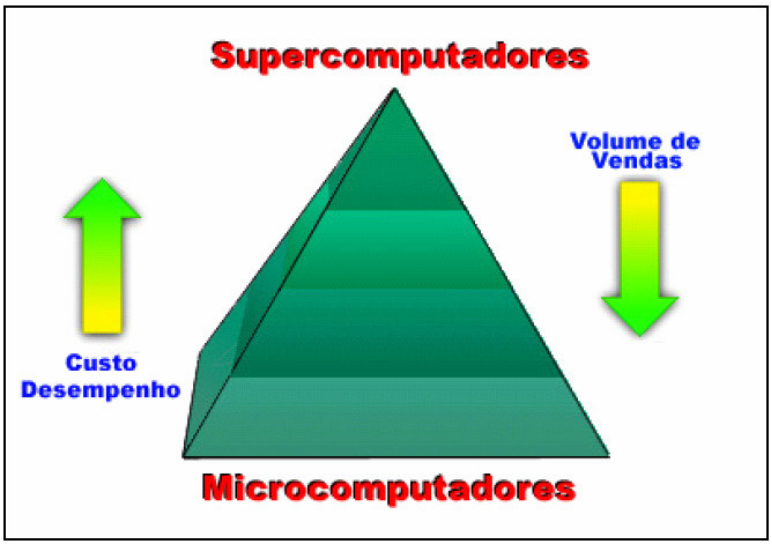


**Figura 3.1** – Distribuição das Categorias de Computadores.
Fonte: Adaptada de Hwang e Xu (1998)

A convergência de desempenho entre os microcomputadores e os supercomputadores com um só processador motivou o aparecimento do processamento paralelo, que vem ultrapassar alguns limites do uniprocessamento, reduzir alguns custos (podem construir-se supercomputadores recorrendo a um conjunto de processadores convencionais), e também para facilitar a expansibilidade de um computador (bastando para isso adicionar mais processadores) (MARTINS e COENTRO, 2005).

O processamento paralelo vem permitindo que problemas complexos sejam solucionados e aplicações de alto desempenho sejam desenvolvidas tanto nas áreas mais tradicionais, tais como ciência e engenharia, como nas áreas mais recentes, tais como inteligência artificial (*data mining*, redes neurais etc), financeira e bio-informática (SKILLICORN e TALIA, 1998).

Contudo, introduzir o conceito processamento paralelo não é tão simples. Existem diversas formas de organizar os processadores em paralelo além da necessidade de manter as informações que trafegam pela máquina coerentes e de gerenciar os elementos de processamento (processadores). O tempo de execução de um programa seqüencial não varia muito quando a execução muda de uma máquina seqüencial para outra. Isto já não pode ser observado quando se trata de programas paralelos, pois um fator muito importante na elaboração de programas paralelos é a comunicação entre os elementos de processamento e esta varia muito de um sistema para outro (SKILLICORN e TALIA, 1998).

Com o avanço da computação paralela, foram propostas várias maneiras para se conectar os recursos computacionais, criando-se diferentes arquiteturas paralelas. Cada arquitetura apresenta determinadas características, visando melhor desempenho sob um dado enfoque (BRANCO, 2002). Para acompanhar o desenvolvimento das arquiteturas paralelas e agrupar os equipamentos com características comuns, foram propostas algumas taxonomias, dentre elas a de Flynn (1972), Flynn e Rudd (1996) e a de Duncan (1990).

### 3.2.1 Multiprocessadores e Multicomputadores

As classificações de Flynn (1972) e Flynn e Rudd (1996) são baseadas no fluxo de dados (*Single Data stream ou Multiple Data stream*) e no fluxo de instruções (*Single Instruction stream* ou *Multiple Instruction stream*) (Figura 3.2), cuja combinação dessas possibilidades, produz quatro novos categorias de máquinas: *Single Instruction Single Data* (SISD), *Single Instruction Multiple Data* (SIMD), *Multiple Instruction Single Data* (MISD) e *Multiple Instruction Multiple Data* (MIMD) (OLIVEIRA, CARISSIMI e TOSCANI, 2005).

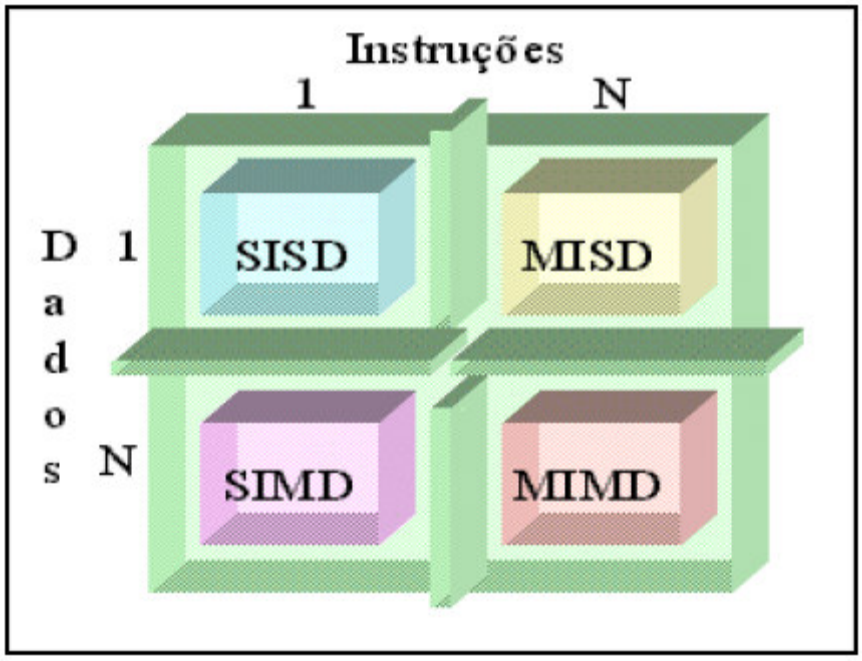


**Figura 3.2** – Classificação de Flynn.
Fonte: Sousa (2001)

As arquiteturas SISD representam o modelo de funcionamento seqüencial que corresponde à arquitetura de Von Neumann. O processador lê as instruções da memória, uma a uma, e as executa sobre um fluxo de dados provenientes da memória (Figura 3.3). Os representantes dessa categoria são os computadores monoprocessados como, por exemplo, os computadores pessoais e estações de trabalho.

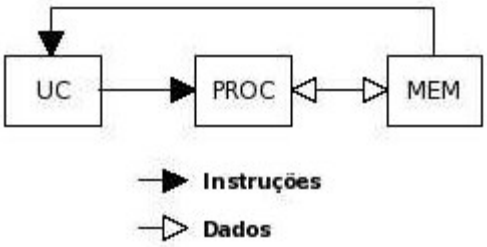


**Figura 3.3** – Arquitetura SISD
Fonte: Yokokura (2005)

Já na arquitetura MISD, enquadram-se as máquinas que executam múltiplas instruções, todas ao mesmo tempo, sobre um único fluxo de dados (Figura 3.4) (PITANGA, 2003).

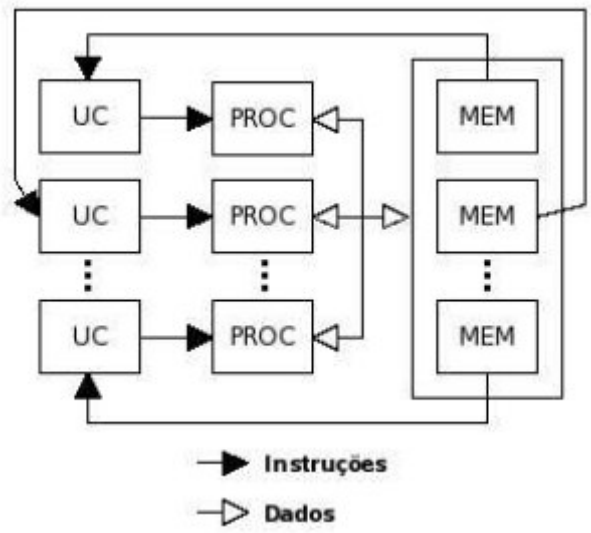


**Figura 3.4** – Arquitetura MISD
Fonte: Yokokura (2005)

Na arquitetura SIMD, a unidade de controle gerencia os processadores, enviando para cada um deles a mesma instrução. Cada instrução é executada de forma paralela e síncrona sobre fluxos de dados distintos (Figura 3.5). Essa é uma arquitetura bem específica, onde se encaixariam os computadores vetoriais como o CRAY-1 (YOKOKURA, 2005).

Uma arquitetura MIMD é composta por diversos processadores, onde cada processador tem a liberdade de executar um fluxo de instruções próprio sobre um fluxo de dados. Têm-se, então, elevados fluxos de instruções sobre diferentes fluxos de dados (Figura 3.6). As máquinas denominadas de multiprocessadores são exemplos típicos dessa categoria (OLIVEIRA, CARISSIMI e TOSCANI, 2005).

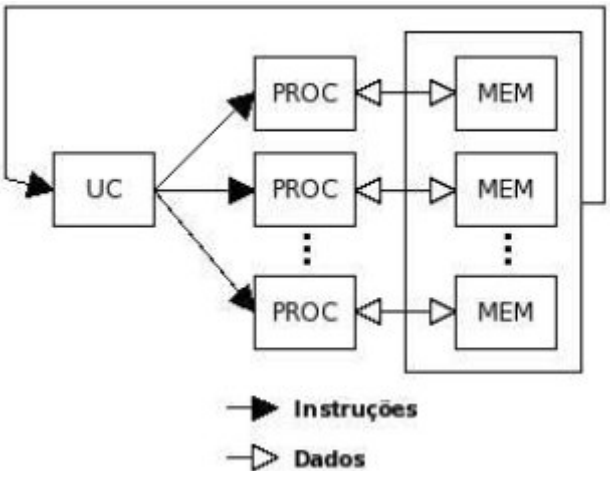


**Figura 3.5** – Arquitetura SIMD
Fonte: Yokokura (2005)

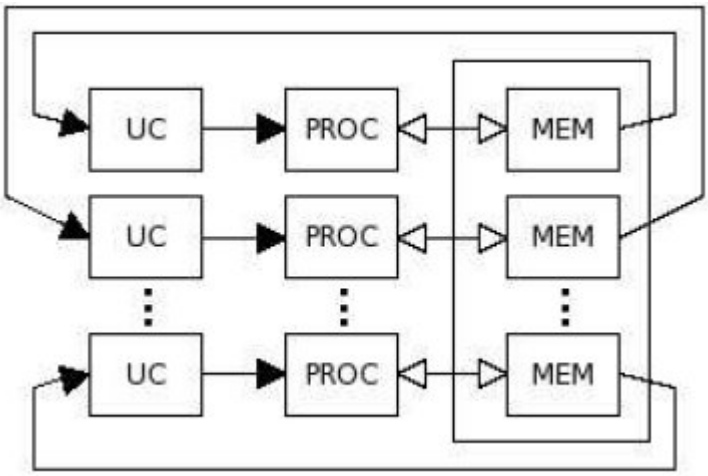


**Figura 3.6** – Arquitetura MIMD
Fonte: Yokokura (2005)

Os critérios adotados por Flynn são muito genéricos e, por conseqüência, sua classificação é inadequada para os sistemas distribuídos porque todos são considerados máquinas MIMD independentes de sua real arquitetura. Atualmente, é comum subdividir a classe MIMD em dois grupos: os multiprocessadores e o multicomputadores (TANENBAUM, 1996; VAN DER STEEN e DONGARRA, 1996) (Figura 3.7).

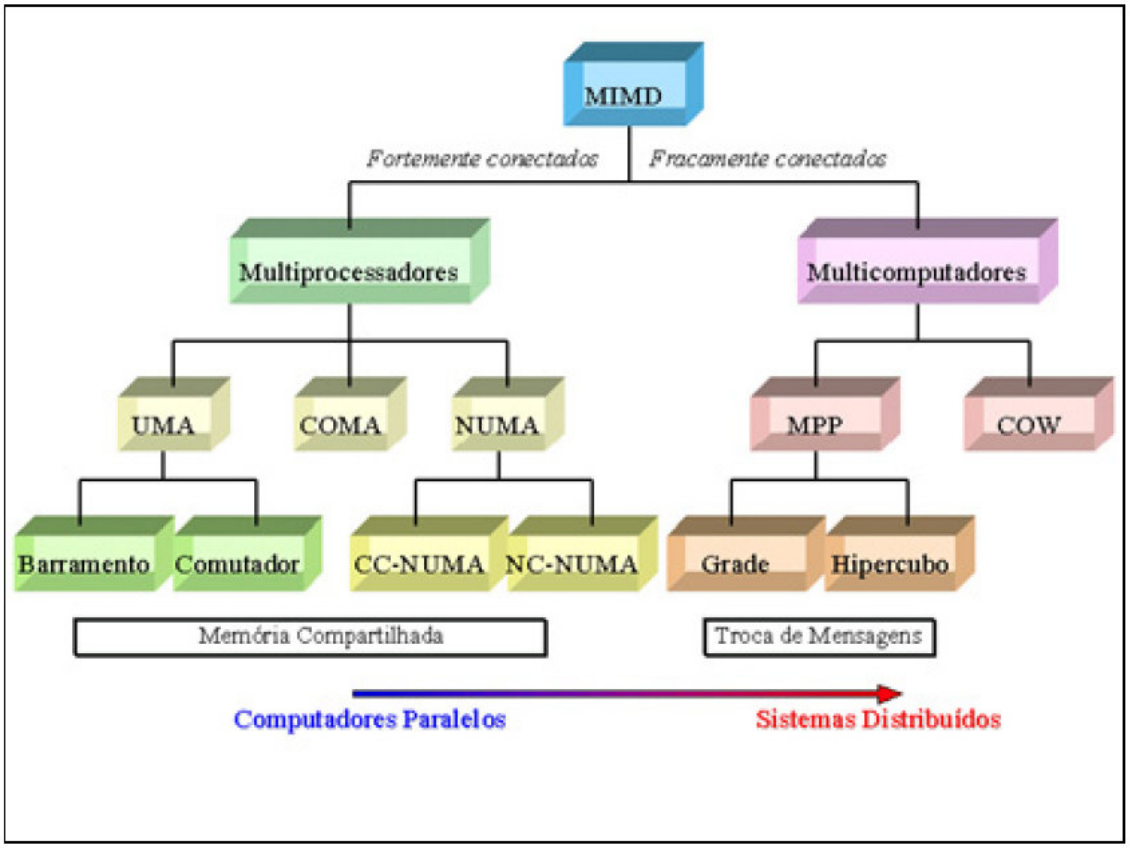


**Figura 3.7** – Classificação da arquitetura MIMD quanto ao acoplamento
Fonte: Sousa (2001)

Os **multiprocessadores**, também denominados de **sistemas fortemente acoplados,** ou ainda, arquiteturas de memória compartilhada, são caracterizados por processadores que dividem um mesmo espaço de endereçamento.

De acordo com o tipo de acesso à memória, os multiprocessadores são ainda divididos em máquinas UMA (*Uniform Memory Access*) ou NUMA (*Non-Uniform Memory Access*).

A **arquitetura UMA** é a mais comum, os processadores, a memória e os dispositivos de entrada e saída são conectados a um mesmo barramento sendo, portanto, compartilhados (Figura 3.8). A principal vantagem dessa

arquitetura é a sua simplicidade de concepção, mas ela apresenta como desvantagem o fato de que, aumentando o número de processadores, aumenta também a contenção de acesso a memória. Para limitar o efeito de contenção se utiliza memórias *cache*, porém isso introduz problema de coerência de dados entre as diferentes *caches* do sistema.

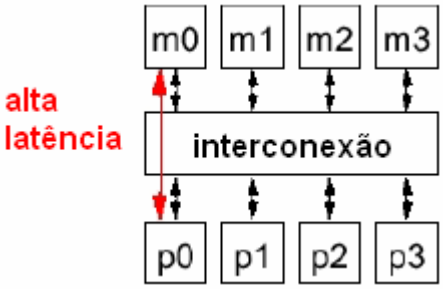


**Figura 3.8** – UMA – Tempos de acesso uniformes (p0→m0 = p0→m1 = p0→m2)
Fonte: Martins e Coentro (2005)

Esse tipo de arquitetura pode funcionar em módulos interconectados de duas formas:

- **Barramento**: Mecanismo mais simples, onde um canal é igualmente compartilhado por todos os processadores. Por questões de desempenho, pode-se inserir *caches* ou mesmo memórias locais (com cópias de alguns dados da memória compartilhada) em cada processador, o que acarreta uma considerável economia de acesso ao barramento e à memória central. Eventualmente, políticas para controle da coerência da memória devem ser empregadas;

- **Comutador**: Permite estender a quantidade de processadores em um sistema uniforme, graças ao chaveamento de regiões de memória (semelhante ao que ocorre em sistemas de telefonia para conectar linhas de entrada a saídas), criando uma rede de interconexão não-bloqueadora, isto é, diversos processadores podem simultaneamente fazer acesso a diferentes regiões de memória.

Em uma **arquitetura NUMA** cada processador possui uma memória local que forma com a memória local dos demais processadores um espaço de endereçamento único. Os acessos a endereços que correspondem às memórias "distantes" (as que pertencem a outros processadores) são mais lentos que os acesso à memória local, por isto a referência "acesso de memória não uniforme" (Figura 3.9). Nesses sistemas, é comum, por questões de performance, também se empregar *caches*.

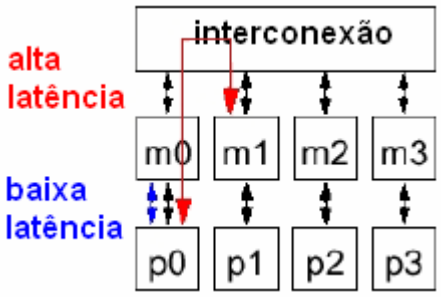


**Figura 3.9** – NUMA – Tempo de acesso p0→m1 maior que p0→m0
Fonte: Martins e Coentro (2005)

A NUMA pode ainda apresentar duas sub-divisões:

- **CC-NUMA:** Engloba máquinas que trabalham com **coerência de *cache*** a fim de manter a integridade e consistência dos dados. Utilizam o conceito de multiprocessador **baseado em diretório**, pois mantém em nível de *hardware*, uma base de dados informando sobre o estado de cada informação na *cache*.

- **NC-NUMA (NCC-NUMA):** Trabalham sem a necessidade de *cache* (o que torna dispensável a utilização de políticas para manutenção de coerência dos dados) e apresentam um tempo de acesso às memórias remotas conhecido, mas consideravelmente mais lento.

Na arquitetura **COMA (*Cache Only Memory Access*)** cada máquina tenta utilizar a memória dos demais como se fosse seu próprio *cache*, de forma que se busca obter uma velocidade de acesso bem superior, graças à elaboração de *hardware* especializado para a transferência de dados. Contudo, problemas relativos à localização e remoção dos dados, tornam a gerência da memória bem mais complexa, tornando necessários sistemas de coerência de cache.

No grupo de **multicomputadores**, também denominados de **sistemas fracamente acoplados**, ou ainda, arquiteturas de memória distribuída, são definidos por um conjunto de nós (computadores) independentes uns dos outros, isto é, cada nó possui seu próprio processador e sua própria memória. A comunicação entre os nós e necessariamente feita por troca de mensagens.

No caso dos multicomputadores, vale destacar que os computadores de memória distribuída não implementam em nível de *hardware*, nenhuma política para acesso às memórias remotas, de forma que tais processadores também são chamados de máquinas NORMA (*NO Remote Memory Access*). Globalmente, classificam-se em:

- **MPP (*Massively Parallel Processors*):** Computadores de memória distribuída formados por processadores independentes interligados por uma rede de interconexão (normalmente proprietária) extremamente rápida. Dentre as características dos MPPs, encontramos (HWANG e XU, 1998): alta escalabilidade de (atingindo patamares de até milhares de processadores); funcionamento assíncrono em nível de *hardware*, com mecanismos de sincronização por *software* (através de barreiras, por exemplo); comunicação via passagem de mensagens. Essa classe é ainda subdivida em hipercubos e grades.

    - Um **sistema de hipercubo** é uma rede de computadores (processadores) onde os processadores estão localizados nos

vértices do hipercubo e as comunicações entre processadores são as arestas do hipercubo.

- **Grades computacionais** são uma rede de computadores na qual um usuário se conecta para obter serviços computacionais que agregam recursos sob demanda (exemplo: ciclos, armazenamento, *software*, periféricos etc).

- **COW (*Cluster Of Workstations*):** Neste caso, cada computador é uma estação de trabalho independente (inclusive em nível de *software*), com memória e discos próprios e conectando-se a outros computadores através de uma rede de baixo custo (como Ethernet ou ATM).

Os *clusters* podem ser classificados de quatro tipos: Alta Disponibilidade (HA - *High Availability*), Balanceamento de Cargas, Combinação Alta Disponibilidade e Balanceamento de Carga e, por fim, Processamento Distribuído ou Processamento Paralelo (*HPC – High Performance Computing*).

O *cluster* de Alta Disponibilidade tem a finalidade de manter determinados serviços, prestados pelo sistema operacional, disponíveis de forma segura o maior tempo possível. Isso é possível replicando-se serviços e servidores, através da redundância de *hardware* e configuração de *software*. Diversos computadores juntos agindo como um só. Cada um monitorando os outros e assumindo seus serviços caso algum deles venha a falhar (Figura 3.10). (YOKOKURA, 2005).

**Figura 3.10** – Esquema de Alta Disponibilidade
Fonte: Yokokura (2005)

A complexidade do sistema deve estar no *software* que deve se preocupar em monitorar outras máquinas de uma rede, saber que serviços estão sendo executados, quem os está executando e como proceder em caso de uma falha. Perdas na performance ou na capacidade de processamento são normalmente aceitáveis; o objetivo principal é não parar. Existem algumas exceções, como sistemas de tempo real e de missão crítica.

A tolerância a falhas é conseguida através de *hardware*, como fontes e placas redundantes, sistemas rede totalmente ligados para prover caminhos alternativos na quebra de um *link*.

Já os *clusters* baseados em Balanceamento de Cargas tem como objetivo integrar seus computadores para que todas as requisições provenientes dos clientes sejam distribuídas de maneira equilibrada entre eles. A principal tarefa é redirecionar essas requisições de entrada por meio de um elemento que fará o balanceamento entre os servidores e os usuários. Portanto, nessa estrutura ter-se-iam múltiplos servidores, mas que parecem ser somente um para o cliente (Figura 3.11). Sua aplicação está relacionada à serviços de comércio eletrônico e provedores de Internet, que necessitam resolver diferenças de

cargas provenientes de múltiplas requisições de entrada em tempo real (YOKOKURA, 2005).

**Figura 3.11** – Exemplo de cluster para balanceamento de cargas
Fonte: Yokokura (2005)

A Combinação de Alta Disponibilidade e Balanceamento de Cargas, como o próprio nome diz, é composta pelos dois tipos de *clusters*, aumentando assim a disponibilidade e escalabilidade dos serviços e recursos. Esse tipo de configuração de *cluster* é bastante utilizado em servidores de *web*, *mail*, *news* ou *ftp*.

Um *cluster* de Processamento Distribuído ou Processamento Paralelo visa obter um alto desempenho de processamento na execução de aplicações paralelas ou seqüenciais. O ambiente computacional é concebido para escalonar uma grande quantidade de tarefas a serem executadas de forma paralela. A finalidade principal é ser uma alternativa a supercomputadores comerciais de alto custo e melhorar a utilização dos computadores que possuem uma carga baixa de processamento, distribuindo as tarefas entre os nós para obter resultados no menor tempo possível (YOKOKURA, 2005). Sua utilização está mais relacionada a resoluções de problemas tipicamente paralelos, ou seja, aplicações desenvolvidas para processarem cálculos de forma paralela.

### 3.2.2 Vantagens e Desvantagens

Inúmeras são as vantagens em se utilizar *clusters* para obtenção de um melhor desempenho, conforme as características sistêmicas apresentadas anteriormente. Dentre essas vantagens, pode-se destacar algumas:

- **alto desempenho**: possibilidade de se resolver problemas complexos em tempo reduzido;
- **escalabilidade**: possibilidade de inclusão de novos componentes à medida que a carga de trabalho cresce;
- **tolerância à falhas**: permanência de funcionamento na ocorrência de falhas;
- **baixo custo**: redução de custo para obtenção de processamento de alto desempenho utilizando computadores pessoais; e
- **independência de fornecedores**: utilização de *hardware* aberto, *software* de uso livre e independência de fabricantes e licenças de uso.

De acordo com Patterson e Hennessy (2000), uma das desvantagens do *cluster* é o fato do custo de administração de um sistema composto por *N* máquinas ser equivalente ao da administração de *N* máquinas independentes, ao passo que, no caso de uma máquina multiprocessadora com espaço de memória compartilhado com *N* processadores, seu custo de administração equivale ao de uma única máquina independe.

Outra desvantagem dos *clusters* é o fato de serem, em geral, conectados usando-se o barramento de entrada/saída de um computador, enquanto os multiprocessadores são ligados ao barramento de memória.

Esse último barramento tem uma alta largura de banda, permitindo que os multiprocessadores usem interconexões de rede a uma velocidade mais alta e com uso intensivo de entrada/saída.

Uma última fragilidade dos *clusters* é a divisão da memória. Um agrupamento de *N* máquinas tem *N* memórias independentes e *N* cópias do sistema operacional, mas uma máquina multiprocessadora com memória compartilhada permite que um único programa use quase toda a memória do computador. Portanto, um programa seqüencial executando em um agrupamento tem *1/N-ésimo* da memória disponível para ele quando comparado com um programa seqüencial que executa em um SMP.

Dada a abordagem geral da evolução dos sistemas computacionais, as próximas seções apresentarão metodologias para prever e avaliar o desempenho de um sistema, através de técnicas de avaliação de desempenho ou modelagem matemática.

## 3.3 Modelagem de Sistemas

Dentre as arquiteturas paralelas citadas anteriormente, as redes de estações de trabalho (*NOWs – Network Of Workstations*) ou *clusters* têm se destacado nos últimos anos como a mais eficiente em termos de custo e desempenho. A construção de sistemas de alto desempenho através da interligação de estações de trabalho com a utilização de redes de comunicação de alta velocidade, aliada aos recentes avanços da tecnologia de circuitos integrados e ao baixo custo destes componentes, fez com que a maioria dos fabricantes, inclusive brasileiros, optasse pelos *clusters* para a execução de aplicações paralelas (PFISTER, 1998).

A essa evolução das estações de trabalho NOW, deu-se o nome de Máquinas Agregadas (COW - *Clusters of Workstations*). Elas também são constituídas por várias estações de trabalho interligadas, mas com a diferença

de terem sido projetadas com o objetivo de executar aplicações paralelas. Dessa forma, a máquina pode ser otimizada para esse fim, e na maioria dos casos, as estações que servem de nó não possuem monitor, teclado e mouse, sendo denominadas "estação de trabalho sem cabeça" (*headless workstation*) (Figura 3.12).

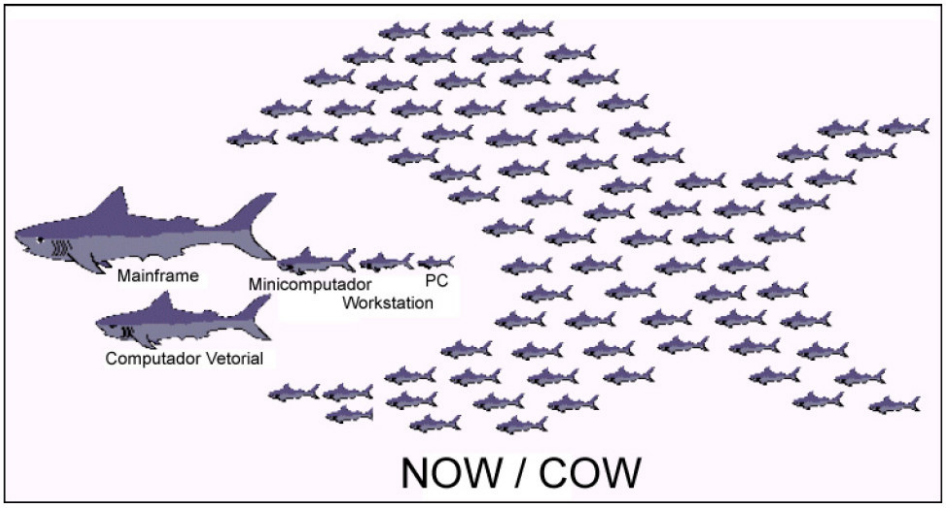


**Figura 3.12** – Representação de NOW/COW
Fonte: Sousa (2001)

Normalmente os programas paralelos apresentam um comportamento não esperado, devido a diversos fatores como a estruturação interna das aplicações paralelas, o sistema paralelo no qual é executado, o número de processadores/nós de processamento usado, os dados de entrada do programa e à medida de desempenho adotada (CROVELLA, 1994). Além disto, um escalonamento efetivo das aplicações e a adoção de uma estratégia para o balanceamento de carga são outros aspectos cruciais para alcançar um bom desempenho em uma rede de estações de trabalho.

Para compreender os fatores que influenciam no desempenho, duas abordagens distintas são consideradas, uma no nível de aplicação e outra no nível do sistema (MEIRA JR., 1995).

A primeira abordagem compreende um conjunto sistemático de passos que conduz o estudo da aplicação. Inicialmente, deve-se executar o programa

sobre o ambiente de teste (testes experimentais) e adquirir medidas de desempenho. Em seguida, baseado nos resultados obtidos identifica-se as potenciais alterações sobre o código do programa para melhorar seu desempenho.

Depois de modificar o programa, ele é executado novamente sobre o ambiente, repetindo-se o ciclo descrito, até que o desempenho alcançado seja satisfatório.

Nessa abordagem é mais complexa a identificação das razões ou os fatores que influenciam o desempenho das aplicações. Isso acontece, porque é difícil estabelecer uma relação entre o código do programa e os fatores externos inerentes às características do ambiente. Esses fatores podem estar relacionados com o sistema operacional utilizado, o tipo de memória do sistema, os adaptadores de redes, entre outros.

Na segunda abordagem, modelos matemáticos são elaborados para representar o comportamento do programa. As características do sistema paralelo são abstraídas através de um conjunto de funções e parâmetros. Essa abordagem, além de utilizar modelos matemáticos, pode fazer uso de uma representação gráfica (modelos baseados em grafos) para ilustrar a estrutura estática do programa, os relacionamentos e as dependências existentes entre os componentes da aplicação.

Essa abordagem permite um estudo mais detalhado sobre os fatores que contribuem no desempenho da aplicação. Esses fatores podem ser representados através dos parâmetros existentes nas funções elaboradas. Os modelos desenvolvidos não só permitem analisar o desempenho da aplicação, mas também predizê-lo em situações hipotéticas. Uma vez elaborado o modelo do programa, pode-se estimar o valor de algumas variáveis como, por exemplo, o tempo de execução do programa considerando uma outra quantidade de processadores envolvidos no processamento.

Apesar de se saber que a construção de modelos precisos e simples não é uma tarefa fácil, essa abordagem consegue desenvolver modelos de predição mais eficientes. Por esses motivos, ela vem sendo muito utilizada no estudo de desempenho de programas paralelos (MEIRA JR., 1995).

Segundo Meira Jr. (1995), pode-se seguir duas estratégias distintas durante a elaboração dos modelos dos programas. A escolha por uma delas depende da intenção da pessoa que desenvolverá o estudo de desempenho. Essas estratégias são:

1) ***Bottom-up***: o sistema é representado através de um conjunto de funções e parâmetros. Utiliza-se essa estratégia quando se pretende investigar a influência de fatores inerentes ao ambiente no desempenho da aplicação. A modelagem analítica utiliza esse tipo de estratégia.

2) ***Top-Down***: considera os aspectos relacionados à implementação da aplicação (código do programa). Esses aspectos incluem o nível de paralelismo empregado durante a implementação das tarefas. Modelos baseados em descrições e análise estatística são exemplos de técnicas que fazem parte dessa estratégia.

Nas próximas seções, são descritas as técnicas mais comuns para modelagem e avaliação de desempenho de sistemas.

## 3.4 Técnicas de Avaliação de Desempenho

Basicamente, as técnicas de avaliação de desempenho são utilizadas em duas situações distintas. A primeira delas é quando se está interessado em comparar sistemas diferentes, mas de um mesmo propósito, e descobrir qual deles é mais eficiente. A segunda situação é quando se deseja estudar alguns parâmetros do sistema e descobrir qual o melhor valor para cada um deles, de modo que o desempenho da aplicação seja melhorado (JAIN, 1991).

Algumas técnicas de avaliação são utilizadas com maior freqüência para esse propósito, dentre elas destacam-se as medições, as simulações, as modelagens analítica, estrutural e híbrida.

Alguns fatores importantes devem ser, previamente, considerados para se determinar a técnica mais adequada aos objetivos desejados. Uma relação desses fatores pode ser encontrada em Jain (1991). Basicamente, eles estão relacionados com o estágio de desenvolvimento da aplicação, a precisão desejada do modelo a ser desenvolvido e o custo de execução da técnica escolhida.

Apesar da relevância individual de cada fator, deve-se atribuir um peso maior ao estágio de desenvolvimento da aplicação. Dependendo desse estágio, não se pode escolher determinada técnica. Por exemplo, não é possível realizar medições de desempenho em um sistema se o mesmo não estiver totalmente implementado.

Dependendo do processo de avaliação desejado e o tipo do sistema em consideração, as informações requeridas por um estudo podem ser obtidas a partir do próprio sistema (técnicas de aferição), voltadas para sistemas já existentes ou em sua fase final de desenvolvimento ou, então, por meio de um modelo representativo do sistema (técnicas de modelagem) (SANTANA et al, 1994), que podem predizer o desempenho de um sistema de forma a analisá-lo, a priori, antes de ele estar pronto. A divisão de algumas dessas técnicas é apresentada na Figura 3.13.

| TÉCNICAS DE AFERIÇÃO | TÉCNICAS DE MODELAGEM |
|---|---|
| Coleta de Dados | Análise Probabilística |
| *Benchmarks* | Análise Determinística |
| Protótipos | Simulação |

**Figura 3.13 –** Técnicas de Avaliação de Desempenho de Sistemas Computacionais

### 3.4.1 Técnicas de Aferição

Essas técnicas são utilizadas para a depuração de sistemas já desenvolvidos ou em fase final de desenvolvimento.

Existem duas grandes razões para a utilização dessas técnicas: ajuste ou aprimoramento de sistemas.

A partir de valores medidos entre execuções consecutivas de um programa, pode-se perceber a variação do comportamento do desempenho da aplicação e do sistema em questão. Durante os testes experimentais, são medidos os valores de algumas variáveis diretamente relacionadas com o desempenho do sistema, como por exemplo, o tempo de execução. Entre uma repetição e outra, são realizados ajustes da aplicação se o código fonte e os algoritmos ainda puderem ser alterados, visando melhorá-lo. Esse processo se repete até que o resultado obtido com as alterações seja satisfatório. Essencialmente, essa técnica visa encontrar informações ou padrões relacionados com a aplicação que poderão ser ajustados para melhorar o desempenho do sistema.

Entretanto, duas dificuldades são apresentadas na experimentação do sistema: a) como fazer a experimentação sem influenciar no comportamento do sistema; b) os altos custos para alterações e testes e dificuldade de execução quando há necessidade de verificação de diferentes alternativas para o sistema (AIZA, 2003).

As principais técnicas de aferição são as seguintes: coleta de dados, *benchmarks* e construção de protótipos. Essas técnicas são descritas a seguir.

#### 3.4.1.1 Coleta de Dados

Essa é uma das técnicas mais precisas em comparação com as demais e deve ser conduzida sobre sistemas reais, operando em condições normais e

submetidos a uma carga em particular, cujo objetivo é adquirir dados do sistema para uma posterior análise e comparação dos valores obtidos com os valores fornecidos por um modelo do mesmo sistema. Assim, o sistema pode ser modelado e resolvido por simulação ou métodos analíticos, e as entradas para esse modelo podem ser abstraídas dos dados coletados no sistema em operação (AIZA, 2003).

Essa técnica envolve a monitoração por algum *hardware* ou *software* específico. Um cuidado especial deve ser considerado, para que não haja interferência significativa nos resultados coletados devido às próprias instruções que envolvem essa técnica (ZANDY, 2004; WILLEBEEK-LEMAIR e REEVES, 1993; WU, 1997).

Os componentes responsáveis pela monitoração do sistema são os seguintes:

- **monitoração por *hardware***: esse tipo de monitoração envolve o emprego de um *hardware* específico para a coleta e análise dos dados;

- **monitoração por *software***: é normalmente realizada através de recursos arquiteturais do sistema, tais como instruções de interrupção, *breakpoints* etc. Pode monitorar características específicas do *software*, como, por exemplo, verificar a existência ou não de uma fila de espera associada a um recurso do sistema.

Esses monitores coletam estatísticas de desempenho, analisam os dados obtidos e apresentam os resultados alcançados (JAIN, 1991). Geralmente, os monitores utilizam comandos de linguagem de alto nível para desempenhar suas funções. As variáveis medidas podem variar de um sistema para outro. Por exemplo, em aplicações do tipo cliente/servidor pode-se estar interessado

em medir o tempo de resposta do servidor, enquanto que em um programa de multiplicação de matrizes deseja-se obter o tempo total de execução.

Portanto, dependendo das características da aplicação, é necessário escolher a variável a ser medida a fim de avaliar o desempenho do sistema.

#### *3.4.1.2 Benchmarks*

São utilizados para medição de padrões de desempenho, obtidos de sistemas já existentes. Quando da aquisição de novos equipamentos, é interessante que a avaliação de seu desempenho seja realizada sob as mesmas condições do equipamento anterior, para efeito de computação, ou seja, a mesma carga de trabalho deve ser executada sobre cada um deles. A carga de trabalho utilizada nesse procedimento (que pode ser uma carga qualquer do usuário ou um procedimento específico) é chamada de *benchmark* (BRUSCHI, 2002; NAIK, 1985; WEICKER, 1990).

Os *benchmarks* permitem a avaliação do sistema como um todo, visto que a execução de um programa real avalia todos os aspectos necessários, incluindo aqueles relacionados com a arquitetura do sistema, eficiência do compilador, sobrecarga do sistema operacional, sobrecarga de comunicação, identificando os pontos fortes e fracos de um sistema em particular ou de aplicações (COUTO, 2000; AIZA, 2003).

#### 3.4.1.3 Construção de Protótipos

O protótipo é uma versão simplificada de um sistema real. Pode representar uma solução atrativa para a avaliação de sistemas em desenvolvimento ou quando a observação do sistema completo não for viável. Porém, apresenta um maior custo em relação a outras técnicas de aferição. Além disto, essa técnica restringe-se ao sistema específico, não se podendo

testar diferentes alternativas, devido ao custo e às dificuldades na modificação do meio físico (SPOLON, 2001; AIZA, 2003).

Santana et al (1994), classifica o desenvolvimento de um protótipo em seis fases:

1) avaliação da adequabilidade da utilização do protótipo;
2) avaliação dos domínios funcionais e comportamentais do sistema e das características principais que devem ser consideradas;
3) desenvolvimento do protótipo;
4) teste do protótipo;
5) modificações e correções no protótipo (nesse estágio, o protótipo já representa uma idéia do sistema real e o desenvolvedor poderá inferir sobre suas características fundamentais e, se for o caso, fazer alterações no mesmo);
6) repetições das fases 4 e 5, interativamente, até que se obtenha toda a funcionalidade do sistema representada no protótipo.

### 3.4.2 Técnicas de Modelagem

Essas técnicas consistem na modelagem de um determinado sistema, de forma a representar as características essenciais do sistema a ser avaliado, visando à obtenção de informações que auxiliem na resolução de questões sobre o sistema. Uma vez definido o modelo, esse pode ser avaliado através de soluções analíticas ou por simulação (AIZA, 2003; COUTO, 2000).

O processo de modelagem é um processo com um certo teor matemático, composto por fases que são ao mesmo tempo independentes e harmônicas entre si. Estas etapas são apresentadas na Figura 3.14.

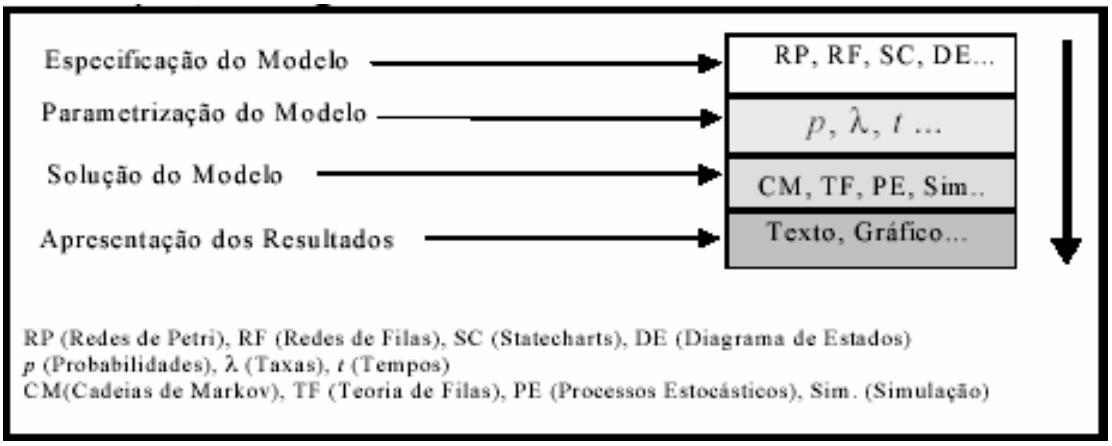


**Figura 3.14** – Etapas do Processo de Modelagem

Fonte: Francês (2001a)

Na fase inicial (modelagem), deve-se criar uma especificação condizente com o sistema real contendo os componentes relevantes à avaliação e à relação entre eles. A inclusão de muitos detalhes pode introduzir complicações desnecessárias, enquanto que a exclusão de características importantes pode invalidar o modelo (SOARES, 1992).

Algumas técnicas têm sido amplamente utilizadas para a representação do modelo, possuindo suas vantagens e desvantagens conforme o domínio de aplicação considerado, dentre elas: Redes de Filas (MACDOUGALL, 1987), Redes de Petri e *Statecharts* (PETRI, 1962), *Statecharts* Estocásticos (FRANCÊS, 2001a) e *Queuing Statecharts* (SANTANA et al, 1994; FRANCÊS et al, 2001b).

Após a confecção do modelo, com uma das técnicas citadas, deve-se parametrizá-lo com elementos que serão dados de entrada para a próxima fase (solução). Uma vez obtido o modelo, pode-se avaliá-lo através das soluções analíticas ou simulação. Por fim, a modelagem deve apresentar de forma conveniente seus resultados, utilizando-se de gráficos ou arquivos-texto (SANTANA et al, 1994).

Nas Seções 3.4.2.1, são discutidas as formas tradicionais para a especificação de modelos (Redes de Filas, Redes de Petri, *Statecharts* e *Queuing Statecharts*) e na Seção 3.4.2.2 suas soluções (Analíticas ou por Simulação).

#### 3.4.2.1 Técnicas para Especificação de Modelos

Para fins de avaliação de desempenho essas técnicas devem se basear em espaços de estados e eventos (interferências que causam mudanças de estados no sistema). Essa restrição deve-se à relação com os métodos de solução, os quais são fortemente baseados em estados e suas transições.

Cada uma das três técnicas de representação de modelos mais referenciadas na literatura (redes de fila, redes de Petri e *Statecharts*) têm suas vantagens e desvantagens quando analisadas sob a ótica do desempenho (FRANCÊS et al, 2001a).

##### 3.4.2.1.1 Rede de Filas

As Redes de Filas possuem uma base matemática bastante sólida, contudo sua representação gráfica oferece apenas os elementos fila e servidor, o que em muitos casos pode ser o suficiente. Entretanto, em alguns sistemas, deseja-se representar situações que não constituem necessariamente uma fila ou servidor. Por exemplo, um processo em um computador pode se encontrar em três possíveis situações: processando, bloqueado ou pronto.

Essas situações representam estados abstratos que são facilmente representados em técnicas como redes de Petri e *Statecharts*, mas por não consistirem em filas ou servidores, não são tratadas apropriadamente em redes de filas, mas necessitam de uma representação mais minuciosa,

através da qual possam ser especificadas as várias situações que tanto fila (uma fila pode estar vazia ou não) quanto servidor (um servidor pode estar livre ou ocupado) podem se encontrar em um determinado instante.

A despeito dessas limitações, a representação do caminho linear que os clientes traçam através do sistema é descrita com bastante propriedade, noção que na maioria das técnicas é perdida com facilidade (FRANCÊS et al, 2001b).

#### 3.4.2.1.2 Redes de Petri

As Redes de Petri possuem características bastante interessantes em sua representação. Uma das mais interessantes é a possibilidade da individualização de clientes, recurso que não é usual na maioria das técnicas.

Há situações em que essa característica é primordial, por exemplo, na migração de processos para balanceamento de carga entre máquinas, onde se pode desejar migrar exatamente um determinado processo e por isso deve-se saber exatamente onde ele se encontra. Entretanto, há uma certa dificuldade na representação de processos paralelos, mesmo utilizando-se da rede distribuição (rede elementar para representar a criação de processos paralelos).

Esse aspecto piora à medida que o modelo cresce, o que aliás é outro aspecto a ser ponderado. As redes originais, por não possuírem nenhum mecanismo de hierarquia, tendem a fazer com que os modelos cresçam substancialmente, quando a complexidade dos elementos aumenta (FRANCÊS et al, 2001b).

Há algumas extensões que tentam minimizar esse efeito, como as redes de petri hierárquicas, que são baseadas em um elemento denominado super-página (MACIEL, LINS e CUNHA, 1996). O “inconveniente” dessa extensão é que as super-páginas são caixas-pretas, ou seja, retângulos que escondem uma complexidade que, em certos casos, é necessária para a compreensão do

modelo. Além disso, algumas extensões desfiguram a notação original, o que, às vezes, parece ser outra técnica à parte, e não uma derivação das redes de Petri (FRANCÊS et al, 2001b).

#### 3.4.2.1.3 Statecharts

Na representação *Statecharts*, a forma explícita de algumas características (como paralelismo e a hierarquia entre estados) é um atrativo considerável. Essa peculiaridade permite que relacionamentos complexos entre componentes de um determinado sistema sejam mostrados de maneira mais efetiva, o que não é contemplado na maioria das técnicas.

Estados *default* também acrescentam a idéia de passo inicial do sistema, o que nas redes de Petri é realizado através da presença dos *tokens* (marcação inicial).

Entretanto, uma possibilidade que os *Statecharts* não contemplam é, quando há presença de estados ortogonais, a descrição explícita do caminho linear seguido por um determinado cliente (que é claro em Redes de Fila e em Redes de Petri), pois pode haver sub-estados ativos em um determinado momento, o que não significa que todos os clientes (ou um cliente em particular) tenham seguido por aquela trajetória. Além disso, individualização de clientes (como feito com a utilização dos *tokens*), fundamental em certas situações (tal como em escalonamento de processos, onde se pode decidir por migrar um processo em particular), não é implementada em *statecharts* (FRANCÊS et al, 2001b).

#### 3.4.2.2 Soluções para o Modelo

Uma vez definida a técnica a ser utilizada para a representação do modelo, deve-se escolher a solução a ser dada ao mesmo. As técnicas para solucionar um modelo são apresentadas a seguir.

##### 3.4.2.2.1 Modelagem Analítica

A modelagem analítica descreve a operação do sistema e sua carga em termos completamente abstratos. Os fatores que influenciam e interferem no comportamento do sistema são modelados e representados através dos parâmetros das equações matemáticas. Essas equações são chamadas de modelos analíticos. Apesar de esses modelos considerarem parâmetros específicos de uma arquitetura, podem ser facilmente adaptados para outras plataformas. As estimativas de desempenho são obtidas por meio da resolução analítica ou numérica do modelo matemático resultante.

A resolução analítica de um modelo apresenta resultados mais precisos, porém a dificuldade é proporcional à complexidade do modelo. É um método mais rápido, mas nem sempre aplicável. Muitas vezes, simplificações são introduzidas ao modelo e podem distorcer a representação real do sistema, fato que pode simplesmente tornar sem sentido a utilização do modelo (JAIN, 1991; SANTANA et al, 1994).

Há ainda algumas outras restrições impostas à solução analítica. Por exemplo, certas distribuições de probabilidade que possuem um tratamento matemático factível em certas situações são um fator limitante.

A modelagem analítica, quando comparada às demais técnicas de avaliação de desempenho, apresenta menor custo de execução. Para validar os resultados alcançados através dos modelos elaborados, a modelagem analítica pode compará-los aos valores reais medidos em testes experimentais.

Esses valores poderão comprovar as predições realizadas através dos modelos analíticos.

Nos casos em que várias simplificações (restrições) comprometem a exatidão dos resultados do modelo, pode ser mais conveniente adotar a solução por simulação (JAIN, 1991; SPOLON, 2001; AIZA, 2003).

Segundo Meira Jr. (1995), pode-se seguir três abordagens distintas ao realizar uma modelagem analítica. Essas abordagens são:

- **Modelagem com Parâmetros Escalares:** o comportamento do sistema é modelado através de um conjunto de parâmetros escalares, que caracteriza o comportamento do sistema sob algumas condições especificadas durante a sua elaboração. A precisão do modelo depende da quantidade de parâmetros considerados na modelagem. Alguns parâmetros, como *overhead* do sistema operacional, dificilmente são modelados e, se considerados, podem tornar o modelo muito complexo e difícil de ser aplicado.

- **Modelagem com Funções:** expressa a influência de parâmetros relacionados com o *software* e o *hardware* sobre o tempo de execução de aplicações. Apesar dessa abordagem possibilitar o desenvolvimento de modelos mais precisos, quando comparada à modelagem com parâmetros escalares, a complexidade desses modelos pode ser maior. Isso acontece, pois é necessário descobrir o tipo de função matemática (linear, quadrática, logarítmica, etc) que melhor representa o comportamento dinâmico do sistema. Além disso, é preciso reconhecer e modelar os parâmetros que serão utilizados pelas funções, para que o modelo seja consistente.

- **Modelos Estatísticos:** utiliza ferramentas de modelagem estatística, como os modelos de Markov, para caracterizar o comportamento do sistema paralelo. Nessa abordagem, os parâmetros definidos são ferramentas estatísticas. Normalmente, esses modelos são utilizados

para analisar o comportamento de sistemas que possuem seu *workload* ideal bem definido. Os parâmetros estatísticos são utilizados para representar o comportamento assintótico do sistema.

#### 3.4.2.2.2 Modelagem Baseada em Descrição

Essa modelagem está baseada em informações ou descrições oferecidas pelo usuário sobre a estrutura e o comportamento do sistema que está sendo modelado. Para a elaboração do modelo do sistema, a modelagem baseada em descrição utiliza uma representação gráfica auxiliar (baseada em grafos), denominada *task graph*, que indica os potenciais pontos de paralelização para elaboração de uma aplicação. Por outro lado, alguns parâmetros relacionados às características do *hardware* também são considerados durante a elaboração do modelo.

#### 3.4.2.2.3 Modelagem com Análise Estática

Nessa técnica, o ponto de partida para o processo de modelagem é a própria aplicação.

Essa modelagem obtém informações do sistema através de uma análise realizada pelo próprio compilador. Como a análise do programa acontece em tempo de compilação, dizemos que é uma modelagem estática. Existem ferramentas específicas para o desenvolvimento essa abordagem, no entanto, o alto custo associado inibe a sua aplicação.

#### 3.4.2.2.4 Modelagem Híbrida

Combina características de modelos distintos, utilizando conceitos e estratégias das diferentes técnicas de modelagem existentes. Essa abordagem vem sendo bastante utilizada e os resultados alcançados têm sido bastante satisfatórios.

#### 3.4.2.2.5 Simulação

A simulação é utilizada tanto em avaliação de desempenho, quanto na validação de modelos analíticos. Nesse caso, o sistema é abstraído em um modelo e, a partir desse, um programa de simulação é gerado (COUTO, 2000). Logo, ela não exige que o sistema esteja totalmente implementado para ser aplicada.

Durante uma simulação pode-se controlar com maior eficiência os valores assumidos por alguns parâmetros do sistema. Com isso, fica mais fácil obter informações relevantes para a avaliação de desempenho. Além disso, as simulações permitem criar modelos mais detalhados, gerando resultados mais precisos (SPOLON, 2001).

Através de simulações, pode-se verificar e prever o comportamento de programas sobre arquiteturas, ainda não utilizadas em testes. Para isso, existe uma técnica de simulação conhecida como *execution-driven*. Basicamente, essa técnica intercala execuções da aplicação na arquitetura real com simulações sobre a arquitetura desejada (HU e GORTON, 1997).

Nessa técnica, uma aplicação paralela é vista como uma composição de eventos globais separados por computações locais. Uma computação local é restrita a um processo específico e não pode ser vista nem influenciada por outros processos. Já um evento global, pode ser visto e influenciado por outros

processos. Esse tipo de evento pode ocasionar a mudança do caminho de execução de um programa paralelo.

As operações básicas de um evento global são o acesso à memória compartilhada e as atividades de sincronização (HU e GORTON, 1997). Apesar de ser uma técnica relativamente nova, tem se mostrado bastante eficiente.

Pode-se classificar uma simulação como determinística ou estocástica, de acordo com os dados fornecidos na entrada. A grande maioria dos sistemas a serem simulados são transformados em simulações estocásticas, nas quais os dados de entrada são gerados aleatoriamente a partir de uma distribuição de probabilidade.

Devido à aleatoriedade dos dados de entrada, a simulação precisa ser repetida várias vezes, de modo a garantir que as variáveis estimadas minimizem a influência das variáveis aleatórias. É onde simulação apresenta uma grande desvantagem, o tempo de execução da simulação (BRUSCHI, 2002).

A simulação tornou-se uma ferramenta atrativa e atuante para análise de desempenho de sistemas devido à sua versatilidade (pode ser usada em diferentes ocasiões), flexibilidade (é adaptável a novas e diferentes situações) e baixo custo (com um mesmo programa pode-se simular diferentes situações) (PEGDEN, SHANNON e SADOWSKI, 1995).

O processo de desenvolvimento de uma simulação envolve diversas etapas apresentadas na Figura 3.15.

1) **Formulação do problema:** descrição do sistema e suas características mais relevantes.

2) **Ajuste dos objetivos e planos:** apontam os objetivos que se deseja alcançar e indicam quais serão as questões que a simulação irá responder.

3) **Construção do modelo:** abstração do sistema em um modelo, o qual irá representar uma visão particular do mesmo.

4) **Coleta de da6dos:** definição de quais os tipos de dados a serem abordados no sistema. Essa etapa pode ser realizada paralelamente à construção do modelo.

5) **Codificação:** definição da linguagem e recursos a serem utilizados para descrever o comportamento do sistema.

Para a codificação do modelo, têm-se os seguintes enfoques (SANTANA, M.J, 1990; SANTANA, R.H.C., 1990):

- desenvolvimento do programa de simulação em uma linguagem convencional: essa abordagem apresenta como vantagens o fato do usuário não precisar aprender uma nova linguagem e de ser flexível. A desvantagem desta abordagem é a inexistência de ferramentas (estruturas de dados, coleta de estatísticas, emissão de relatórios, etc.) para auxiliar no desenvolvimento da simulação;

- utilização de um pacote de uso específico para o desenvolvimento do programa de simulação: normalmente são ambientes voltados para a avaliação de sistemas particulares. Possui a vantagem de ser de fácil utilização para o objetivo proposto, mas são pouco flexíveis;

- uso de linguagens de simulação de uso geral: são linguagens já projetadas para o desenvolvimento de uma simulação. Possuem a vantagem de livrar o programador de certos detalhes de implementação da simulação, mas apresenta a desvantagem de ter que aprender uma nova linguagem.

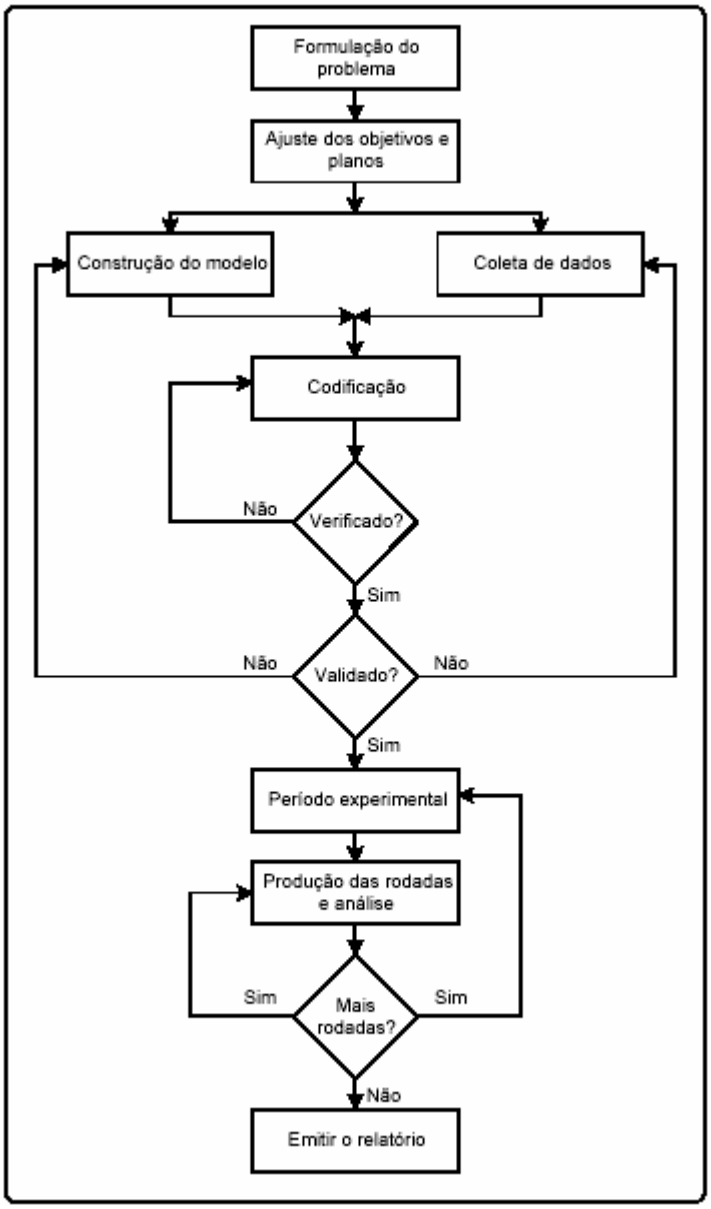


**Figura 3.15** – Etapas para o Desenvolvimento de uma Simulação
Fonte: Bruschi (2002)

- uso de uma extensão funcional de uma linguagem de programação de uso geral: são conjuntos de bibliotecas inseridas nas linguagens convencionais compondo um ambiente completo para o desenvolvimento da simulação. Une as vantagens das linguagens convencionais e das linguagens de simulação.

6) **Verificação:** consiste em verificar se o programa de simulação é uma implementação válida do modelo.

A simulação é um processo amostral, isso significa que é gerado um grande número de valores amostrais e utiliza-se a média amostral como uma estimativa da média real.

Conseqüentemente, algumas diferenças entre os resultados analíticos e da simulação podem ser resultado da variação amostral (MACDOUGALL, 1987).

Para resolver esse problema, pode-se conduzir uma análise adicional: coletar dados adicionais, computar limites de confiança para a estimativa da simulação e determinar se esse limite inclui o resultado obtido.

7) **Validação:** nessa fase deve-se demonstrar que o modelo proposto é uma representação do sistema a ser simulado respondendo a seguinte pergunta: "O modelo conceitual pode substituir o sistema real?" (BANKS, 1998).

A validação pode ser considerada em dois casos: quando o sistema simulado existe e pode ser medido e quando o sistema modelado não existe, tendo-se apenas o projeto do sistema a ser simulado.

No primeiro caso, o objetivo da análise é avaliar uma mudança proposta no sistema e a validação é baseada na comparação dos resultados do modelo com as medidas reais do sistema.

No segundo caso, o objetivo da análise é estimar o desempenho do projeto ou mesmo avaliar projetos alternativos. Neste caso, o modelo é validado baseando-se nos resultados esperados no projeto do sistema e cada suposição e abstração são justificadas.

Outra possibilidade para avaliação é através de comparações com outros modelos do projeto já validados, como por exemplo, modelos analíticos

(MACDOUGALL, 1987) ou através de situações em que os resultados já são conhecidos.

Uma das técnicas utilizadas para validar um modelo é *face validity* a qual consiste em verificar se o modelo satisfaz a necessidades dos usuários, isto é, estar de acordo com o que se espera do modelo, ou seja, os resultados obtidos devem ser consistentes com o sistema modelado. Uma maneira de realizar *face validity* é a utilização de uma análise de sensibilidade sobre as variáveis de entrada (BANKS et al, 2001). Uma estrutura para verificação e validação é proposta por Carson (2002):

- teste do modelo para *Face Validity;*
- teste do modelo para um número de parâmetros de entrada;
- quando possível, pode-se comparar as predições do modelo através de um modelo base representando o sistema, ou por desempenhos obtidos no sistema atual. Já no projeto de um novo sistema, pode-se usar hipóteses e especificações para efetuar a comparação com relação ao comportamento do modelo.

8) **Período experimental:** são definidos os parâmetros com relação a simulação de maneira a se obter as informações desejadas a um custo mínimo.

9) **Produção das execuções e análise:** As execuções iniciam e os resultados a serem analisados começam a ser coletados. Em simulações estocásticas, como as entradas de uma simulação são baseadas na geração de valores aleatórios e as saídas são funções dessas variáveis de entrada, a análise dos resultados de uma simulação é um problema estatístico. Os métodos para análise destes resultados baseiam-se em estimar o intervalo de confiança dos valores médios obtidos na simulação (MACDOUGALL, 1987). Isto quer dizer que a simulação pára quando a estimativa da variância da média está dentro de uma certa tolerância especificada (intervalo de confiança). Se a precisão não for atingida é necessária uma maior número de

execuções. Entre os métodos para análise de saída pode-se destacar os métodos regenerativos, por replicação e *batch means* (BANKS et a., 2001; BRUSCHI, 2002).

10) **Emissão do relatório:** Os resultados desejados e obtidos com a simulação devem ser fornecidos ao usuário de uma forma clara e concisa.

## 3.5 Predição de Desempenho

Entende-se por predição de desempenho o ato de estimar valores aproximados de algumas variáveis inerentes à aplicação como, por exemplo, o tempo de execução do programa que está sendo avaliado. Dessa forma, a atividade de predição de desempenho consiste em realizar estudos que permitem antecipar e predizer o desempenho de um sistema sob determinadas condições previamente, estabelecidas (LAINE, 2003).

Os métodos de predição existentes têm usado, freqüentemente, um modelo de representação baseado em dois níveis. O primeiro nível considera aspectos relacionados com a aplicação como, por exemplo, o grau de paralelismo explorado no programa. Já os aspectos relacionados com o ambiente, como o número de processadores utilizados, são considerados no segundo nível de representação (YAN e ZHANG, 1996).

Li (2001) também utiliza uma representação de dois níveis para separar os fatores internos e externos que contribuem para o desempenho do sistema. Os fatores internos podem ser o tipo de paralelismo utilizado, os métodos de sincronização escolhidos, entre outros. Como fatores externos, pode-se citar o número de processadores e as características das máquinas utilizadas.

Antes de iniciar qualquer predição de desempenho, deve-se construir um modelo capaz de representar com precisão as características da aplicação, seguindo algumas das técnicas descritas nas seções a seguir. Dependendo do

grau de abstração que se pretende explorar, o modelo desenvolvido pode ser simplesmente uma representação gráfica de um programa, elaborado através de grafos orientados. No entanto, um modelo mais detalhado pode ser expressões matemáticas que descrevem o comportamento do programa em função de alguns parâmetros de entrada.

## 3.6 Considerações Gerais

Nesse capítulo foram apresentadas as principais características sobre sistemas de modelagem, técnicas aplicadas na análise de desempenho computacional e em sistemas cuja performance se deseja avaliar a um menor custo e com uma maior eficiência.

As técnicas de aferição são indicadas quando o sistema a ser avaliado já existe ou está em fase final de desenvolvimento. A construção de protótipos é mais indicada para a avaliação de desempenho de sistemas que se encontram na fase final de desenvolvimento.

Os *benchmarks* tem o seu uso preferencial para avaliar o desempenho, para decidir sobre a seleção de novos sistemas. A coleta de dados é a técnica de aferição que oferece os resultados mais precisos. Esta técnica é recomendada para os casos em que o sistema já exista.

A técnica de aferição é a que oferece os resultados mais precisos. Porém, quando não existir o sistema, ou nos casos em que se deseja uma avaliação de desempenho sem que haja a influência da própria aferição no resultado da avaliação, usualmente, toma-se uma abstração do sistema, levando-se em consideração as características mais essenciais do sistema, denominada modelo, a partir do qual são realizadas certas inferências, sem a necessidade de realizar-se a efetiva experimentação do sistema. Dentre as técnicas para representação de modelos existentes pode-se destacar as redes de filas, redes de Petri e *Statecharts*. As redes de filas é uma técnica bastante utilizada para

avaliação de sistemas computacionais, pois estuda o fenômeno da formação de filas.

Após a escolha da técnica para representação de modelos a ser empregada deve-se analisar qual a solução do modelo a ser empregada: analítica ou simulação. Para modelos menos complexos é mais recomendável a utilização de soluções analíticas, pois podem admitir certas simplificações no modelo. Porém, em sistemas mais complexos, é mais recomendável o uso de simulação. Pois, a simulação é uma técnica que considera um modelo mais apurado do sistema, sendo possível especificar mais detalhes dentro do modelo.

Neste trabalho é utilizada a solução por simulação, visto que é capaz de lidar com sistemas e ambientes mais complexos e que qualquer alteração é refletida pelo programa. A simulação permitirá avaliar o desempenho do TLBA, diante da variação dos parâmetros de balanceamento de carga e da ampliação da heterogeneidade ao qual ele se aplica.

A validação do TLBA foi efetuada por meio de três abordagens. A primeira através da cuidadosa comparação do modelo identificando as características importantes para sua construção. A segunda na avaliação através de comparações com o modelo do projeto já validado por Mello (2003). Na terceira abordagem utilizou-se a técnica de *face validity*, na qual os resultados obtidos com a simulação são comparados com o comportamento esperado do sistema.

Capítulo

4

# 4 ESTUDO SOBRE A VARIAÇÃO DOS PARÂMETROS DO *TREE LOAD BALANCING ALGORITHM*

## 4.1 Introdução

Neste trabalho é utilizado o algoritmo de balanceamento de carga denominado TLBA (*Tree Load Balancing Algorithm*), proposto por Mello (2003) e descrito anteriormente (Capítulo 2). O TLBA foi projetado de forma a organizar computadores de um determinado sistema em uma topologia lógica na forma de árvore, independentemente da topologia física da rede de computadores, sobre a qual são executadas operações de balanceamento de carga.

Este trabalho amplia os estudos sobre o TLBA, com a elaboração de um novo simulador e apresenta diferentes resultados obtidos na simulação através da variação de seus parâmetros e aplicação em ambientes homogêneos e heterogêneos mais diversificados. Ele visa suprir algumas das limitações e características do TLBA apresentadas a seguir.

Nesse capítulo são analisados os parâmetros de balanceamento de carga do TLBA que têm influência direta no seu desempenho, os fluxogramas das políticas de balanceamento de carga do TLBA, a descrição do novo simulador e estudos de casos envolvendo ambientes homogêneos e heterogêneos.

## 4.2 Características e Limitações do TLBA

Em estudos prévios sobre o TLBA (MELLO, 2003; MELLO, TREVELIN e PAIVA, 2003a e 2003b) foi comprovada a sua eficiência para ambientes heterogêneos e altamente escaláveis através de um simulador e um sistema protótipo.

No entanto, algumas características e limitações foram evidenciadas e motivaram a elaboração deste trabalho para o estudo dos parâmetros do TLBA aplicando-se novos conceitos e complementando os estudos prévios. Algumas de suas limitações são descritas a seguir:

- a função de distribuição de probabilidade adotada por Mello (2003), foi a distribuição de Poisson com média de 100s. Atualmente, a maioria dos trabalhos apresentados utilizam-se do método de Feitelson com distribuição de Hiperexponencial com média de 1500 segundos como modelo de carregamento de cargas em sistema distribuido;

- foram analisadas apenas duas capacidades de processamento heterogêneas. Para melhor verificar o comportamento desse algorimo em ambiente heterogêneos, é necessário um ampliação da aplicação desse algoritmo para ambientes hereterôgenos.

- o número de repetições realizada pelo simulador para a análise de precisão do tempo médio de resposta foi de 100 vezes. Esse número não garante, realmente, a coerência e a precisão dos resultados. Dessa forma, neste trabalho foram utilizados intervalos de confiança e o número de repetições ficou condicionado à obtenção de um resultado dentro do intervalo desejado.

- não foi feita uma nenhuma análise das característas do comportamento da aplicação, quando se tem *CPU Bound, I/O Bound, Network Bound* etc.

- os custos com relação à migração dos processos não foram levados totalmente em conta. Esses custos estão relacionados à operação de transferência do contexto de um processo em uma rede de computadores e o restabelecimento dos canais de comunicação existentes entre os processos (HALCHOR-BALTER e DOWNEY, 1997; MELLO e SENGER, 2004).

Com relação ao nível de protótipo do algoritmo do TLBA, esse algoritmo também apresentou algumas limitações de acesso ao volume NFS, acesso de base de dados, submissão de processos e recuperação dos canais de comunicação, cujas descrições e análises podem ser encontradas em Mello (2003) e não serão consideradas neste trabalho.

## 4.3 Parâmetros de Balanceamento de Carga do TLBA

No levantamento dos parâmetros relacionados ao TLBA, foi feito um levantamento dos que influenciariam de alguma forma no desempenho do algoritmo, quando aplicado em sistemas distribuídos. Esses parâmetros são os índices de carga com base no cálculo do percentual de ocupação de Memória e CPU, *Thresholds* (para fins de atualização de informações de carga e de migração de processos), número de níveis aplicados à árvore de computadores, número de processos, número de computadores (nós) e variação das capacidades computacionais em meios homogêneos ou heterogêneos. Suas descrições são apresentadas nas Seções seguintes.

Alguns desses parâmetros também podem ser encontrados em outros algoritmos de balanceamento de carga (SHIVARATRI, KRUEGER e SINGHAL, 1992; THEIMER e LANTZ, 1989; ZHOU e FERRARI, 1987). De acordo com Livny e Melmann (1982), a obtenção dessas informações promove uma melhora significativa no desempenho do sistema computacional.

Em balanceamento de carga há duas métricas importantes: o tempo médio de resposta dos processos submetidos ao sistema e o número de mensagens trocadas nas operações de balanceamento de carga (JAIN, 1991; MELLO, 2003; MELLO, TREVELIN e PAIVA, 2003b; SHIVARATRI, KRUEGER e SINGHAL, 1992).

O tempo médio de resposta dos processos influencia o desempenho final do sistema. Quanto menor o tempo de resposta, maior o desempenho. O tempo de resposta mede o tempo consumido por um processo no sistema, do início ao fim de sua execução (SHIVARATRI, KRUEGER e SINGHAL, 1992; ZHOU e FERRARI, 1987).

Quanto ao número de mensagens trocadas nas operações de balanceamento de carga, esse não pode ser alto, pois sobrecarrega o meio de comunicação. Há sistemas que atingem bom desempenho para baixas cargas, mas em altas cargas o desempenho é degradado, principalmente pelo alto número de mensagens geradas no meio de comunicação (SHIVARATRI, KRUEGER e SINGHAL, 1992; ZHOU e FERRARI, 1987).

A seguir são apresentadas as descrições dos parâmetros de balanceamento de carga que possuem influencia significativa no desempenho do sistema.

### 4.3.1 Índices de Carga da Memória e da CPU

Um dos principais elementos para a análise dos algoritmos dinâmicos de balanceamento de carga é o índice de carga (SHIVARATRI, KRUEGER e SINGHAL, 1992). Ele é parte integrante da política de informação do algoritmo e permite medir e comparar a carga dos computadores, independentemente do *hardware*. É o responsável pela determinação de computadores ociosos e sobrecarregados do sistema.

A grande maioria dos trabalhos sobre balanceamento de carga utiliza como índice de carga o comprimento da fila de processos em espera pela CPU (SHIVARATRI, KRUEGER e SINGHAL, 1992). Essa técnica não é efetiva quando se consideram ambientes que possuem capacidade de processamento heterogênea e os processos não têm ocupação semelhante (MELLO, TREVELIN e PAIVA, 2003a, 2003b).

No trabalho apresentado por Mello (2003), foi proposto um índice de carga baseado na análise de ocupação percentual da CPU e da Memória, apresentando melhores resultados em relação ao algoritmo *Lowest* modificado (MELLO, TREVELIN e PAIVA, 2003b) e aos Algoritmos Simetricamente Iniciados (SHIVARATRI, KRUEGER e SINGHAL, 1992).

Neste trabalho foi utilizado o índice de carga proposto por Mello (2003), que utiliza o percentual de ocupação da CPU e Memória para o cálculo do índice de carga do sistema.

O cálculo do índice de carga é dado através da análise das características computacionais de cada computador para descobrir o seu estado em um determinado momento e verificar a necessidade de migração de seus processos ou de novos processos para outras máquinas mais ociosas.

Uma análise em termos práticos é feita através das eqs.(1) e (2), conforme definido por Mello (2003), que permitem, respectivamente, a avaliação da ocupação dos recursos de CPU e memória pelo TLBA.

Na eq.(1) é possível obter a percentagem de ocupação de recurso de CPU, onde $\lambda$ é o *quantum* total de cada processo no sistema, $\mu$ é o tempo relativo de $\lambda$ em que a CPU manteve-se utilizada e *p* é o número de processos do sistema (MELLO, 2003).

$$O_{cpu} = \frac{\left( \sum_{i=0}^{p} \frac{\mu_i}{\lambda} \right)}{p} \qquad (1)$$

Com a eq.(2) obtém-se a percentagem de ocupação de recurso de memória principal, onde θ é a quantidade total de memória principal, σ é a quantidade de memória principal utilizada por um processo e p é o número de processos do sistema (MELLO, 2003).

$$O_{mem} = \frac{\left(\sum_{i=0}^{p} \sigma_i\right)}{\theta} \tag{2}$$

As partir das eqs. (1) e (2), obtém-se uma expressão para o cálculo do índice de carga ($I_{comp}$) de um computador e que é apresentada na eq.(3), onde:

a) $O_{cpu}$ é a ocupação percentual do recurso de CPU; e

b) $O_{mem}$ é a ocupação percentual do recurso de memória principal.

$$I_{comp} = \frac{(O_{cpu} + O_{mem})}{2} \tag{3}$$

Nesse trabalho, foi analisado o índice de carga baseado na ocupação percentual dos recursos de CPU e Memória, em função de diferentes números de processos, computadores, *Thresholds* etc.

O índice de carga é um dos principais parâmetros para o balanceamento de carga determinando a necessidade de transferência de processos ou ajustes de outros parâmetros adicionais de balanceamento de carga para melhor desempenho de um sistema.

Dentre os parâmetros adicionais tem-se os *Thresholds* de Atualização e Viabilidade de Transferência de Carga, utilizados pelas políticas de balanceamento para coletar informações do sistema e verificar a necessidade de transferência de um processo. Esses parâmetros são apresentados nas seções a seguir.

### 4.3.2 *Threshold* de Atualização de Carga

Esse parâmetro é utilizado pela política de informação do TLBA, que decidirá quando as informações de outros computadores do sistema serão coletadas (MELLO, 2003). Assim sendo, há uma diminuição da atualização dos índices de cargas locais e, conseqüentemente, um menor número de mensagens no meio de comunicação, garantindo a estabilidade do sistema (SHIVARATRI, KRUEGER e SINGHAL, 1992).

Em seguida é apresentada a eq.(4), envolvendo o *Threshold* de atualização de carga, para determinação das variações superiores e inferiores do índice de carga (MELLO, 2003)..

$$V_s = I_{comp-1} + \left(I_{comp-1}\delta\right) \; e \; V_i = I_{comp-1} - \left(I_{comp-1}\delta\right) \tag{4}$$

Onde:

a) *V* é caracterizado como variação superior ($V_s$) e inferior ($V_i$) do índice de carga, quando se tem a adição e subtração, respectivamente, dos membros da equação;

b) $\delta$ é definido como *Threshold* de Atualização, que influencia a variação percentual mínima da eq.(4), para que o índice de carga $I_{comp}$ seja considerado alterado em relação ao seu anterior $I_{comp-1}$.

A atualização do índice de carga é feita conforme a eq.(5). Caso uma das desigualdades seja confirmada, o índice de carga atualizado é enviado para o computador antecessor na árvore. Um computador que não tem sucessores somente submete seu índice $I_{comp}$ (MELLO, 2003).

$$se \quad I_{comp} > V_s \quad ou \quad I_{comp} < V_i \tag{5}$$

A variação do *Threshold* de Atualização foi avaliada no presente trabalho, com o objetivo de verificar sua influência no desempenho e no número de mensagens geradas no sistema.

### 4.3.3 *Threshold* para Analisar Viabilidade de Transferência

Um outro *Threshold*, denominado *Threshold* de Viabilidade de Transferência, é encontrado em diversas políticas de balanceamento de carga (DUBROVSKY, FRIEDMAN e SCHUSTER, 1996; MELLO, TREVELIN e PAIVA, 2003a; SHIVARATRI, KRUEGER e SINGHAL, 1992; SIEGELL e STEENKISTE, 1997) e também é utilizado pelo TLBA (MELLO, 2003). Esse *Threshold* viabiliza a transferência de carga de um computador para outro e evita a sobrecarga de um computador enquanto outros estão ociosos.

De acordo com Mello (2003), a capacidade ociosa de um computador é definida como:

$$\overline{I_{comp}} = 100 - I_{comp} \tag{6}$$

Quando um computador apresenta sucessores, a sua nova definição é dada por:

$$\overline{I_{comp-\alpha}} = (\tau 100) - I_{comp-\alpha} \tag{7}$$

onde:

a) $\overline{I_{comp-\alpha}}$ representa o complemento dos índices de carga de todos os computadores presentes na sub-árvore do computador α, ou ramo, a partir dele mesmo;

b) $I_{comp-\alpha}$ representa o somatório dos índices de carga de todos os computadores presentes na sub-árvore do computador α;

c) $\tau$ representa o número total de computadores presentes na sub-árvore de α e é calculado através do somatório de computadores que sucedem α mais uma unidade, que representa o próprio computador α.

Assim, quando um computador está sobrecarregado, é ativada a política de transferência e enviada uma mensagem ao computador raiz, requisitando o complemento médio do índice de carga de toda a árvore ($!I_{comp\text{-}médio\text{-}\alpha}$), que é representado na através da eq.(7).

Em seguida, aplicando-se a eq.(8), obtém o limite superior do complemento médio do índice de carga do computador α ($L_{\alpha}$).

$$L_{\alpha} = \overline{I_{comp-médio-\alpha}} - \left(\overline{I_{comp-médio-\alpha}}\,\delta t\right) \tag{8}$$

onde:

a) $\delta t$, conhecido como *Threshold* de Transferência, é o parâmetro percentual, configurável, utilizado para analisar a viabilidade de transferência de carga de um computador sobrecarregado para outro ocioso.

Caso a capacidade média livre dos computadores do ambiente seja maior do que a capacidade livre do computador sobrecarregado (eq.(9)), a transferência é viável para o processo e para o desempenho global do sistema.

$$\overline{I_{comp-sobrecarregado}} < L_{\alpha} \tag{9}$$

As vantagens dos algoritmos que utilizam esse parâmetro são: redução do número de mensagens geradas no sistema e transferências desnecessárias de processos. Uma desvantagem é que todo o processo está alocado localmente e existe um pequeno *overhead* para a transferência de seu contexto para um computador receptor, ou seja, com menor carga e apto para receber o processo do computador sobrecarregado (MELLO, 2003).

### 4.3.4 Número de Níveis da Árvore Computacional

Com relação ao número de níveis da árvore computacional, as simulações realizadas por Mello (2003) levaram em consideração a formação da estrutura lógica da árvore e não a quantidade de computadores por nível. Logo, será definida a formação mais apropriada da árvore para o algoritmo TLBA. Estudos sobre a influência do número de computadores nas árvores são apresentados na Seção 4.3.5.

Loh et al (1996) mostra que topologias com grandes distâncias médias entre os processadores e pouca conectividade entre os nós introduzem significativas sobrecargas de comunicação no sistema durante o processo de balanceamento de carga. Isto é devido à falta de *links* diretos entre os processadores desses computadores. Logo, a transferência de tarefas necessita, na maioria das vezes, passar por diversos processadores intermediários até alcançar o computador receptor. Além disto, haverá mais recursos locais se um processador tiver mais *links* direto entre os computadores. A situação agrava-se quando existe um aumento da estabilidade do sistema.

O trabalho de Loh et al (1996), mostra que a variação de parâmetros físicos nas interconexões de rede afetam significativamente o desempenho de estratégias de balanceamento de carga dinâmicos, sem considerar a sobrecarga dos níveis da rede.

Demais casos de estudos aplicados a esse item englobam os já aplicados no trabalho de Mello (2003), em termos de comparação e confirmação de seus resultados, além de outros para verificação da melhor estrutura de árvore para o TLBA.

No presente trabalho a simulação foi realizada em um ambiente composto por um número variado de computadores envolvendo capacidades de processamento homogêneas e heterogêneas. Foi submetida uma carga

mínima e uma carga máxima de processos ao ambiente, segundo o modelo de Feitelson com distribuição de Hiperexponencial com média de 1500 segundos.

A partir do conjunto de configurações aplicado ao TLBA, verificou-se o número de mensagens geradas por nível e os tempos médios de resposta (ver Capítulo 5).

### 4.3.5 Número de Computadores (nós)

Neste caso de estudo, verifica-se o comportamento do TLBA para um número variado de computadores, que são os computadores sucessores do computador α (raiz), localizado em um nível $N_k$.

Nos estudos realizados por Mello (2003), a simulação mostrou-se melhor para um sistema contendo 7 computadores, ou seja, obteve um melhor desempenho para essa configuração.

Neste trabalho, o número de computadores, variou em função do parâmetro estudado (ver Capítulo 5).

### 4.3.6 Ambientes Homogêneos e Heterogêneos

Para a análise da aplicação do TLBA em ambientes homogêneos e heterogêneos (ANÁLISE, 2004) foram simuladas diferentes configurações, mantendo-se os demais parâmetros inalterados.

No trabalho de Mello (2003), esta análise foi realizada para dois casos, com capacidade de processamento de 139 e 368 ciclos/ms que corresponde, respectivamente, a computadores AMD K6 430 MHz e AMD Duron de 950 MHz com 256 Mb de memória RAM (MELLO, 2003).

As análises com relação à heterogeneidade do sistema para o TLBA são apresentadas no Capítulo 5 deste trabalho.

## 4.4 Métricas de Análise e Políticas de Balanceamento do TLBA

As métricas de análises empregadas neste trabalho são as mesmas utilizadas por Mello (2003).

Essas métricas baseiam-se no tempo médio de resposta e no número de mensagens geradas no sistema nas operações de balanceamento (JAIN, 1991; MELLO, 2003; MELLO, TREVELIN e PAIVA, 2003a; SHIVARATRI, KRUEGER e SINGHAL, 1992).

Com relação às políticas de balanceamento de cargas do TLBA, foram elaborados fluxogramas (Figuras 4.1 a 4.5) que conduziram à proposta do novo simulador. As variáveis apresentadas nesses fluxogramas fazem parte dos cálculos aplicados as políticas de balanceamento do TLBA e são encontradas no decorrer das seções anteriores deste Capítulo 4.

O simulador possibilitou uma análise mais detalhada dos parâmetros de balanceamento de carga do TLBA e forneceu recursos essenciais para sua utilização seja em caráter didático ou científico.

Os principais parâmetros envolvidos nas políticas de balanceamento de carga do TLBA listados a seguir são apresentados no fluxograma pelos números que os antecedem.

1) Alteração dos Percentuais de Ocupação de Memória e CPU;

2) Alteração do *Threshold* de Atualização;

3) Alteração do *Threshold* de Transferência;

4) Alteração do número de computadores e/ou níveis da árvore;

5) Alteração do Índice de Carga e Heterogeneidade;

Na Figura 4.1 é apresentado o fluxograma da política de informação do TLBA.

Nesta política é definido um valor também conhecido como *poll time* que indica de quanto em quanto tempo serão coletadas informações do sistema. Assim, quando este tempo é alcançado, tem-se início a execução da política propriamente dita.

De posse das informações do sistema são calculadas as ocupações percentuais de CPU e memória e, por fim, o índice de carga de cada computador.

Pelo fluxograma pode-se perceber que caso não haja mudança significativa dos índices de carga, nenhuma ação é tomada e a política será executada novamente em um outro momento.

Caso ocorra uma mudança significativa no sistema, o novo índice de carga propagará do computador em questão para os seus antecessores até chegar ao computador raiz.

No caso de falhas um pedido é enviado ao Broker para a reorganização da árvore computacional.

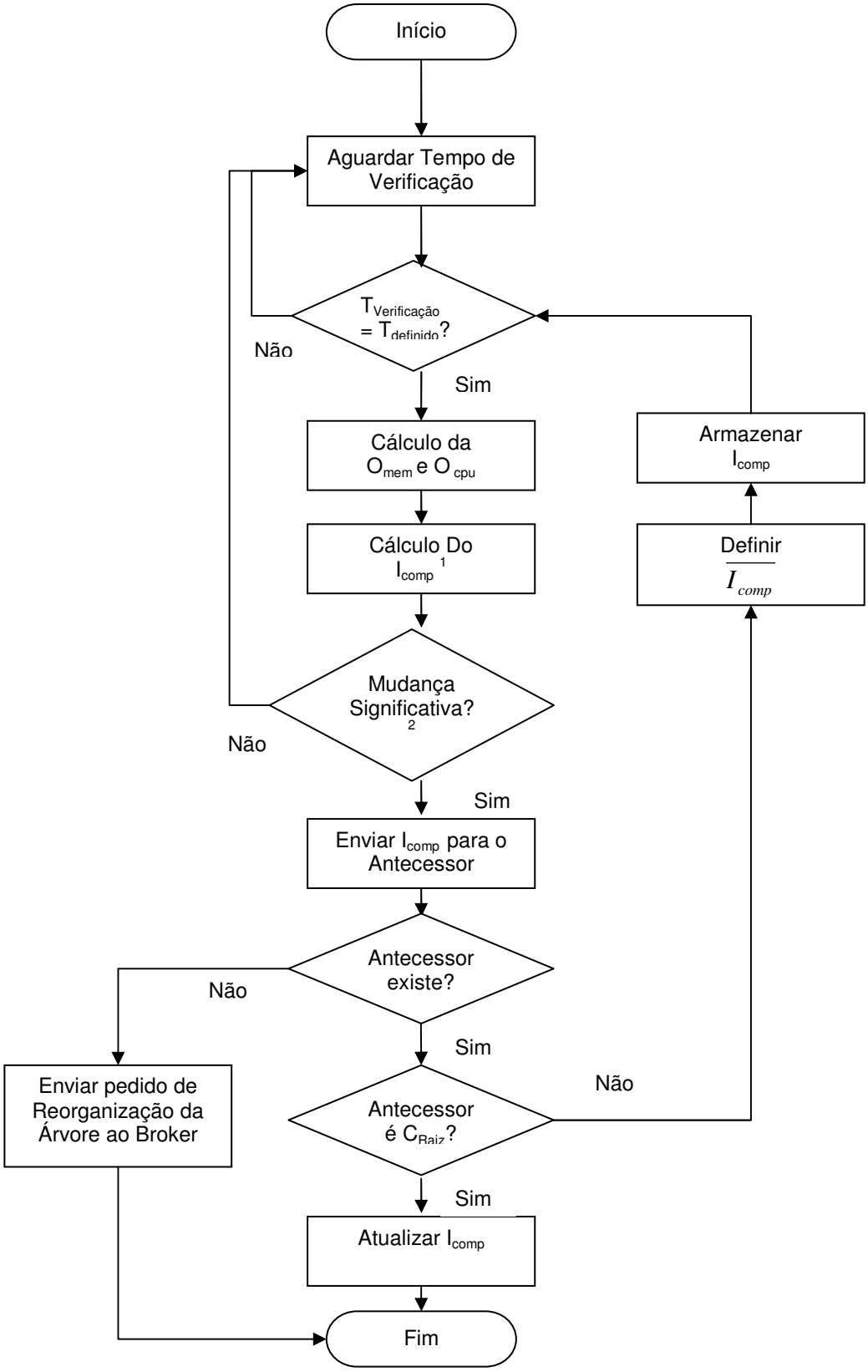


**Figura 4.1** – Fluxograma da Política de Informação do TLBA

Na Figura 4.2 é apresentado o fluxograma da Política de Localização do TLBA. Nesta política quando um computador recebe um pedido de execução de um processo através de um *broker* ou de um computador sobrecarregado são analisados os complementos dos índices de carga ($\overline{I_{comp}}$) dos computadores (eq.(7)).

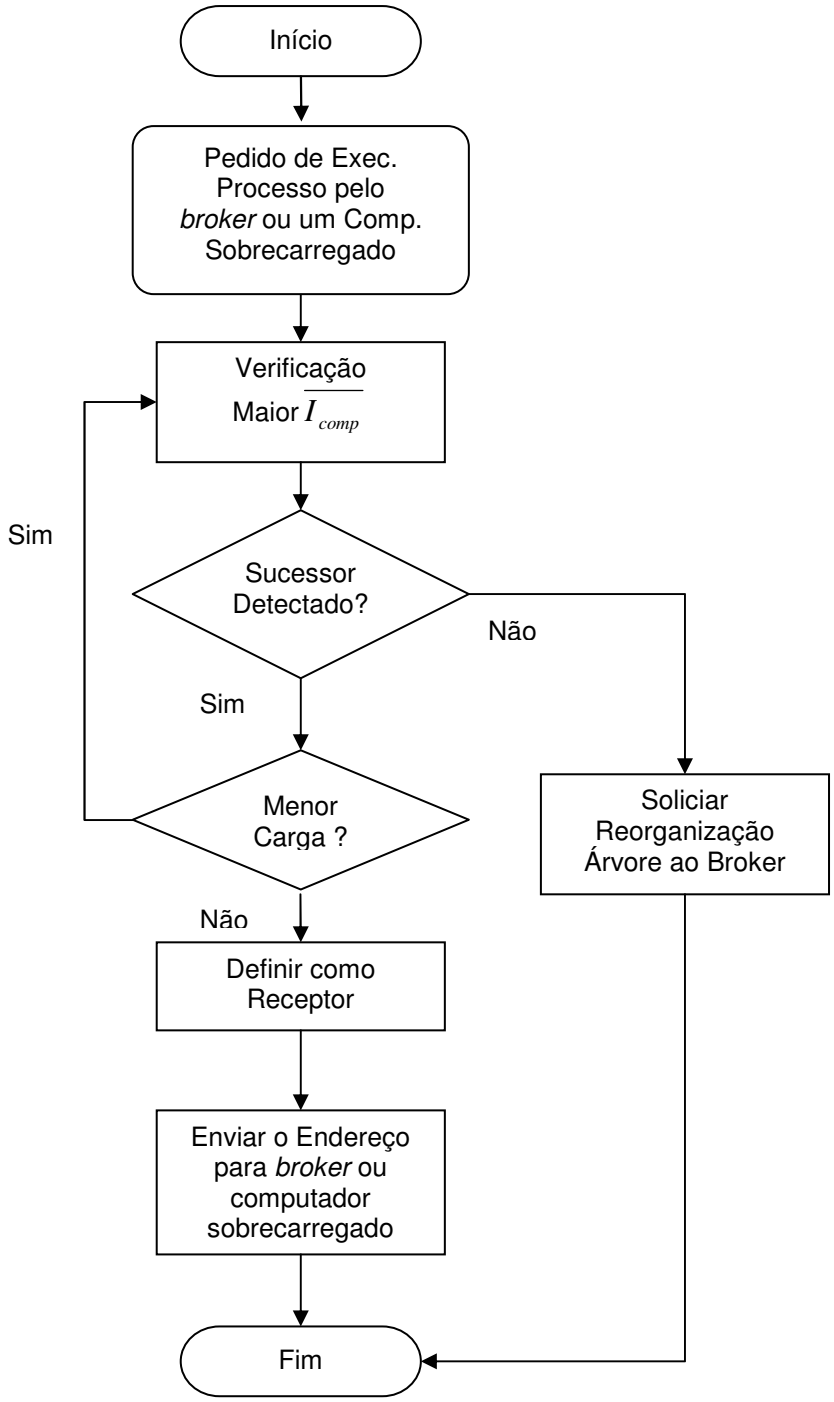


**Figura 4.2** – Fluxograma da Política de Localização do TLBA

Toda a operação de busca é iniciada no computador raiz, localizado no primeiro nível da árvore e propagada até encontrar o computador mais ocioso. Em seguida, seu endereço é enviado ao responsável pela ativação desta política podendo ser o *broker* ou um computador sobrecarregado.

Na Figura 4.3 é apresentada a política de transferência. É nessa política que é realizada a transferência de processos propriamente dita em função de análises realizadas por ela e as outras políticas de balanceamento de carga.

De acordo com Mello (2003), a política de transferência pode ser ativada em duas circunstâncias: a primeira é quando uma requisição, para executar um processo, chega ao *Broker* e a segunda quando um computador sobrecarregado busca dividir sua carga com outro elemento da árvore.

Na primeira circunstância de transferência, o *Broker* recebe uma requisição de um cliente que deseja executar processos no ambiente. Neste caso, o *Broker* ativa a política de localização do TLBA para encontrar o computador mais ocioso do sistema, que é definido como o receptor de processos. O *Broker* recebe o endereço desse receptor e migra o processo para ele, o qual é responsável por iniciar a execução.

Na segunda circunstância de transferência, um computador sobrecarregado ativa a política de transferência e envia ao computador raiz uma mensagem requisitando o complemento médio do índice de carga de toda a árvore (ver eq.(7)). De posse desse índice o computador sobrecarregado obtém o limite superior do complemento médio do índice de carga do computador raiz ($L_{raiz}$) (ver eq.(8)).

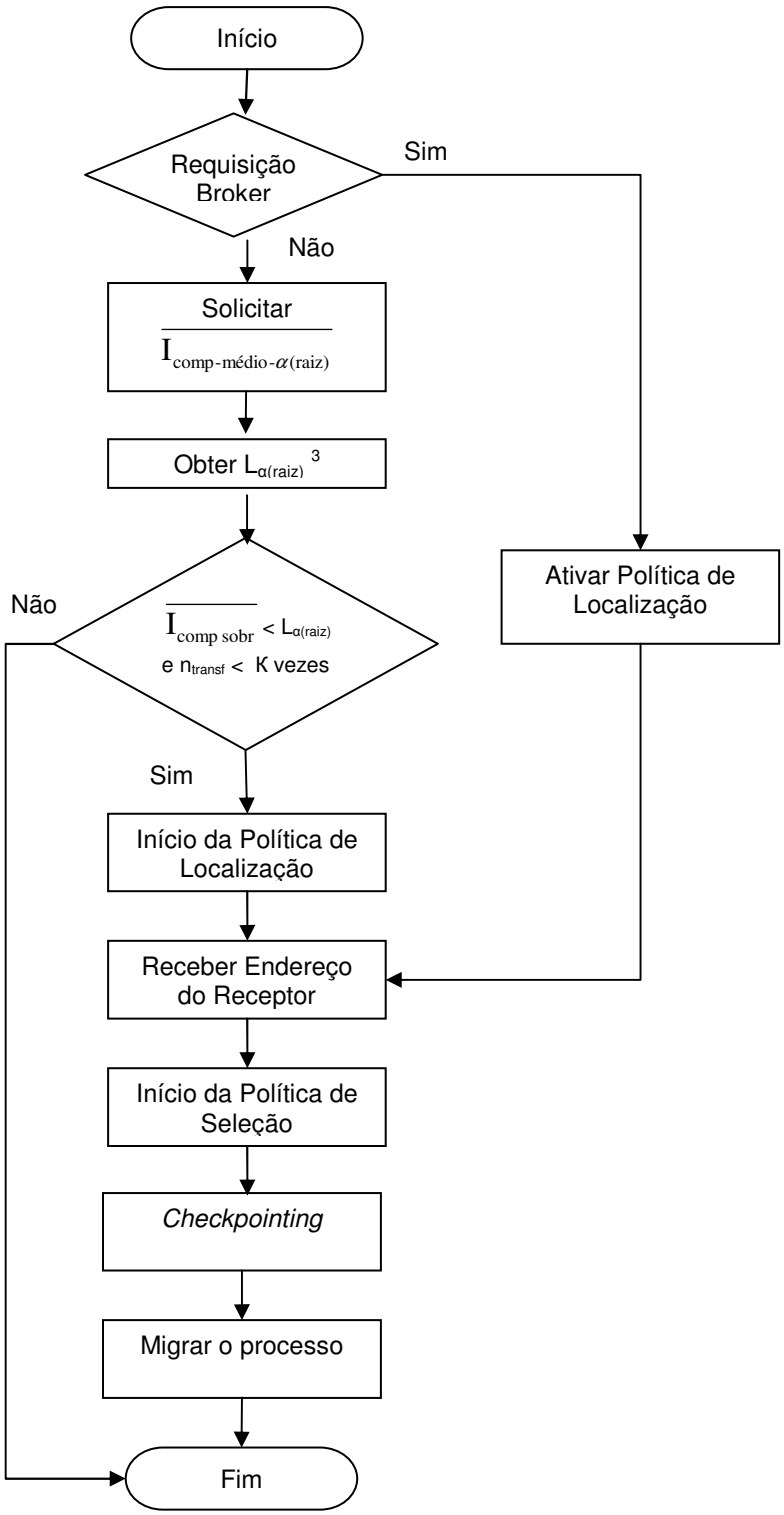


**Figura 4.3** – Fluxograma da Política de Transferência do TLBA

O computador sobrecarregado utiliza $L_{raiz}$ para avaliar se a capacidade média ociosa da árvore torna viável uma transferência de processo. Se a condição for satisfeita a transferência torna-se benéfica ao processo e ao desempenho global do ambiente. Caso contrário a política de transferência é cancelada.

Em caso de continuidade, essa política ativa a política de localização para encontrar um receptor de processos. O endereço desse receptor de processo é enviado para o computador sobrecarregada, também conhecido como emissor ou servidor de processos, que inicia a política de seleção (Figura 4.4).

No algoritmo TLBA essa política busca pelo processo que ocupa maior quantidade dos recursos de CPU e memória em um computador sobrecarregado (MELLO, 2003).

Uma vez localizado o processo de maior ocupação, seu identificador é submetido para a política de transferência que cuida das etapas de salvar seu contexto, transferi-lo para seu destino e reiniciá-lo.

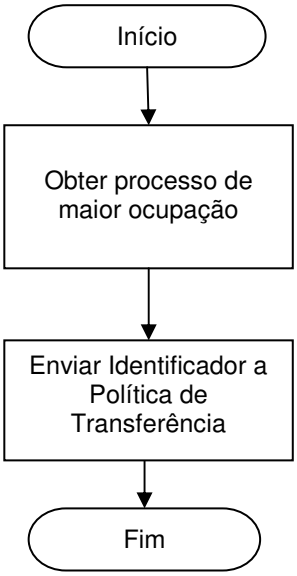


**Figura 4.4** – Fluxograma da Política de Seleção do TLBA

A partir dos conceitos das políticas de balanceamento do TLBA foi elaborado um estudo para o seqüenciamento e integração dos componentes que fariam parte do simulador.

Na Figura 4.5 é apresentado o fluxograma que define o funcionamento do simulador de uma maneira geral e a seqüência de obtenção de dados para aplicação das políticas de balanceamento do TLBA. Neste caso, tem-se basicamente a inicialização do sistema e a integração das políticas de balanceamento de carga de forma seqüencial.

Uma descrição mais detalhada do funcionamento do novo simulador pode ser encontrada na Seção 4.5.

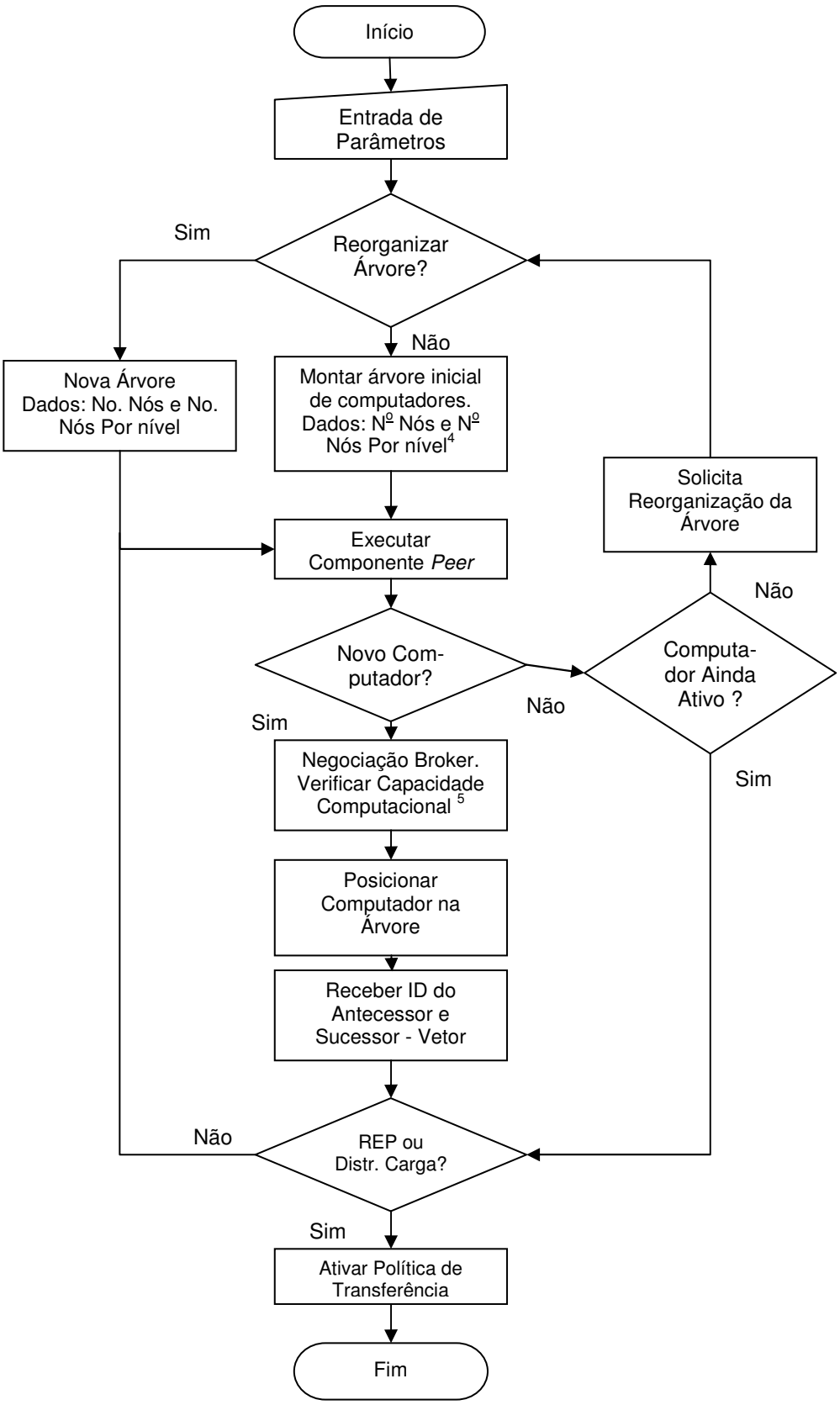


**Figura 4.5** – Fluxograma do Simulador

## 4.5 Descrição do Novo Simulador

O desenvolvimento inicial desse simulador teve como base o emulador e simulador[3] apresentados por Mello (2003), implementado em Linguagem de Programação da *Sun Microsystems* (2004), utilizada no desenvolvimento de sistemas orientados a objeto (*JAVA Technology*, 2004) e respeitado os termos especificados na *GNU General Public License* (2004).

O novo simulador foi implementado em linguagem de programação C#, devido aos vários recursos disponíveis e de forma a suportar as análises propostas deste trabalho, atender às políticas de balanceamento de carga do TLBA (Seção 4.4) e incorporar novos recursos e características para a sua melhoria.

No decorrer do seu desenvolvimento novos conceitos foram introduzidos e aprimorados fornecendo uma melhor interface com o usuário, praticidade na obtenção e busca de resultados e utilização de dados estatísticos que tornaram as simulações mais condizentes com um sistema real.

### 4.5.1 Objetivos da Simulação

Como já mencionado no Capítulo 3, o objetivo da simulação é o de avaliar sistemas e seus comportamentos com um menor custo e maior flexibilidade de variação dos parâmetros para encontrar as condições que conduzam a um melhor desempenho. Isto se aplica ao TLBA e através da variação dos seus parâmetros e posterior análise, será buscada a identificação de sistemas nos quais ele apresenta um melhor desempenho.

[3] Informações adicionais sobre simuladores e novos projetos relacionados ao TLBA, visite o site: http://www.icmc.usp.br/~mello/proj.html.

O novo simulador, além de facilitar a análise dos diversos parâmetros de balanceamento de carga, proporciona uma maior gama de estudos em um menor tempo de execução.

### 4.5.2 Diferenças entre os Simuladores

A seguir são apresentadas as principais diferenças entre o simulador proposto neste trabalho e o implementado por Mello (2003).

1) **Simulação em ambientes mais diversificados**, dando um enfoque maior às características do ambiente, desde ambientes totalmente homogêneos até ambientes totalmente heterogêneos e, também, levando-se em conta o tamanho do sistema. O simulador implementado por Mello (2003) permite a análise de apenas três casos de estudos (com relação aos níveis da árvore computacional do TLBA) e pouca diversificação do ambiente de *hardware*.

2) **Interface Gráfica** mais amigável com o usuário, facilitando a alteração dos parâmetros de entrada e a visualização dos resultados na própria janela onde se encontram os parâmetros, além de externar em arquivos resultados que demandam um maior processamento. O simulador implementado por Mello (2003) anterior executava no modo *Console* e para conhecer os parâmetros de entrada era necessário um conhecimento prévio do funcionamento do simulador.

3) **Abordagem de um maior número de parâmetros**. O implementado por Mello (2003) leva em consideração os seguintes parâmetros: capacidade de processamento por computador, ocupação da CPU por processos, números de processos no sistema e geração aleatória do tempo de chegada de cada processo através da distribuição de Poisson. Além disto, utiliza como índice de carga o número de processos na fila de espera do escalonador e não leva em consideração o percentual de ocupação de memória pelos processos.

Já o simulador aqui proposto leva em consideração não só os parâmetros citados anteriormente, mas também os apresentados a seguir.

a) Cálculo do índice de carga, baseado em percentuais da ocupação da CPU e da Memória, para os quais diferentes taxas de ocupação são atribuídas a cada um desses elementos (CPU e memória), verificando-se, assim, qual deles causaria uma maior influência no desempenho do sistema;

b) Intervalos de confiança e desvios padrão;

c) *Thresholds* de viabilidade de transferência de processos e de atualização dos índices de cargas;

d) Número de mensagens condizentes com as políticas de balanceamento de carga do algoritmo;

e) Cálculo das capacidades ociosas e seus complementos tanto em sistema com ou sem sucessores;

f) Cálculo das variações superiores e inferiores que permitem verificar mudanças significativas nos índices de carga dos computadores; e

g) Cálculo da média das capacidades dos computadores sucessores ao computador raiz, possibilitando a verificação da viabilidade de migração de processos.

4) **As Bases estatísticas fundamentadas em estudos prévios** e os dados estocásticos foram obtidos da monitoração e coleta de informações de sistemas reais. Nesse último caso, os tempos de execução e chegada de processos são baseados no trabalho de Feitelson (1996) e o número de iterações é dependente do intervalo de confiança e erro percentual sobre a média estabelecidos. O simulador anterior baseia-se na geração de dados randômicos e aleatórios.

5) **Sistema de escalonamento** no simulador proposto tende a uma maior similaridade com os sistemas reais, exceto pelas *threads*, porém com bases conceituais muito similares, em que um processo, ao chegar no sistema, entra em execução caso não haja tarefas pendentes e seguindo a política de escalonamento de cada sistema. No novo simulador foram implementadas quatro políticas de escalonamento: *Round-Robin, Shortest Job First, First-come First-served* e por prioridades (GUIMARÃES, 1980; OLIVEIRA, CARISSIMI e TOSCANI, 2001; SHAY, 1996; TANENBAUM, 1992). Se o tempo de execução do processo for atendido, este é removido do sistema, senão é decrementado. Além disso, caso já existam outros processo na fila de espera do escalonador e dependendo do escalonamento utilizado, define-se um *quantum* para cada processo.

### 4.5.3 Componentes do Simulador

O código fonte do simulador apresenta alguns componentes interessantes para o ambiente de simulação e que fazem com que o simulador forneça resultados mais próximo dos sistemas reais. Esses componentes internos do *software* são apresentados e discutidos a seguir.

***Classes Relacionadas ao TLBA***

**Broker**: traz consigo métodos que envolvem a montagem da árvore lógica, busca pelo melhor nó no sistema, envio dos processos para execução etc. Seus métodos são um dos primeiros a serem chamados na rotina de execução, seguindo as políticas de balanceamento de carga do TLBA.

**Peer**: é a classe responsável pela aplicação das políticas do TLBA e contém métodos para migração e adição de novos nós nos escalonadores de

cada computador, cálculos da ocupação da CPU, Memória e Índice de Carga. O escalonador nada mais é do que uma instanciação desse componente para classe *Computer*.

**Classe Computer**: como o próprio nome diz, está relacionada aos nós/computadores do sistema. Ela contém métodos para fila de processos, cálculos das capacidades computacionais, variações superiores e inferiores e limite superior do complemento médio do índice de carga do computador raiz ($L_{raiz}$).

**Classe MidProcess**: contém o seu construtor e destrutor que carregam os parâmetros necessários de cada processo no sistema e métodos que retornam o tempo final de resposta e o tempo decorrido de cada processo.

***Características das Simulações***

**Metodologia de Feitelson**: no simulador foi utilizado um arquivo no formato texto, que contém o modelo proposto por de Feitelson (1996) e baseado em observações de 6 diferentes *logs* produzidos por máquinas paralelas (128-nós iPSC/860 na NASA Ames, 128-nós IBM SP1 em Argonne, 400-nós Paragon na SDSC, 126-nós Butterfly na LLNL, 512-nós IBM SP2 na CTC e 96-node Paragon na ETH, Zurique). O objetivo de seu uso é o de se ter um modelo de carregamento e que possa ser utilizado em simulações com escalonamento de processos. Esse modelo foi de extrema importância para a criação de um sistema mais real no âmbito de simulação.

**Tipos de Escalonamentos**: o novo simulador foi implementado utilizando diferentes tipos de escalonamento para obtenção do tempo final de resposta de um processo em execução, seguindo as regras de escalonamentos reais.

### *Projetos Auxiliares Utilizados no Simulador*

O novo simulador contempla recursos mais diversificados que o simulador apresentado por Mello (2003), sendo de caráter didático e podendo ser utilizado para apresentações de teorias em sala de aula no intuito de introduzir conceitos básicos relacionados aos ambientes distribuídos e algoritmos de balanceamento de cargas.

Dessa forma, foi necessário a utilização de outros projetos que foram importantes para a elaboração final do simulador e de suas características. A maioria dos projetos são encontrados em Code Project (2005), Codeguru (2005) e Sourceforge.NET (2005).

Dentre os principais projetos utilizados em C#, para a implementação da interface gráfica do novo simulador, podem-se destacar:

**SourceGrid**: um Projeto *Open Source* desenvolvido por David Icardi. Esse projeto é responsável pela elaboração dos *grids* onde os dados finais das simulações são apresentados em série no formato de uma tabela (ICARDI, 2005). É um projeto simples, porém com diversos recursos e modelo de elaboração e edição de tabelas.

**Lithium Tree Control**: também conhecido como *Netron Project* (2005). É responsável pela diagramação da árvore de computadores de acordo com suas capacidades computacionais. Foi a princípio desenvolvido para edição e visualização de dados em XML ou qualquer árvore de dados estruturados.

**Statistics:** Projeto desenvolvido por Brent (2002) e provê as distribuições estatísticas de Poisson, Binomial, Exponencial, Gaussiana, Gamma e Uniforme. Os resultados fornecidos por essas classes são utilizados como tempo de chegada do processo ao sistema.

**TSWizard:** é um *framework* para .NET utilizado para a implementação do *Wizard* para a montagem final dos gráficos e foi desenvolvido por Johnson (2005).

**ZedGraph:** esse projeto foi utilizado para a elaboração dos gráficos apresentados no simulador em função dos resultados obtidos das simulações e apresentados no formato de uma tabela (ZEDGRAPH, 2005).

**WinFormsUI:** Essa biblioteca é responsável pela elaboração de todo o ambiente de trabalho, onde o simulador é suas telas são apresentadas. Possui recursos adicionais e maneiras de exposição de propriedades, *toolbars* etc. O projeto desenvolvido por Luo (2004) tem sido utilizado em projetos em .NET.

O simulador e suas interfaces são apresentados na Seção Resultados e Discussões (Capítulo 5).

### 4.5.4 Lógica Implementada no Simulador

A lógica utilizada no simulador é descrita brevemente nesta Seção, destacando-se os principais pontos a serem considerados na execução do simulador.

Primeiramente, deve ser feita uma análise e ajuste dos parâmetros do TLBA para cada caso a ser estudado. Como esses valores configurados pode se dar início ao processo de simulação através do botão *Start* do Menu *Simulation*.

Já em execução, o primeiro passo a ser realizado pelo simulador é a montagem da árvore lógica através de solicitações ao componente Broker.

A montagem da árvore lógica inicia-se com a criação do computador raiz e atribuição de seu nível como sendo zero. Em seguida, os demais nós são inseridos no nível um até chegarem ao valor máximo de computador permitidos por nível e são calculadas as capacidades de cada computador. É estabelecido o nó pai e os nós filhos nessa montagem. No passo seguinte, é feita uma análise em função das capacidades computacionais, onde a capacidade do primeiro nó (filho) recebe um valor *default*. É realizada uma busca por

computadores com capacidade inferior a esse valor. Quando encontrada essa capacidade é atribuída a esse nó filho que então assume uma posição na árvore computacional (nesse caso, nível dois). Demais buscas são realizadas agora visando a alocação dos outros computadores na sub-árvore com menor capacidade computacional.

Os dados obtidos dos estudos de Feitelson (1996), Feitelson e Rudolph (1995) e Feitelson e Jette (1997) são utilizados como o tempo total em que o processo permanece no sistema, com um tempo de chegada baseado na distribuição Hiperexponencial com média de 1500 segundos.

A metodologia de criação dos processos no sistema segue os padrões convencionais de sistemas de escalonamento em tempo real. A cada processo são atribuídos um tempo de execução, ocupação de CPU e memória e um número máximo de migrações permitidas por processo.

Esses processos chegam ao sistema através do Broker (primeira caso de transferência) que será o responsável por encontrar o computador mais ocioso, sempre a partir do computador raiz, e adicioná-lo na fila de espera de cada escalonador.

Utilizando-se como exemplo o algoritmo de escalonamento *Round-Robin*, onde, à medida que um processo P1 chega ao computador mais ocioso do sistema sua execução é imediata. Ele permanece em execução até o momento em que outro processo P2 chega ao sistema, de acordo com a distribuição selecionada. O tempo, Td em que P1 esteve em execução é decrementado de seu tempo total de execução Tt e pode-se determinar o tempo restante Tr do processo pela eq.(11).

$$\mathbf{T_r = T_t - T_d} \qquad (11)$$

Se o tempo decorrido $\mathbf{T_d}$ for inferior ao tempo que o processo **P2** entrará em execução, então, ocorre o término do processo, remoção desse da fila do

escalonador e obtenção do tempo de resposta total de execução do processo **P1** no computador selecionado.

O simulador também obedece a outras políticas de escalonamento, como comentado anteriormente na Seção 4.5 (GUIMARÃES, 1980; OLIVEIRA, CARISSIMI e TOSCANI, 2001; SHAY, 1996; TANENBAUM, 1992).

No algoritmo de execução do processo no computador selecionado também são realizados os cálculos de capacidades ociosas e médias, índices de carga, variações superiores e inferiores e limite superior do complemento médio do índice de carga do computador raiz ($L_{raiz}$).

De posse desses valores, é verificado se a capacidade ociosa do nó, considerado sobrecarregado (segundo caso de transferência) é inferior à capacidade média ociosa de todos os nós presentes no sistema a partir do raiz, conforme a eq.(10). Satisfazendo-se essa condição, dá-se continuidade à política de transferência do processo.

Esse processo é removido da fila de execução, somente ao completar o seu tempo total de execução. É, também, respeitada a chegada de outros processos, ocorrendo o chaveamento de um para o outro de acordo com o algoritmo de escalonamento selecionado.

Quando seu tempo de execução é superado, obtêm-se o tempo de resposta final desse processo.

Caso não ocorram migrações, é verificado se o sistema apresentou mudanças significativas através da variação superior e inferior do índice de carga.

O ciclo de simulações é concluído após a execução de todos os processos, conforme parâmetros estabelecidos no início da simulação.

Neste trabalho, foram considerados também os números de mensagens de balanceamento de carga, atualização da ocupação dos recursos de CPU e

memória e das mensagens de solicitação de capacidade ociosa média total de todos nós presentes na árvore, a partir do computador raiz.

## 4.6 Considerações Gerais

Esse capítulo apresentou o levantamento dos parâmetros de balanceamento de carga envolvido no TLBA e as características que devem ser consideradas na elaboração de um simulador para tal algoritmo de balanceamento de cargas.

Esse Capítulo servirá de base para a implementação do novo simulador e dos casos de estudos a serem abordados durante o processo de simulação com o objetivo de uma melhor análise dos parâmetros de balanceamento de carga do TLBA (MELLO, 2003) e uma maior diversificação dos meios de aplicação.

Os resultados obtidos do novo simulador (Capítulo 5) permitiram a análise do TLBA em ambientes mais distintos e verificação dos impactos que a variação de seus parâmetros pode trazer ao desempenho computacional de um sistema.

Capítulo

5

# 5 RESULTADOS E DISCUSSÃO

Como resultados finais deste trabalho é apresentada a implementação interface do novo simulador (ver Seção 5.1) e o estudo dos parâmetros de balanceamento de carga do TLBA para diferentes tipos de sistemas e casos de estudo (ver Seção 5.2).

À medida que os resultados são apresentados, também se faz a discussão sobre as mudanças do comportamento do TLBA e dos principais pontos observados para cada simulação.

## 5.1 Novo Simulador e Seus Recursos

Ao simulador apresentado nesse trabalho foi dado o nome de TLBASim (*Tree Load Balancing Algorithm Simulator*) e, como já mencionado no Capítulo 4, ele foi desenvolvido em C#.

A linguagem de programação C# juntamente com as características do *framework* .NET possibilitaram a implementação de diversos recursos para o novo simulador, devido à praticidade e facilidade na programação, além de projetos auxiliares estarem disponíveis em Code Project, Codeguru e Sourceforge.NET (2005).

Na Figura 5.1 é apresentada a *Splash Window* do simulador que fornece subsídios para operações de inicialização de interfaces gráficas e demais variáveis que façam necessárias.

**Figura 5.1** – *Splash Window* do Simulador

A interface gráfica do TLBASim é apresentada na Figura 5.2. A princípio consiste de um janela principal e várias janelas filhas. O projeto WinformsUI (LUO, 2004) utilizado para elaboração dessa interfaces iniciais possibilita a movimentação das janelas filhas em diversas posições da janela principal. Isso fornece ao usuário uma maior flexibilidade para se trabalhar com o ambiente e posicionar as telas mais relevantes durante a fase de simulação.

Essas janelas filhas podem ser movimentadas uma em cima da outra, ou uma ao lado da outra, ou separadas da janela principal em um formulário à parte, ou minimizadas, fechadas, ocultadas etc.

Cada uma das janelas filhas apresentadas na janela principal serão descritas a seguir. Os demais recursos presentes no simulador, principalmente as características relativas a cada uma das janelas (copiar, colar, remover, editar, modificar, colorir etc) não serão comentadas aqui de forma a evitar o excesso de informações e manter o foco somente nas características relacionadas ao processo de simulação.

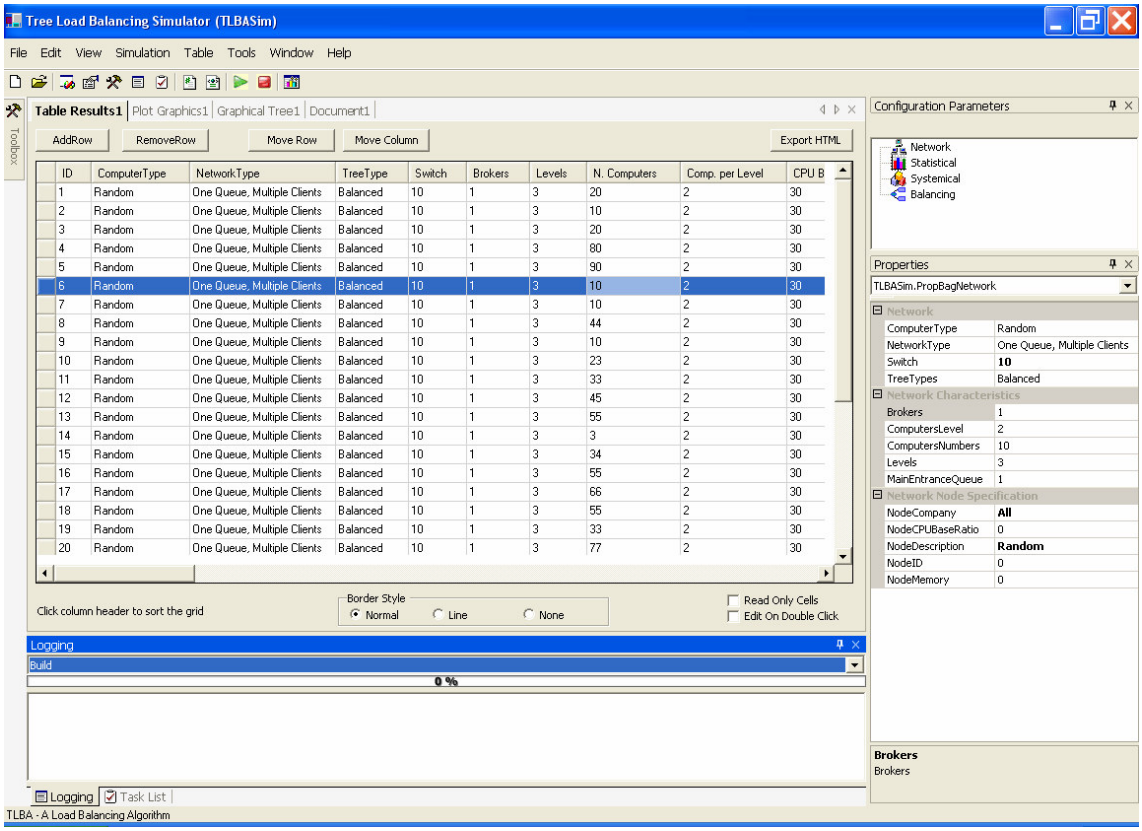


**Figura 5.2** – Interface Gráfica inicial de apresentação do TLBASim.

Além da janela principal e suas principais janelas filhas - janelas onde são apresentadas as tabelas com os resultados das simulações, gráficos, árvore com a diagramação dos computadores distribuídos durante a simulação e a janela para edição de texto - tem-se também as janelas com as propriedades do simulador.

Primeiramente, é importante ressaltar que os parâmetros de configuração relacionados aos algoritmos de balanceamento de carga, dados estatísticos e ao ambiente de simulação foram divididos em quatro classes denominadas de Redes, Estatística, Sistêmica e Balanceamento (Figura 5.3).

O acesso às propriedades dessas classes é dado através da seleção individual de cada uma delas e podem ser visualizadas em uma outra janela filha (Figura 5.4).

**Figura 5.3** – Quatro classes com as características do ambiente de simulação.

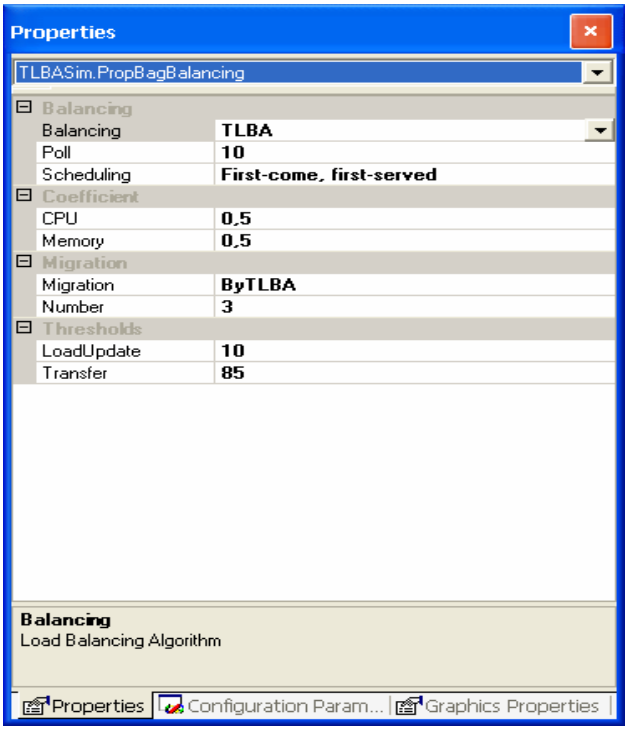


**Figura 5.4** – Propriedades Individuais das quatro classes

- **Classe Rede (*Network*):** apresenta as propriedades de uma rede distribuída e da árvore a ser montada pelo TLBA no processo de simulação. Dentre os parâmetros desta classe são destacados os tipos de computadores presentes na rede, suas capacidades computacionais e memória, *switch*, número de níveis da rede, número de computadores presentes nessa rede etc;

- **Classe Estatística (*Statistical*):** apresenta os parâmetros relacionados aos dados estatísticos em questão, como por exemplo, arquivos contendo tempos de execução, intervalo de confiança a ser utilizado etc.

- **Classe Sistêmica (*Systemical*):** apresenta parâmetros gerais do sistema, referentes ao tipo de aplicações envolvidas no ambiente, número de aplicações que chegaram em cada simulação etc.

- **Classe Balanceamento (*Balancing*):** aqui estão presentes os parâmetros relacionados aos algoritmos de balanceamento de carga, tais como algoritmo a ser utilizado, número máximo de migrações, *Thresholds* etc.

Uma outra janela filha que merece destaque é a de *Logging* (Figura 5.5). Essa janela traz os *logs* de alguns dos principais pontos do simulador para depuração do código e de parâmetros específicas durante cada simulação para encontrar possíveis falhas.

O Modo Debug pode ser habilitado ou desabilitado através do Menu *Window*.

Existem ainda outras janelas filhas com menor importância para o TLBASim, como a *Task List* e *Toolbox* que estão presentes na janela principal, mas não têm nenhuma funcionalidade. Elas poderiam ser utilizadas para a adição de procedimentos de simulação e recursos adicionais, respectivamente, que futuramente o simulador viesse a incorporar.

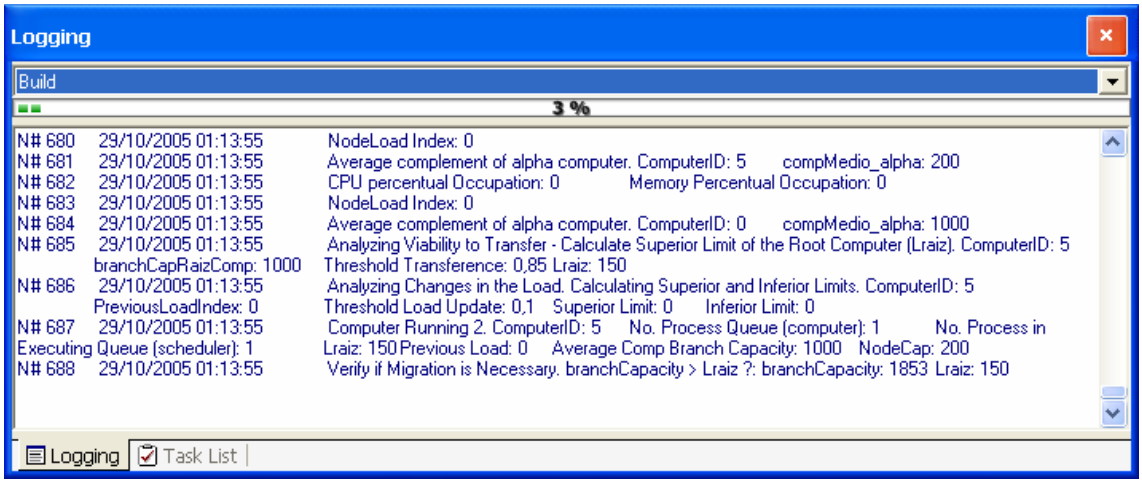


**Figura 5.5** – Janela de *Log* do TLBASim

Já com relação às janelas principais, o simulador foi implementado de forma a fornecer ao usuário recursos suficientes para a execução das simulações sem a necessidade de novas janelas ou atalhos.

Desta forma, os resultados finais de cada ciclo de simulação são apresentados em uma única tabela (Figura 5.6), com características adicionais que proporcionam um menor trabalho ao usuário para o manuseio e transporte de dados para outros programas de edição de texto ou gráficos.

Nessa tabela estão os parâmetros de entrada de cada ciclo de simulação e seus resultados finais.

Ela pode ser salva em XML, aberta novamente no simulador ou em outro programa que seja capaz de interpretar a formação presente nesse arquivo, como é o caso do Microsoft Excel®.

Table Results1

AddRow | RemoveRow | Move Row | Move Column | Export HTML

| ID | ComputerType | NetworkType | TreeType | Switch | Brokers | Levels | N. Computers | Comp. per Level | CPU Bound |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 20 | 2 | 30 |
| 2 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 10 | 2 | 30 |
| 3 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 20 | 2 | 30 |
| 4 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 80 | 2 | 30 |
| 5 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 90 | 2 | 30 |
| 6 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 10 | 2 | 30 |
| 7 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 10 | 2 | 30 |
| 8 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 44 | 2 | 30 |
| 9 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 10 | 2 | 30 |
| 10 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 23 | 2 | 30 |
| 11 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 33 | 2 | 30 |
| 12 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 45 | 2 | 30 |
| 13 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 55 | 2 | 30 |
| 14 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 3 | 2 | 30 |
| 15 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 34 | 2 | 30 |
| 16 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 55 | 2 | 30 |
| 17 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 66 | 2 | 30 |
| 18 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 55 | 2 | 30 |
| 19 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 33 | 2 | 30 |
| 20 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 77 | 2 | 30 |
| 21 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 66 | 2 | 30 |
| 22 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 66 | 2 | 30 |
| 23 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 43 | 2 | 30 |
| 24 | Random | One Queue, Multiple Clients | Balanced | 10 | 1 | 3 | 44 | 2 | 30 |

Click column header to sort the grid

Border Style: Normal | Line | None

Read Only Cells

Edit On Double Click

**Figura 5.6** – Tabela com os Parâmetros de Entrada e Resultados

A partir dos resultados obtidos e apresentados na Tabela da Figura 5.6 é possível à elaboração de três tipos de gráficos que são apresentados nas Figura 5.7 a 5.9.

Esses gráficos podem ser criados através dos valores apresentados nas colunas ou linhas da tabela. Suas restrições estão apenas com relação ao número de curvas desejadas por gráfico.

Os parâmetros de entrada são utilizados como coordenadas do eixo X e os resultados como ordenadas do eixo Y.

Terminado o gráfico, ele pode ser salvo no formato **png** que é universal para esse padrão de imagem é pode ser aberto também por diversos aplicativos.

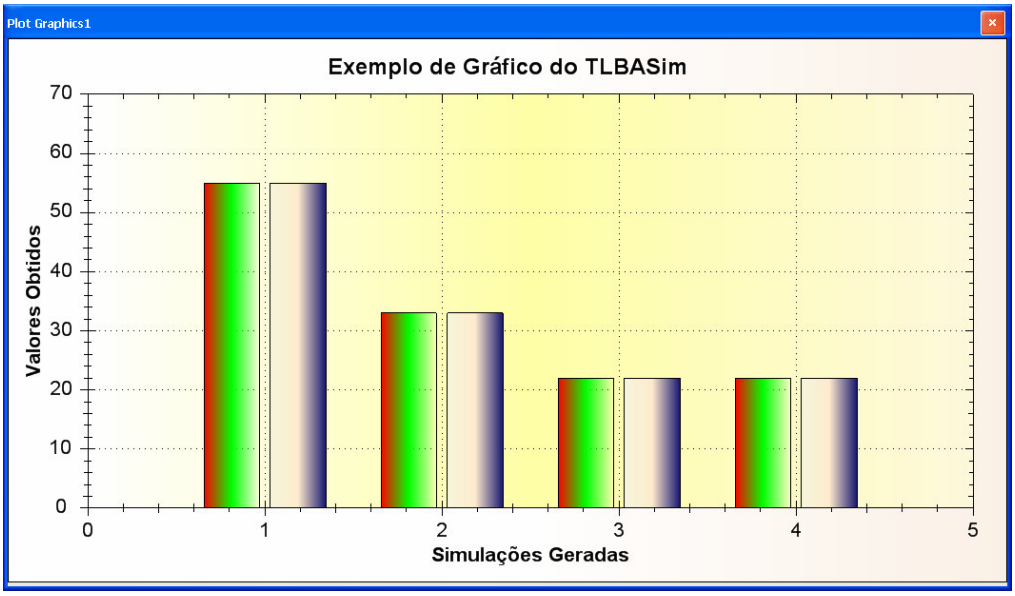


**Figura 5.7** – Gráfico exemplo do tipo colunas.

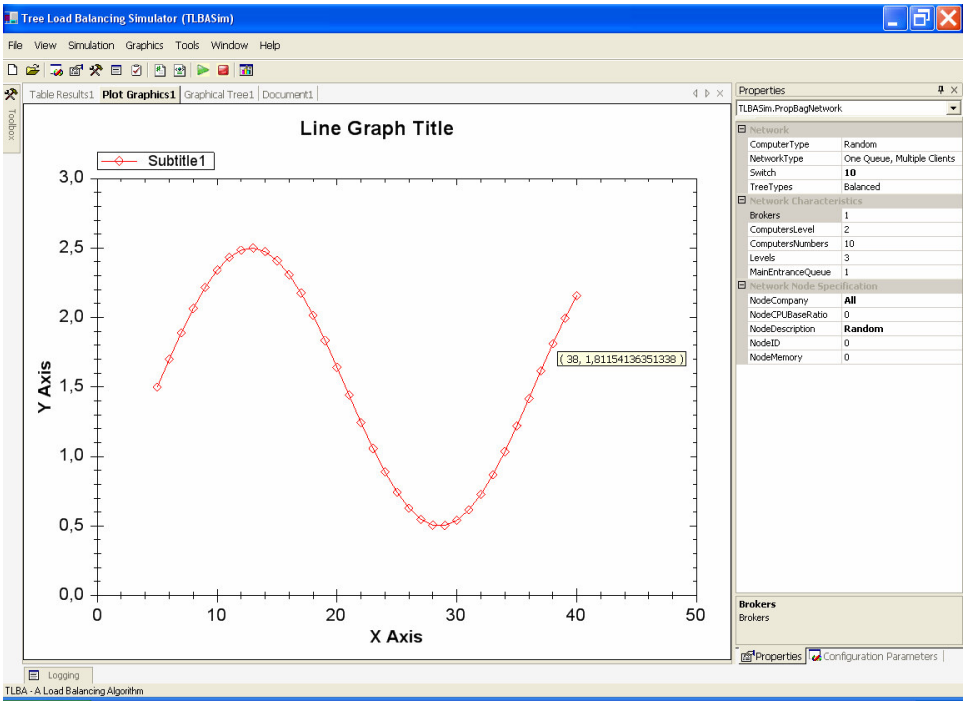


**Figura 5.8** – Gráfico exemplo do tipo linhas.

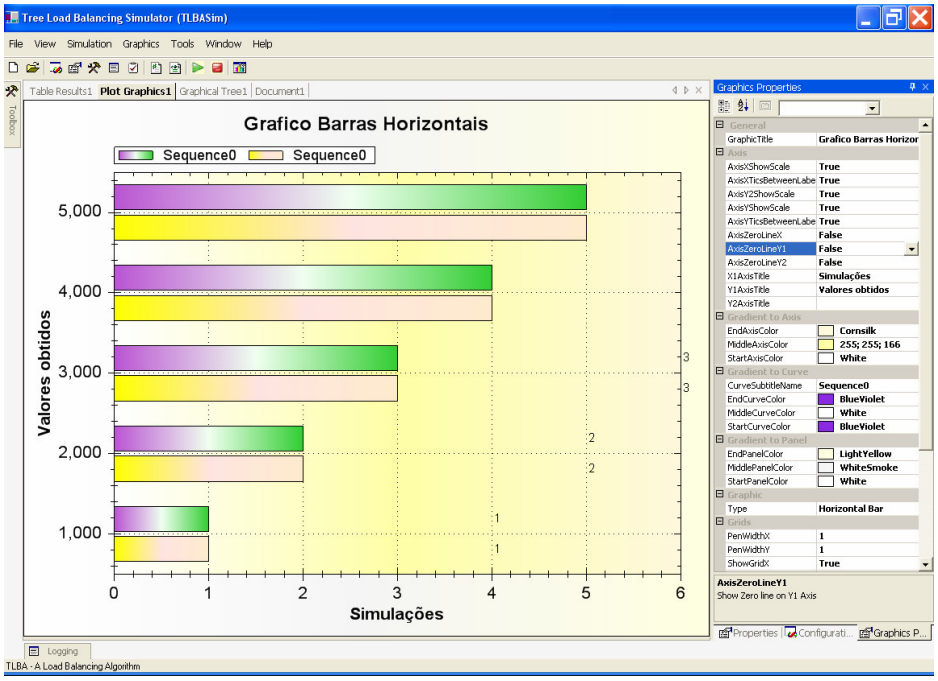


**Figura 5.9** – Gráfico exemplo do tipo barras horizontais

Com o gráfico já na tela principal é possível a utilização de recursos tais como: *zoom in* e *out*, ajuste de escalas, seleção de cores para as barras, fundos internos e externos, alteração de títulos, legendas, *grids* etc.Essas propriedades de edição do gráfico são apresentadas na Figura 5.10.

Para auxiliar o usuário no processo de criação dos gráficos foi desenvolvido um *Wizard* baseado nos formulários do Projeto TSWizard (JOHNSON, 2005), que permite a seleção dos dados a serem utilizados no gráficos, escolha de suas características, número de curvas desejados etc.

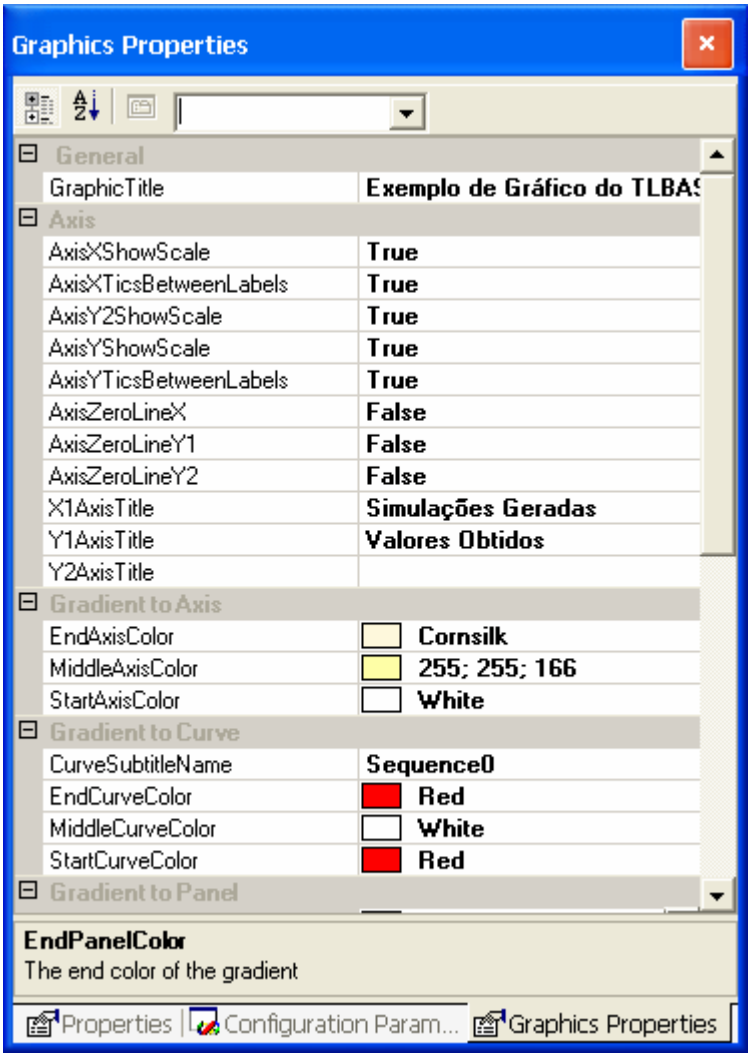


**Figura 5.10** – Propriedade Relacionadas aos Gráficos

A Figura 5.11 refere-se à apresentação do *Wizard* que pode ser iniciado através do menu *Graphics* ou do ícone , apresentado na barra de ferramentas. Entretanto, para se evitar quaisquer problemas na hora de visualização do gráfico é necessário que o usuário selecione a janela principal *Plot Graphics*, antes de iniciar a sua construção.

O passo seguinte à janela de apresentação é a seleção do tipo de gráfico que se deseja trabalhar (Figura 5.12).

**Figura 5.11** – Janela de Apresentação do *Wizard*.

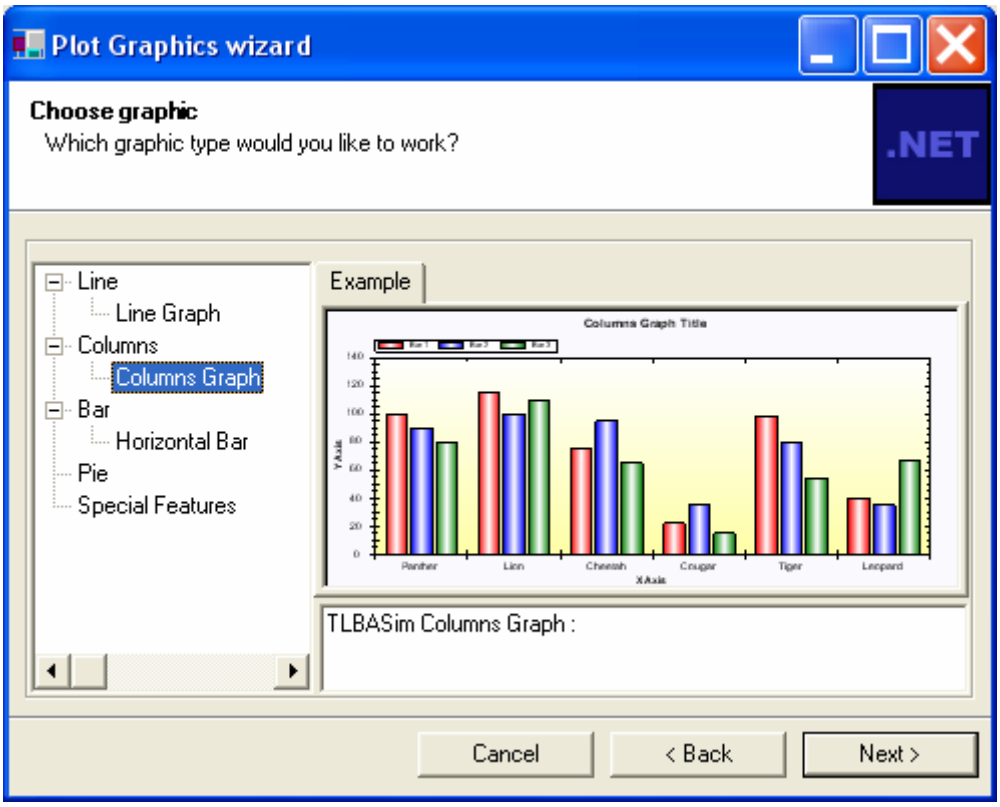


**Figura 5.12** – Seleção do Tipo de Gráfico

Como mencionado anteriormente o TLBASim suporta apenas gráficos do tipo de colunas, linhas e barras horizontais. Entretanto, novas implementações através da ZedGraph (2005) podem ser facilmente elaboradas uma vez que sua biblioteca disponibiliza exemplos e ferramentas auxiliares.

O próximo passo a ser considerado é a dos dados a serem utilizados pelo gráfico (Figura 5.13).

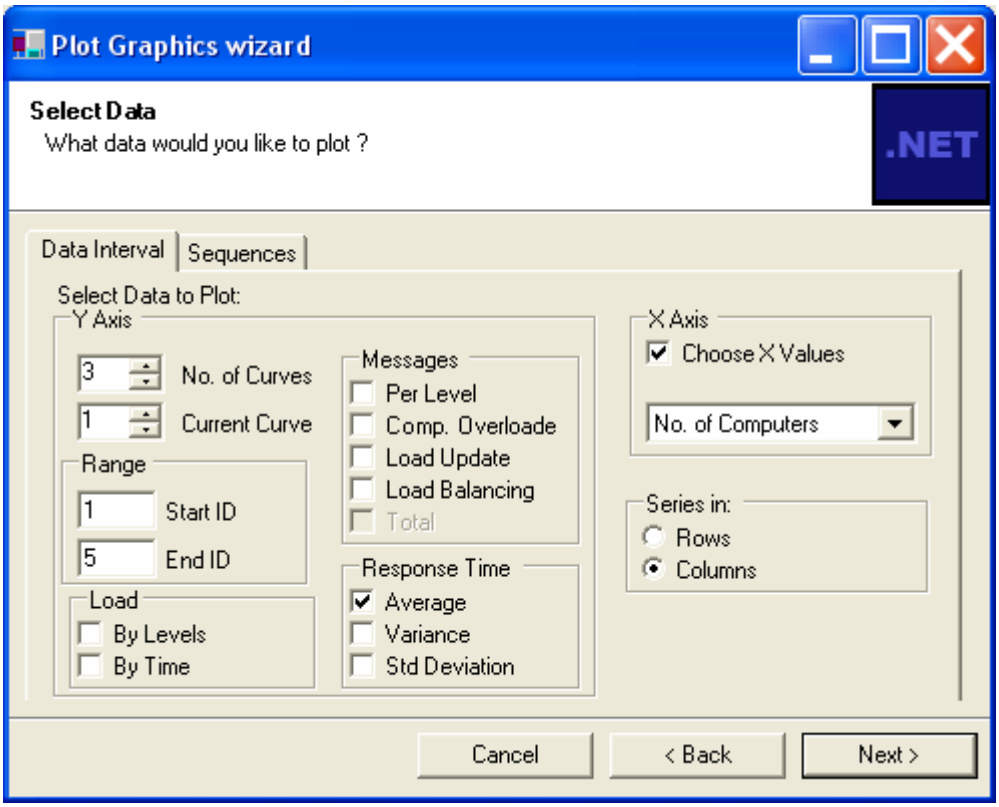


**Figura 5.13** – Passo 1: Seleção dos Dados

Para a seleção dos dados é necessário ressaltar alguns procedimentos básicos para que a elaboração do gráfico seja bem-sucedida. Primeiramente, deve-se escolher o número de curvas a serem consideradas, ou seja, os tipos de resultados que devem ser apresentados.

Esses resultados podem ser selecionados de uma mesma coluna sendo de um único tipo como, por exemplo, o tempo médio de resposta ou uma mistura de resultados quando da seleção de mais de uma coluna da tabela. A

variação estará em função dos parâmetros de entradas selecionados, por exemplo, diferentes números de processos simulados para diferentes *Thresholds*.

Os resultados finais da simulação são apresentados nos *CheckBoxes*, enquanto que os parâmetros de entrada estão relacionados no *ComboBox*.

Na tabela de resultados do simulador deve-se selecionar a coluna *Plot* para aqueles valores que se deseja montar o gráfico. Os resultados selecionados podem ser visualização em função da variação dos elementos do eixo da coordenadas. Para isso, basta clicar em *Choose X Values*.

O passo seguinte é a escolha das características do gráfico (Figura 5.14).

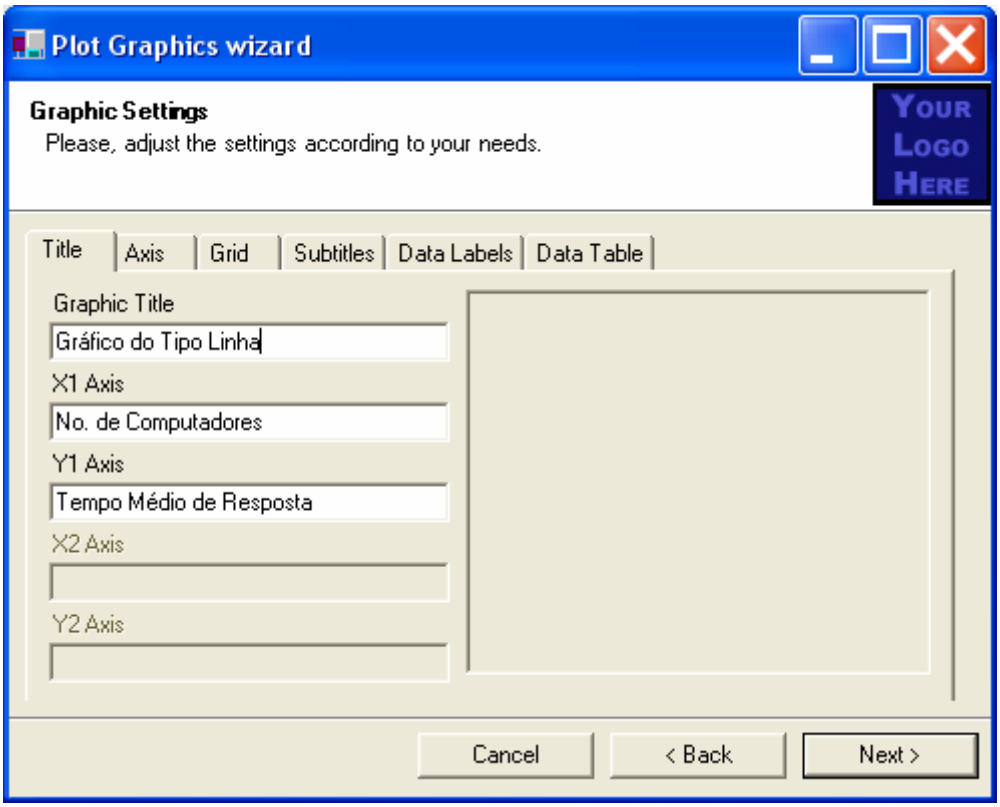


**Figura 5.14** – Passo 2: Seleção da Propriedades

Essas características são relacionadas aos títulos, *grids*, exibição e posicionamento de legendas etc.

Uma outra janela principal é a do diagrama de árvores (Figura 5.15). Nesta janela é montada a árvore lógica dos computadores em tempo real de simulação em função de suas capacidades computacionais relativas.

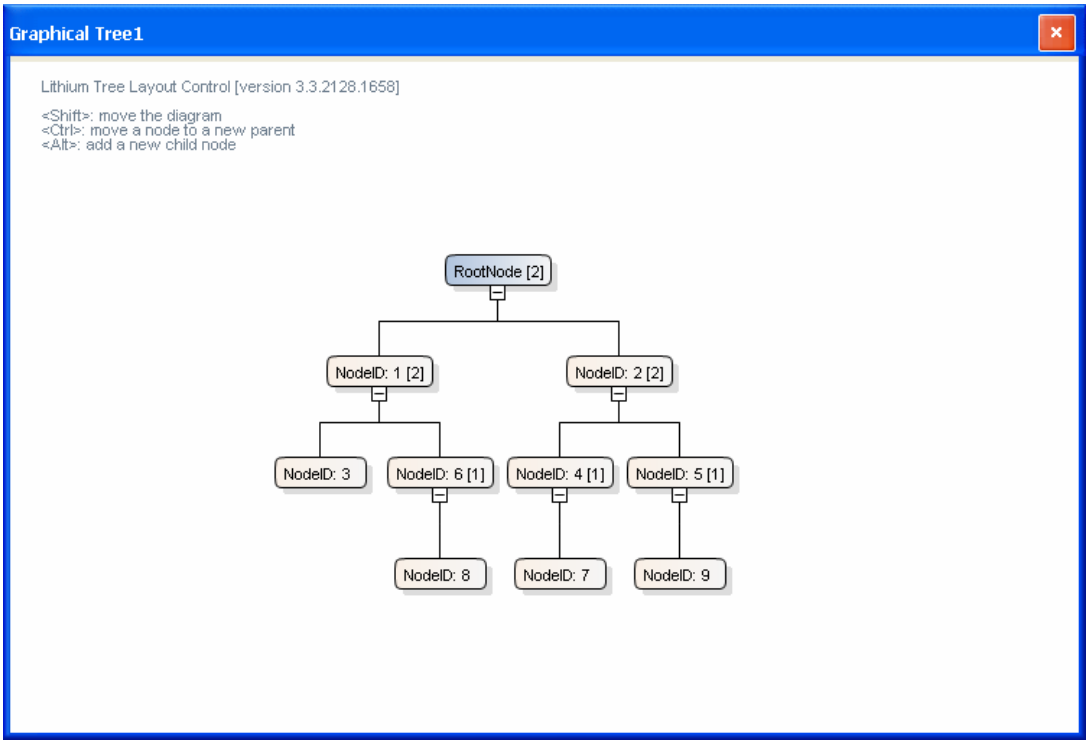


**Figura 5.15** – Diagrama de árvores.

Devido às várias iterações até se obter os resultados finais, o usuário pode selecionar se deseja ou não ter a montagem consecutiva de todas as árvores juntas num mesmo diagrama ou em cada ciclo criar uma nova árvore.

Na Figura 5.16 são apresentadas duas iterações e, conseqüentemente, para um mesmo diagrama e um mesmo computador raiz o número de computadores pertencentes à árvore é duplicado.

Caso o usuário queira verificar as características de um determinado computador, basta um clique em cima do mesmo e suas propriedades são refletidas da janela de propriedades do lado direito da árvore.

A última janela principal do simulador é utilizada para a edição de notas ou comentários significativos para cada processo de iteração (Figura 5.17).

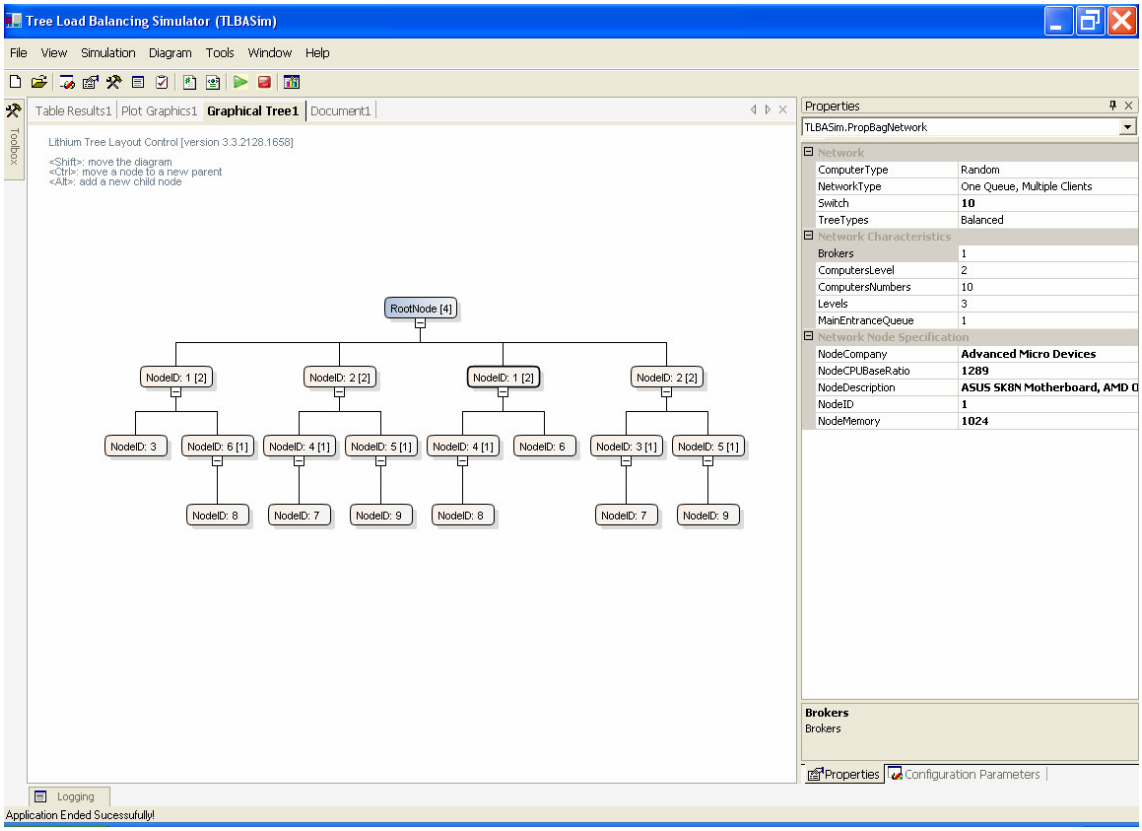


**Figura 5.16** – Diagrama de árvores – duas iterações

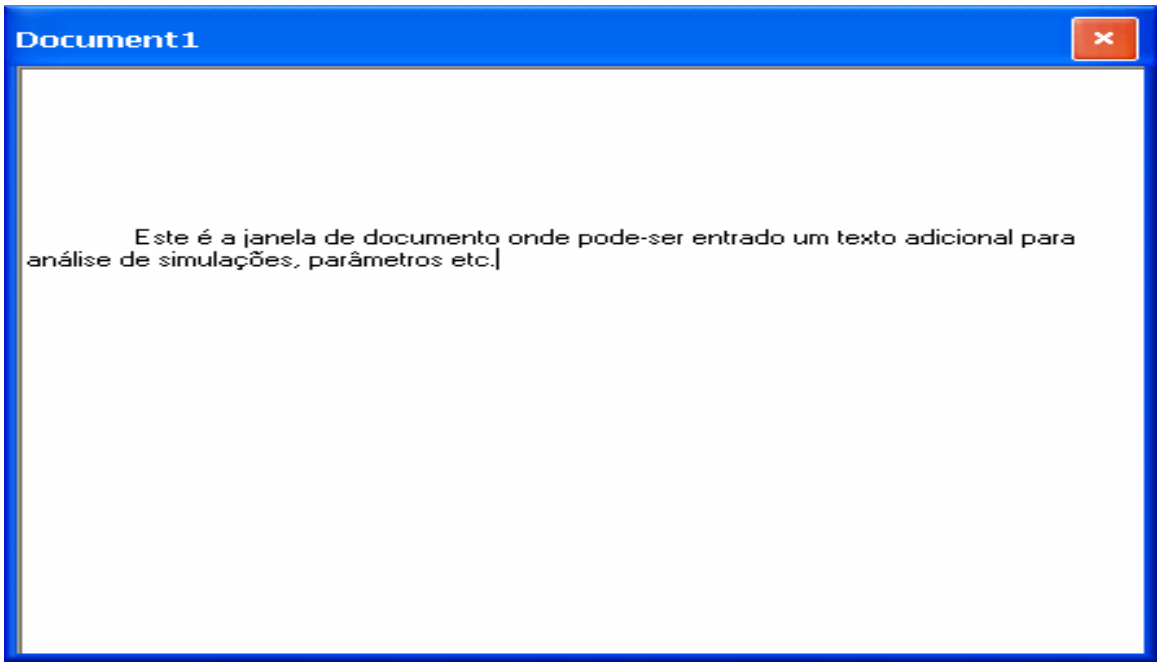


**Figura 5.17** – Janela para edição de textos

Além das janelas principais do simulador há ainda duas outras características que merecem destaque.

A primeira é uma interface que possibilita a inserção de novos computadores no ambiente de simulação como, por exemplo, o próprio computador do usuário e que fará parte do conjunto de computadores a serem simulados no ambiente (Figura 5.18).

Essa interface é feita através de uma *Schema Definition Language* (XSD) que possibilita a definição de estruturas e tipos de dados a partir de um documento em XML.

Os atributos e elementos presentes do documento XML são carregados nessa interface possibilitando a sua edição.

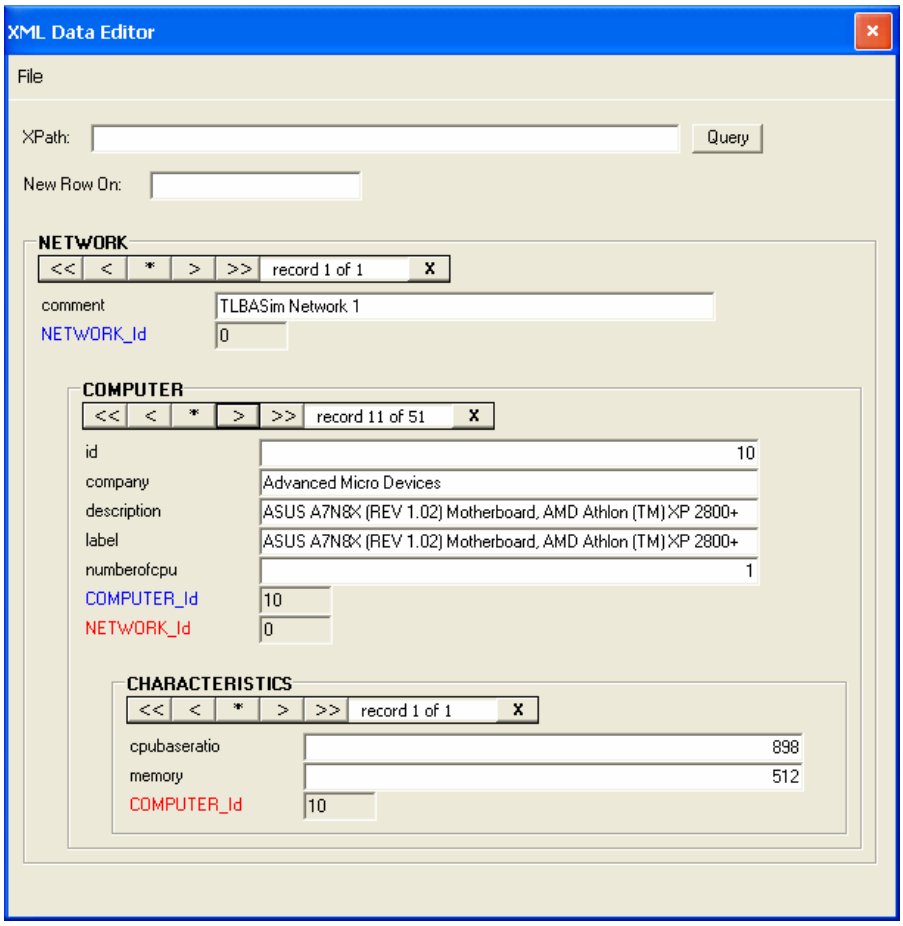


**Figura 5.18** – *Schema* para edição da Lista de Computadores em XML.

E, para finalizar, a Figura 5.19 apresenta a segunda característica adicional do simulador, uma calculadora (CODEGURU, 2005).

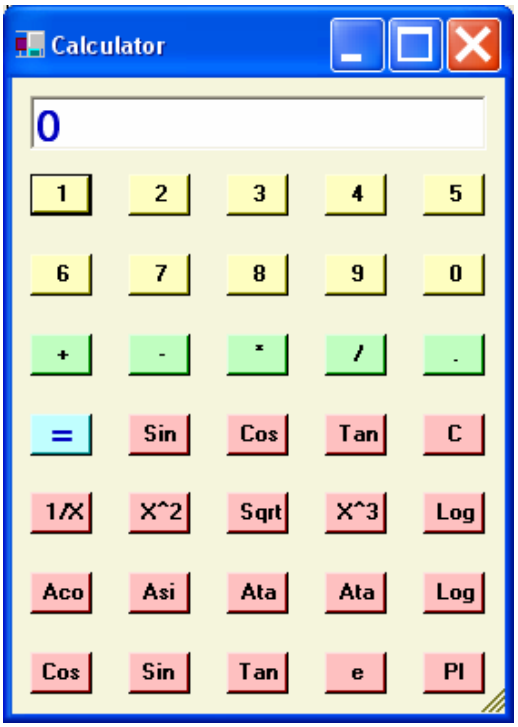


**Figura 5.19** – Calculadora de fácil acesso e utilização no TLBASim

## 5.2 Casos de Estudo para Análise do TLBA

Já com o simulador pronto e analisado a coerência de seus resultados iniciais, os passos seguintes foram o de validação do simulador através da comparação dos resultados obtidos do novo simulador com os do antigo simulador de Mello (2003) e o estudo dos parâmetros do TLBA.

Nas Seções 5.2.1 e 5.2.2 é feito o processo de validação, analisando-se os tempos médios de resposta e o número de mensagens geradas no sistema. Para tanto, o simulador implementado por Mello (2003) será definido como TLBA V1.0 e o implementado neste trabalho como TLBASim.

Já as demais Seções são relativas aos estudos da variação dos parâmetros do TLBA e fazem parte da proposta deste trabalho para ampliar os estudos desse algoritmo.

As simulações apresentadas neste trabalho foram geradas com intervalos de confiança de 95%, excetuando-se as simulações para comparação com o simulador de Mello (2003) em virtude de se manter o código original desse simulador que não apresentava tal recurso.

Desta forma é possível uma melhor análise dos resultados, garantindo que se houver uma mudança no comportamento do algoritmo será em virtude de uma determinada configuração e não do aumento do desvio padrão e variância sobre a média.

### 5.2.1 Análise do TLBA V1.0 e o TLBASim em Função da Variação do Número de Processos Submetidos ao Sistema

Essa Seção apresenta os resultados das simulações do simulador implementado por Mello (2003) (TLBAV1.0) e do novo simulador proposto neste trabalho (TLBASim[1 e 2]).

As simulações foram realizadas sobre um ambiente composto por 7 computadores, sub-divididos em 2 computadores por nível, tendo-se assim uma árvore composta de 3 níveis. Os processos chegam ao sistema através da distribuição de Poisson com média de 100s. Esses parâmetros foram mantidos para manter coerência com as simulações realizados por Mello (2003).

Para essas simulações não foi adotado o intervalo de confiança. Assim sendo, para cada caso estudado foram realizadas um número de iterações equivalente de 100 simulações.

Com relação à taxa de transferência de dados do *switch* foi adotado o valor igual a 10 Mbits/s e para o tempo de verificação de mudança de carga (*polltime*) referente ao componente *Peer* do TLBA foi adotado o valor de 1s.

Na Tabela 5.1 a 5.3 são apresentados os resultados obtidos dessas simulações.

**Tabela 5.1 –** Número de mensagens por nível e total dos simuladores em função da variação dos processos submetidos ao sistema

| Número de Processos Submetidos | Mensagens Por Nível TLBA V1.0 | Mensagens Por Nível TLBASim[1] | Desvio Padrão[1] | Total Mensagens TLBASim[2] | Desvio Padrão[2] |
|---|---|---|---|---|---|
| **10** | 11,0000 | 12,3650 | 0,0541 | 50,4450 | 4,3307 |
| **100** | 120,5000 | 120,6100 | 1,0617 | 301,9100 | 11,1961 |
| **500** | 606,0000 | 575,5900 | 3,7464 | 1255,6350 | 14,6133 |
| **1000** | 1214,0000 | 1129,9600 | 5,3541 | 2248,4350 | 16,3052 |
| **10000** | 12142,0000 | 11713,1250 | 13,5441 | 29445,7467 | 49,0398 |

As colunas representadas por TLBA V1.0 e TLBASim[1] apresentam os resultados do antigo e novo simulador e levam em consideração apenas as mensagens de balanceamento do algoritmo para cada nível dos processos submetidos ao sistema. Já a coluna TLBASim[2] considera o número total de mensagens geradas no sistema.

Essa última análise só é possível porque o novo simulador faz a análise de mensagens adicionais que estão envolvidas nas políticas de balanceamento do TLBA. Além disso, também são apresentados os desvios padrão do TLBASim[1] e TLBASim[2].

Essas mensagens adicionais são geradas quando um computador sobrecarregado solicita ao computador raiz, informações do sistema para transferência de seus processos, mensagens de atualização de carga, mensagens de balanceamento de carga (encontrar o computador mais ocioso e envio do processo), mensagem requisitando execução de processos no sistema e mensagens para inserir um novo computador na árvore.

O valor total das mensagens são os mais indicados para a análise do número de mensagens geradas pelo TLBA em um sistema distribuído e serão adotadas na análise da variação dos parâmetros desse algoritmo.

A diferença dos valores das mensagens por nível do antigo e do novo simulador deve-se ao fato de que esses valores são gerados em função do nível de cada computador escolhido pela política de localização e pelo número de mensagens de balanceamento geradas no sistema.

No TLBAV1.0 foram consideradas somente as mensagens de balanceamento de carga para o caso em que a política de transferência é ativada para a execução de processos no sistema (primeiro caso da política de transferência – ver Seção 2.3). No entanto, existe ainda o caso em que um computador sobrecarregado requisita a transferência de processos para um outro computador (segundo caso da política de transferência – ver Seção 2.3). Nesse último caso, também existe o tráfego de mensagens de balanceamento de carga na rede e que devem ser monitoradas.

A política de localização no TLBAV1.0 procura por aquele computador que apresenta o menor tempo total de execução (gerado aleatoriamente e em função da capacidade da CPU) e no TLBASim a busca é dada por aquele que apresenta a menor carga (como descrito no algoritmo). Logo, a escolha dos computadores ociosos e seus respectivos níveis serão diferentes. Devido a esse fato, o número de mensagens por nível também será diferente.

No gráfico da Figura 5.20 são apresentados os números de mensagens da Tabela 5.1. Através desse gráfico pode-se observar a sobrecarga do sistema em função do número de mensagens geradas por nível à medida que se têm um aumento no número de processos submetidos ao sistema. Assim sendo, um sistema com um baixo número de computadores e um elevado número de processos estará sobrecarregado em virtude do número de mensagens (Figura 5.22) e terá um maior tempo médio de resposta para cada processo (Figura 5.23).

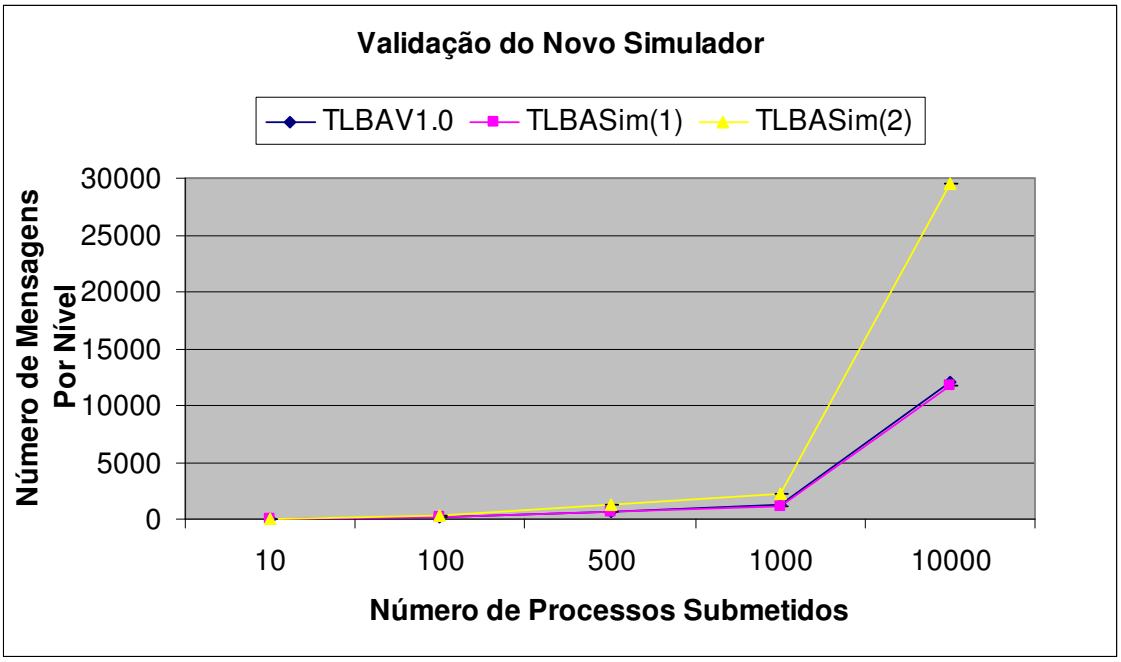


**Figura 5.20** – Variação do número de mensagens em função do número de processos

A Tabela 5.2 apresenta as taxas percentuais utilizadas de um *switch* de rede para a árvore adotada, variando-se apenas o número de processos submetidos no sistema. Os demais parâmetros seguem as mesmas características apresentadas no início dessa Seção.

**Tabela 5.2** – Percentual do *switch* utilizado (10 Mbits/s), em função da variação dos processos submetidos ao sistema

| Número de Processos Submetidos | TLBA V1.0 (%) Switch Utilizado | TLBASim[1] (%) Switch Utilizado | TLBASim[2] (%) Switch Utilizado |
|---|---|---|---|
| **10** | 0,0350 | 0,0396 | 0,1614 |
| **100** | 0,3856 | 0,3860 | 0,9661 |
| **500** | 1,9392 | 1,8419 | 4,0180 |
| **1000** | 3,8848 | 3,6159 | 7,1950 |
| **10000** | 38,8544 | 37,4820 | 94,2264 |

Como o cálculo do percentual de utilização do *switch* está diretamente relacionado com o número de mensagens geradas no sistema e como foi

explicado, anteriormente, isso ocasionará as diferenças entre os valores apresentados na Tabela 5.2.

No caso da coluna que apresenta os valores para o TLBASim[2], pode-se notar que para sistemas com um elevado número de computadores e poucos níveis na árvore lógica, o número de mensagens totais geradas pelo TLBA chega a ser quase três vezes maior do que os considerados por Mello (2003).

Esse estudo mostra uma sobrecarga na comunicação desse tipo de sistema, mas não é possível dizer se o TLBA é melhor ou pior que o *Lowest* Modificado (MELLO, TREVELIN e PAIVA, 2003b) ou outros algoritmos encontrados na literatura (SHIVARATRI, KRUEGER e SINGHAL, 1992; THEIMER e LANTZ, 1989; ZHOU e FERRARI, 1987), pois necessitaria de uma implementação equivalente envolvendo as mensagens aqui consideradas.

No gráfico da Figura 5.21 são apresentados os percentuais de utilização do *switch* de rede da Tabela 5.2. Através desse gráfico fica evidente a sobrecarga do concentrador de rede (*switch*) em virtude do número total de mensagens geradas pelo TLBA.

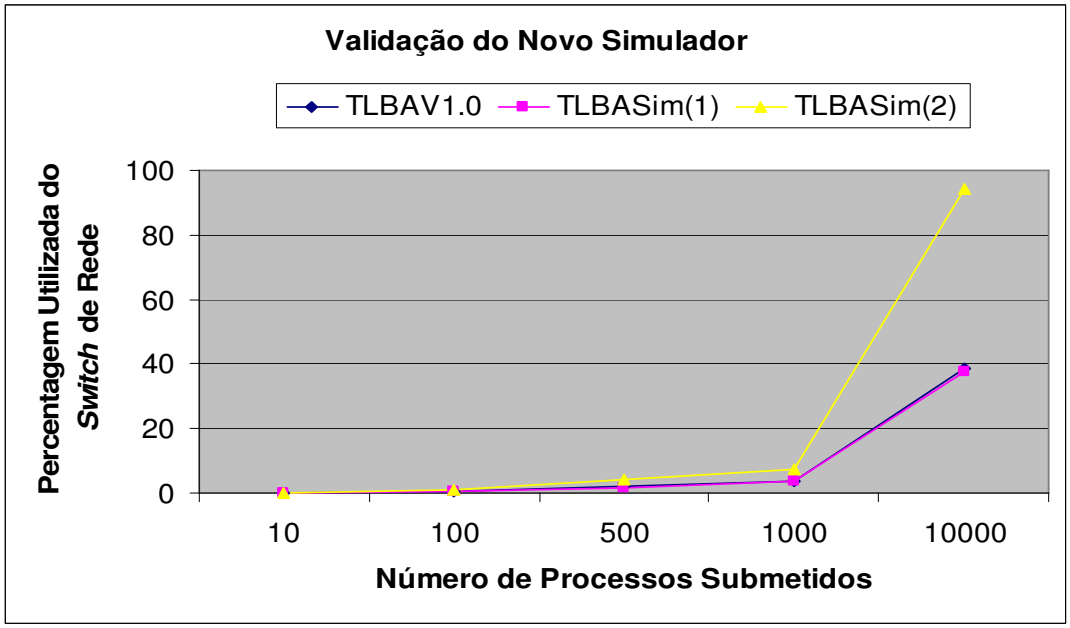


**Figura 5.21** – Percentual do *switch* de rede utilizado em função do número de processos

Com relação à Tabela 5.3 são apresentados os estudos de ambos os simuladores para os tempos médios de resposta em função dos processos enviados ao sistema.

Para essas simulações foram adotadas as mesmas características iniciais dessa Seção ressaltando-se que a ocupação dos processos por ciclo foi de $10^7$ ciclos e a capacidade de processamento dos computadores de 368 ciclos/ms (parâmetros e unidades exclusivos do TLBA V1.0), o que equivaleria a aproximadamente um computador AMD Duron de 945 MHz (Mello, 2003).

**Tabela 5.3 –** Tempo médio de resposta dos processos em função da variação do número de processos submetidos ao sistema

| Número de Processos Submetidos | TBLA V1.0 Tempo Médio de Resposta (s) | TLBASim[1] Tempo Médio de Resposta (s) | Desvio Padrão[1] | TLBASim[2] Tempo Médio de Resposta (s) | Desvio Padrão[2] |
|---|---|---|---|---|---|
| **10** | 34911,9110 | 3048,1880 | 499,6310 | 3471,4990 | 569,3240 |
| **100** | 198564,8525 | 4590,8270 | 397,2580 | 5317,1170 | 414,2070 |
| **500** | 934820,0705 | 18952,6920 | 1652,9260 | 18366,3820 | 1760,2340 |
| **1000** | 1855218,7360 | 37675,0010 | 3123,9570 | 34840,0670 | 2887,6760 |
| **10000** | 18423725,6404 | 296562,3580 | 5626,6230 | 464008,9320 | 8976,3770 |

A divergência entre os valores apresentados do novo simulador e do simulador de Mello (2003) deve-se ao fato de que esse último considera fixa a ocupação percentual de cada processo no sistema (que é da ordem de no mínimo $10^7$) o que nem sempre representa o comportamento dos sistemas reais e eleva, então, o valor final do tempo de resposta de cada ciclo de execução.

Já o novo simulador assume valores de tempos de chegadas de processos de acordo com o modelo de Feitelson (1996), capacidades computacionais obtidas de especificações de sistemas reais (ANÁLISE, 2005) e sua ocupação percentual varia para cada computador.

No gráfico da Figura 5.22 são apresentados os tempos médios de resposta da Tabela 5.3. Pelo gráfico pode-se notar uma grande variação do tempo médio de resposta entre 1000 e 10000 processos. No entanto, se for calculada a variação entre os tempos médios de resposta em função da variação do número de processos submetidos, pode-se notar que o aumento do tempo final de execução dos processos é gradual e seu comportamento é equivalente ao do antigo simulador, mesmo para sistema com um elevado número de computadores.

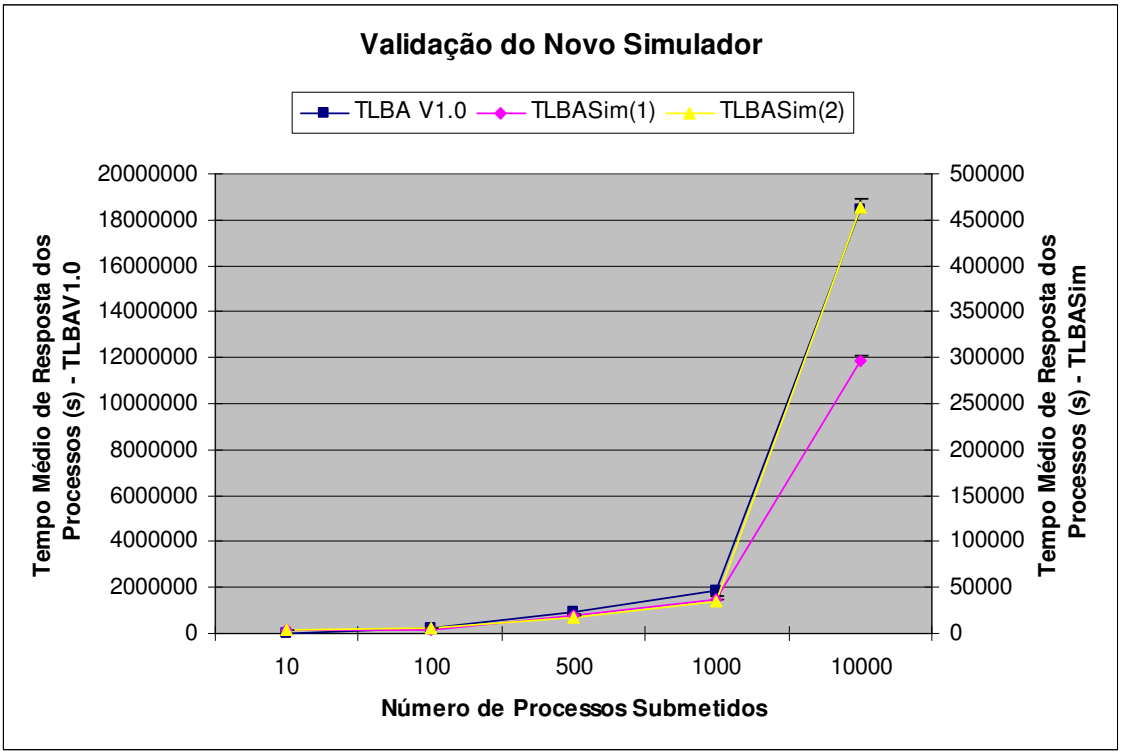


**Figura 5.22** – Tempo médio de resposta (segundos) dos processos em função do número de processos no submetidos ao sistema

### 5.2.2 Análise da Variação do Número de Computadores por Nível e de Níveis da Árvore Computacional do TLBA V1.0 com o TLBASim

Para essa Seção foram fixados os seguintes parâmetros no simulador: 500 processos foram enviados ao sistema e com tempos de chegada baseados na distribuição de Poisson com média de 100s; a taxa de transferência de dados para o *switch* foi de 10 Mbits/s e *polltime* de 1s. Para esse estudo foi escolhido um meio homogêneo (Computador ASUS A7N8X Deluxe Motherboard, AMD Athlon (TM) XP 3000+1) para manter a equivalência dos níveis entre os simuladores.

A princípio foi escolhido o valor inicial para número de computadores por nível, igual a dois. Em seguida, mantendo fixo o número de computadores por nível variou-se o número de níveis para o TLBA V1.0 e o número de computadores para o TLBASim (em ambos os casos as variações são análogas e geram o mesmo efeito). Obtidos os resultados dessa simulação, o número de computadores por nível foi incrementado e novamente o ciclo foi repetido para diferentes números de níveis e de computadores para o TLBA V1.0 e TLBASim, respectivamente. E, assim, sucessivamente.

As Tabela 5.4 a 5.6 apresentam o número médio de mensagens por nível geradas no sistema, em função da variação do número de computadores por nível e do número de níveis da árvore computacional para o TLBA V1.0 e TLBASim[1e2], respectivamente. Para o novo simulador também são apresentados seus desvios padrão.

As diferenças entre ambos os simuladores, como já explicado anteriormente, foi devido ao fato do TLBA V1.0 utilizar-se do tempo total de execução dos processos gerados aleatoriamente e utilizados na localização de computadores ociosos. Já no TLBASim[1 e 2] a localização é dada em função da carga de cada computador como descrito na política TLBA. Dessa forma, o nível do computador escolhido nem sempre foi igual e como a geração das

mensagens varia em função desse nível, isso ocasionou as diferenças entre os valores dos simuladores.

A outra diferença está relacionada a monitoração das mensagens de balanceamento de carga. No TLBA V1.0 essas mensagens são consideradas somente quando um processo requisita a sua execução no sistema, porém deve-se levar em conta também as mensagens geradas quando um computador sobrecarregado ativa a política de transferência para a migração de um processo.

**Tabela 5.4 –** Número de mensagens por nível geradas em função da variação do número de computadores por nível e do número de níveis do TLBA V1.0

| Número de Níveis | Computadores por Nível | | | | | |
|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 |
| **2** | 833,0000 | 875,0000 | 900,0000 | 916,0000 | 928,0000 | 937,0000 |
| **3** | 606,0000 | 489,3750 | 406,8000 | 345,7500 | 300,8571 | 266,0625 |
| **4** | 465,1429 | 271,6920 | 175,0476 | 119,2258 | 88,2790 | 65,5438 |
| **5** | 345,3333 | 139,6250 | 66,9412 | 36,9551 | 21,1390 | 12,7125 |
| **6** | 246,1936 | 65,5041 | 22,4076 | 8,8579 | 4,2251 | 2,1785 |

**Tabela 5.5 –** Número de mensagens por nível geradas em função da variação do número de computadores por nível e do número de níveis do TLBASim[1]

| Número de Níveis | Computadores por Nível | | | | | |
|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 |
| **2** | 834,1800 | 879,8300 | 909,0300 | 915,9400 | 922,5200 | 942,7400 |
| | ± 4,6111 | ± 3,4206 | ± 3,0800 | ± 3,3618 | ± 3,3225 | ± 2,9070 |
| **3** | 601,8550 | 488,5013 | 406,3080 | 344,2725 | 302,3057 | 266,2763 |
| | ± 3,1158 | ± 1,7936 | ± 1,4332 | ± 0,9782 | ± 0,6237 | ± 0,4531 |
| **4** | 462,9400 | 273,5415 | 175,4314 | 121,1045 | 88,4223 | 67,2579 |
| | ± 2,1072 | ± 0,6539 | ± 0,3122 | ± 0,2083 | ± 0,0984 | ± 0,0393 |
| **5** | 345,8217 | 141,4388 | 68,6782 | 36,9551 | 21,1390 | 12,7125 |
| | ± 1,3833 | ± 0,2461 | ± 0,0636 | ± 2,41E-07 | ± 9,70E-08 | ± 2,66E-08 |
| **6** | 246,4152 | 68,2785 | 22,4076 | 8,8579 | 6,9813 | 3,0328 |
| | ± 0,6683 | ± 0,0632 | ± 1,53E-07 | ± 7,59E-08 | ± 1,63E-09 | ± 2,13E-10 |

**Tabela 5.6 –** Número de mensagens por nível geradas em função da variação do número de computadores por nível e do número de níveis do TLBASim[2]

| Número de Níveis | Computadores por Nível | | | | | |
|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 |
| **2** | 1880,4000 | 1938,5600 | 1997,7400 | 2045,4500 | 2081,5800 | 2136,4700 |
| | ± 7,4148 | ± 8,4788 | ± 12,7054 | ± 14,9637 | ± 18,8038 | ± 21,2816 |
| **3** | 1236,1350 | 1108,0163 | 998,3280 | 875,3025 | 779,1214 | 685,5806 |
| | ± 15,1164 | ± 22,5716 | ± 26,5409 | ± 30,9049 | ± 28,4568 | ± 29,4522 |
| **4** | 1039,3057 | 687,2092 | 445,5762 | 315,9613 | 272,6595 | 252,8021 |
| | ± 27,3617 | ± 28,6134 | ± 23,8520 | ± 26,0111 | ± 21,0566 | ± 18,8855 |
| **5** | 843,8633 | 355,1231 | 233,8297 | 166,5941 | 112,1031 | 85,0070 |
| | ± 34,2244 | ± 27,2580 | ± 18,7234 | ± 12,2784 | ± 6,9264 | ± 4,0512 |
| **6** | 603,6339 | 233,2007 | 111,7388 | 79,1832 | 43,8717 | 27,1859 |
| | ± 34,9855 | ± 20,1650 | ± 7,6193 | ± 4,9814 | ± 2,7672 | ± 1,6232 |

No gráfico das Figuras 5.23 a 5.25 é apresentado o número de mensagens por nível das Tabelas 5.4 a 5.6, respectivamente. Esses resultados confirmam os apresentados por Mello (2003), onde árvores com muitos níveis e, conseqüentemente, poucos computadores por nível, geram uma maior sobrecarga nas buscas em profundidade utilizadas para localizar os computadores ociosos e árvores com poucos níveis geram maior número de mensagens entre níveis e diminuem a escalabilidade do ambiente.

Nas simulações do TLBA em dois níveis, o algoritmo tem estrutura semelhante ao Central (ZHOU e FERRARI, 1987). Essa estrutura apresenta baixa escalabilidade para o sistema, pois não há suportes físicos ilimitados para interconectar computadores em uma mesma rede fisicamente única. Outra limitação técnica é que, quanto maior o número de computadores, maior a cargas do computador central (MELLO, 2003; SHIVARATRI, KRUEGER e SINGHAL, 1992; ZHOU e FERRARI, 1987).

Uma das características do TLBA é sua escalabilidade, conseqüência da hierarquia em forma de árvore, que cria grupos de comunicação entre um antecessor e seus sucessores (MELLO, 2003). No entanto, grandes distâncias

médias entre os processadores e a pouca conectividade entre os nós introduzem significativas sobrecargas de comunicação na transferência de tarefas, visto que na maioria das vezes será necessária sua travessia por processadores intermediários até alcançar o computador receptor (LOH et al, 1996).

A variação de parâmetros físicos nas interconexões de rede afetam significativamente o desempenho de estratégias de balanceamento de cargas dinâmicos, sem considerar a sobrecarga dos níveis da rede (LOH et al, 1996).

Pode-se concluir que para um sistema bem dimensionado, o TLBA gera um menor número de mensagens e uma menor sobrecarga no meio de comunicação, quando em fase de balanceamento de cargas e transferência de processos.

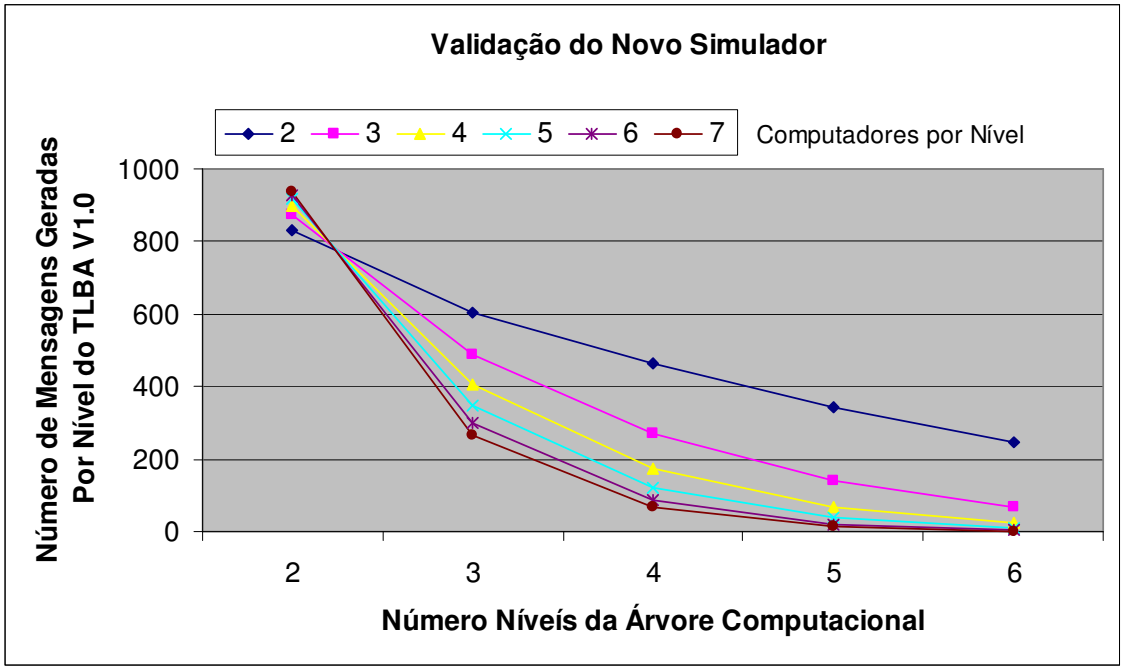


**Figura 5.23** – Número de mensagens por nível geradas em função da variação do número de níveis e computadores por nível para o TLBA V1.0

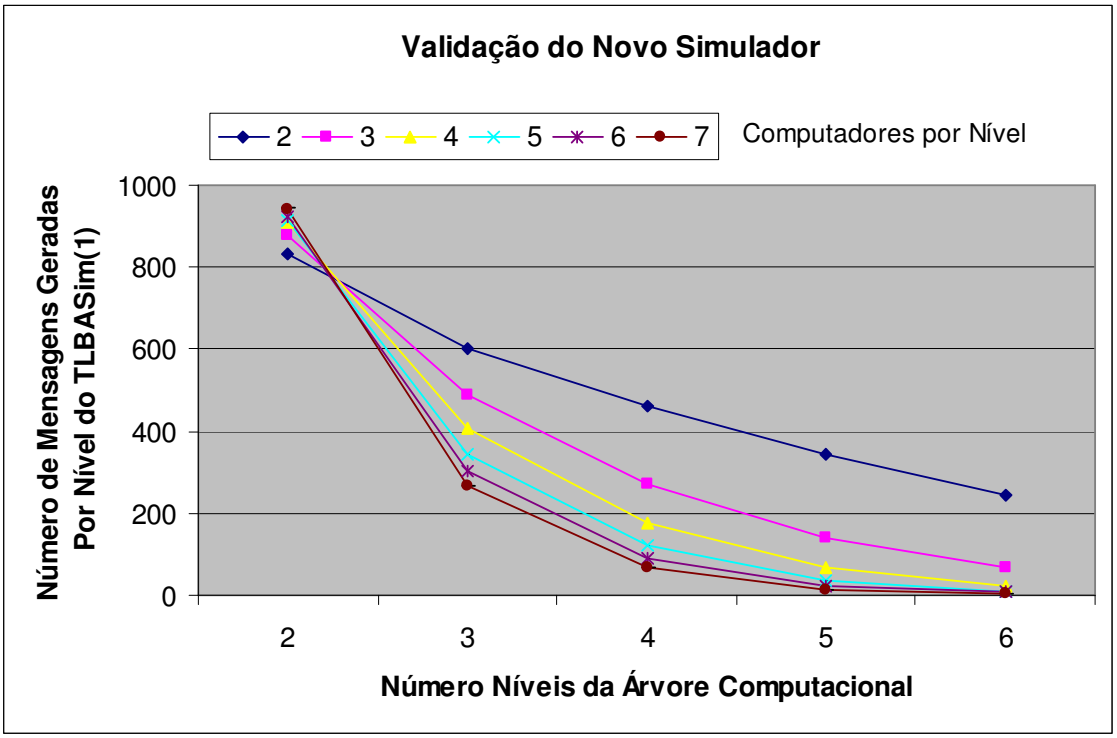


**Figura 5.24** – Número de mensagens por nível geradas em função da variação do número de níveis e computadores por nível para o TLBASim[1]

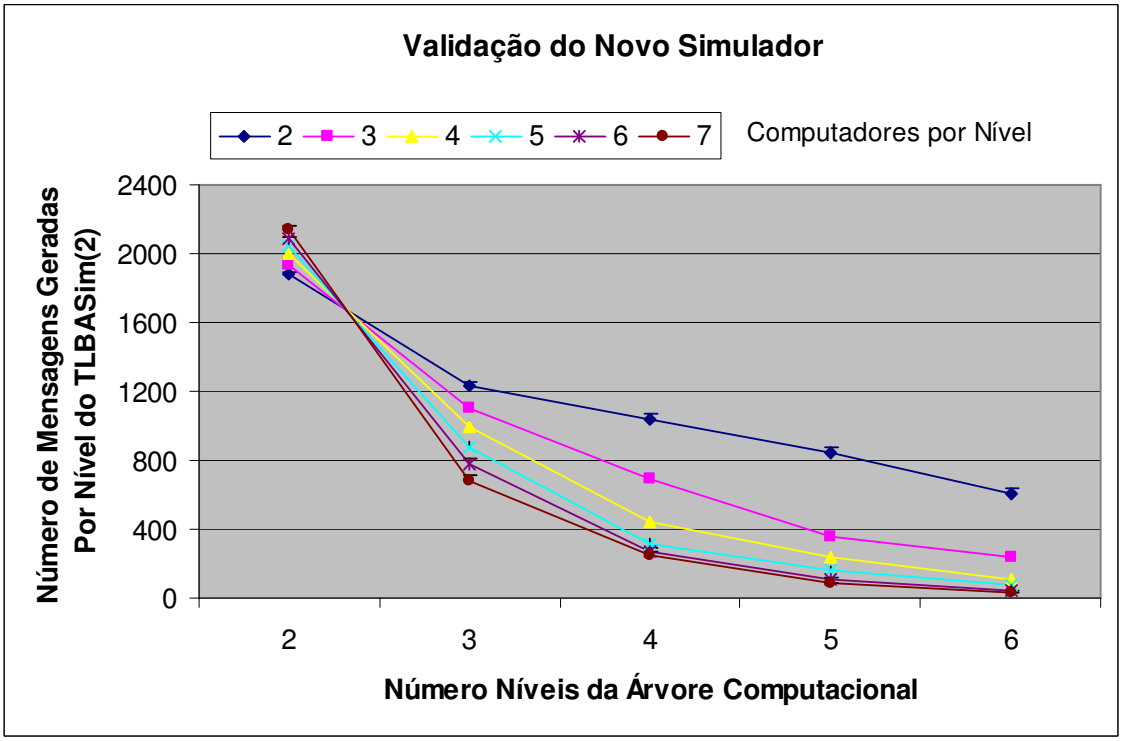


**Figura 5.25** – Número de mensagens por nível geradas em função da variação do número de níveis e computadores por nível para o TLBASim[2]

### 5.2.3 Análise da Variação dos *Thresholds* de Viabilidade de Transferência e Atualização de Carga

A partir dessa Seção são apresentados os resultados pertinentes ao trabalho proposto. Para esses estudos foram adotados intervalos de confiança de 95%. O momento em que cada processo chega ao sistema foi dado pela Distribuição de Hiperexponencial com média de 1500s, baseado nos estudos de Feitelson (1996). A taxa de transferência de dados para o *switch* foi de 100 Mbits/s por ser a mais comum nas redes atuais e *polltime* de 1s. O número sugerido de migrações permitidas foi de três (HALCHOR-BALTER e DOWNEY, 1997).

Os parâmetros específicos para essa Seção foram o de 300 processos enviados em um sistema formado por uma árvore com um total de 100 computadores, sendo quatro por nível e ambiente heterogêneo. Para os demais parâmetros foram considerados os valores *default*.

A partir das simulações foi gerado um conjunto mensagens no sistema que são apresentadas nas Tabelas 5.7 e 5.8. Dessas mensagens as que possuem relação direta com os *Thresholds* de viabilidade de transferência e de atualização de carga são as mensagens de balanceamento de carga, computadores sobrecarregados (que também podem ser consideradas como mensagens geradas devido ao balanceamento de cargas) e as de atualização de carga.

No gráfico das Figuras 5.26 e 5.27 é apresentado o número de mensagens contidas nas Tabelas 5.7 e 5.8, respectivamente. Esses resultados mostram que para um sistema em processo de balanceamento de cargas é necessária uma maior preocupação com relação ao número de mensagens geradas pela política de localização que busca pelo computador mais ocioso do sistema do que demais mensagens.

**Tabela 5.7** – Número de mensagens geradas no sistema devido a variação do *Threshold* de viabilidade de transferência

| Variação Percentual | Inserção Novo Nó | Envio de Processos | Por Nível | Nós Sobrecar-regados | Atualização de Carga | Balancea-mento de Carga |
|---|---|---|---|---|---|---|
| **5** | 200,0000 ± 0 | 600,0000 ± 0 | 251,7621 ± 1,3959 | 467,9938 ± 4,3003 | 44,4047 ± 12,5146 | 1180,0463 ± 0,7482 |
| **10** | 200,0000 ± 0 | 600,0000 ± 0 | 249,2333 ± 1,4475 | 447,4204 ± 4,5193 | 44,5401 ± 12,9202 | 1175,4495 ± 0,7338 |
| **15** | 200,0000 ± 0 | 600,0000 ± 0 | 249,0379 ± 1,4531 | 457,1064 ± 4,5363 | 46,7202 ± 13,0418 | 1161,6489 ± 0,7450 |
| **25** | 200,0000 ± 0 | 600,0000 ± 0 | 252,8726 ± 1,5484 | 491,3043 ± 4,8764 | 41,7871 ± 13,7495 | 1170,3473 ± 0,7922 |
| **35** | 200,0000 ± 0 | 600,0000 ± 0 | 253,5138 ± 1,5401 | 493,5792 ± 4,8330 | 45,1788 ± 13,6999 | 1171,0289 ± 0,8126 |
| **45** | 200,0000 ± 0 | 600,0000 ± 0 | 205,9681 ± 1,1980 | 523,0426 ± 4,9907 | 43,7559 ± 13,3811 | 1182,0565 ± 0,7266 |
| **55** | 200,0000 ± 0 | 600,0000 ± 0 | 259,1916 ± 1,5261 | 554,7258 ± 5,3483 | 39,0570 ± 13,3717 | 1172,2140 ± 0,8069 |
| **65** | 200,0000 ± 0 | 600,0000 ± 0 | 257,5950 ± 1,4618 | 539,0313 ± 5,2407 | 39,3785 ± 12,7446 | 1171,7810 ± 0,7389 |
| **75** | 200,0000 ± 0 | 600,0000 ± 0 | 259,1305 ± 1,4956 | 555,1424 ± 5,2432 | 39,3143 ± 13,1402 | 1170,9357 ± 0,7782 |
| **85** | 200,0000 ± 0 | 600,0000 ± 0 | 257,7665 ± 1,5026 | 546,6200 ± 5,4691 | 40,5255 ± 13,0961 | 1164,7431 ± 0,7179 |
| **95** | 200,0000 ± 0 | 600,0000 ± 0 | 263,6124 ± 1,4835 | 593,7404 ± 5,6957 | 36,7061 ± 12,7733 | 1179,3167 ± 0,7430 |

Para os casos onde é aplicado um *Threshold* para limitar o número de migrações permitidas e outro para avaliar a carga ociosa de um determinado computador com relação a árvore do sistema (*Threshold* de viabilidade de transferência), o número de mensagens de balanceamento será menor e a variação do *Threshold* de transferência não afetará de maneira significativa o comportamento do sistema.

Com relação às mensagens de inserção de computadores e envio de processos elas foram consideradas como constantes no sistema e por isso não há variação com a mudança dos *Thresholds* e seu desvio é sempre nulo.

Já com a variação do *Threshold* de Atualização de Carga, houve uma tendência crescente para as mensagens de atualização e de computadores sobrecarregados, porém se comparadas com as mensagens de balanceamento de cargas, pouco afetariam o desempenho do sistema.

**Tabela 5.8** – Número de mensagens geradas no sistema devido a variação do *Threshold* de atualização de carga

| Variação Percentual | Inserção Novo Nó | Envio Processos | Por Nível | Nós Sobrecar-regados | Atualização de Carga | Balancea-mento de Cargas |
|---|---|---|---|---|---|---|
| **5** | 200,0000 ± 0 | 600,0000 ± 0 | 265,0389 ± 5,7793 | 485,6566 ± 4,6752 | 164,6782 ± 56,4447 | 1173,5508 ± 0,7805 |
| **10** | 200,0000 ± 0 | 600,0000 ± 0 | 258,1301 ± 3,0410 | 491,3081 ± 4,7000 | 90,2894 ± 29,0211 | 1173,8900 ± 0,7359 |
| **15** | 200,0000 ± 0 | 600,0000 ± 0 | 253,8004 ± 2,2694 | 478,4058 ± 4,6704 | 72,0749 ± 21,3813 | 1162,1435 ± 0,8332 |
| **25** | 200,0000 ± 0 | 600,0000 ± 0 | 254,5023 ± 1,4470 | 496,5607 ± 4,6619 | 44,2017 ± 12,8399 | 1178,8099 ± 0,8035 |
| **35** | 200,0000 ± 0 | 600,0000 ± 0 | 250,8860 ± 1,0906 | 474,8215 ± 4,6789 | 32,2285 ± 9,0483 | 1176,7216 ± 0,8116 |
| **45** | 200,0000 ± 0 | 600,0000 ± 0 | 250,8369 ± 0,8816 | 486,3281 ± 4,5688 | 22,3156 ± 6,7073 | 1174,6417 ± 0,7161 |
| **55** | 200,0000 ± 0 | 600,0000 ± 0 | 249,7298 ± 0,7436 | 482,8826 ± 4,5556 | 17,7003 ± 5,0416 | 1171,7425 ± 0,7826 |
| **65** | 200,0000 ± 0 | 600,0000 ± 0 | 250,5222 ± 0,7027 | 490,9253 ± 4,7308 | 14,1283 ± 4,2517 | 1175,1157 ± 0,7586 |
| **75** | 200,0000 ± 0 | 600,0000 ± 0 | 250,0190 ± 0,6441 | 492,6856 ± 4,7265 | 10,9989 ± 3,3728 | 1171,5037 ± 0,7679 |
| **85** | 200,0000 ± 0 | 600,0000 ± 0 | 198,0151 ± 0,4691 | 474,6472 ± 4,6501 | 9,6089 ± 2,6972 | 1166,1807 ± 0,7598 |
| **95** | 200,0000 ± 0 | 600,0000 ± 0 | 247,6787 ± 0,5427 | 475,5400 ± 4,6112 | 6,5084 ± 1,8014 | 1169,9705 ± 0,8129 |

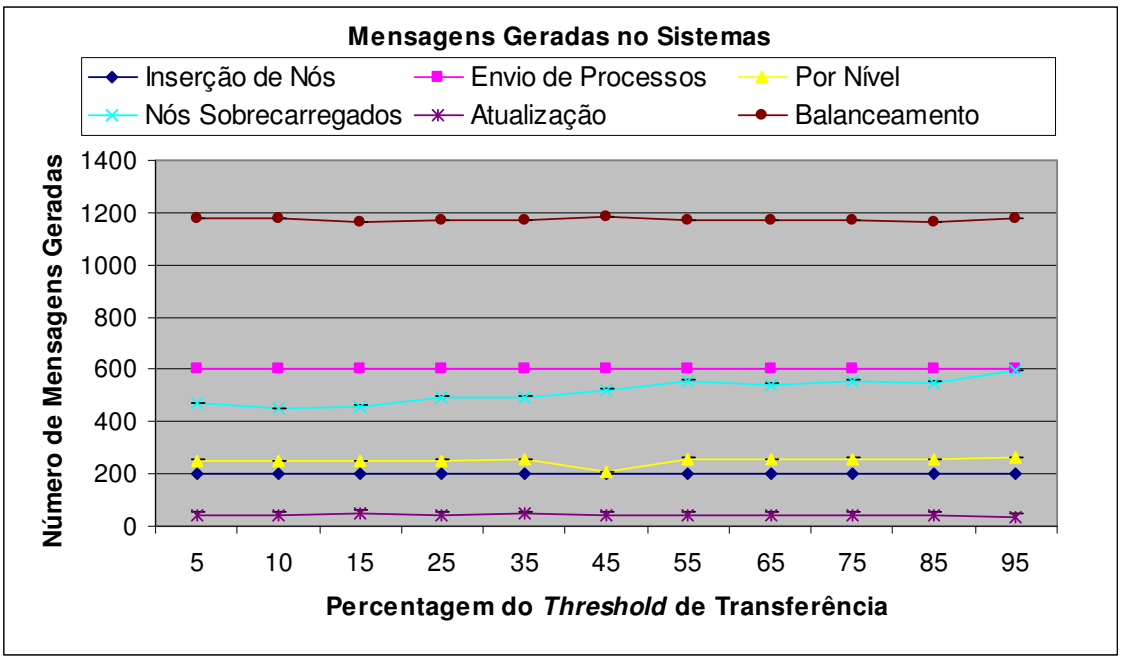


**Figura 5.26** – Número de mensagens geradas em função da variação do *Threshold* de transferência

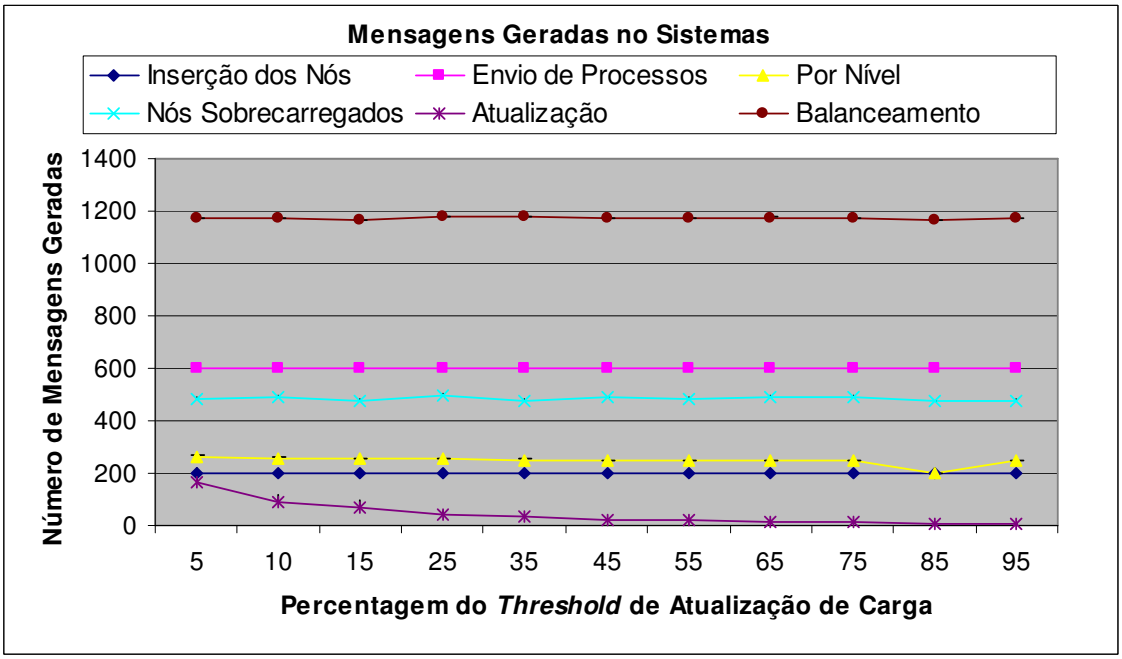


**Figura 5.27** – Número de mensagens por nível geradas em função da variação do *Threshold* de atualização de carga

Para o caso das migrações, como já era esperado, à medida que se aumenta o percentual do *Threshold* de transferência ocorre a diminuição no número de migrações de processos. Como esse aumento no *Threshold* reflete diretamente no aumento da capacidade média ociosa dos computadores de toda a árvore (ver eq.(8)) fica cada vez mais difícil da capacidade média do computador sobrecarregado ser inferior a esse valor e, conseqüentemente, haverá a diminuição no número de migrações (ver Tabela 5.9). Já com relação ao *Threshold* de atualização, esse não afetou o número de migrações mantendo seu valor médio próximo um do outro (ver Figura 5.28).

Apesar do número de migrações ter sido maior para um menor o *Threshold* esse valor não alterou significativamente o tempo médio final de resposta de cada processo, visto que o número máximo de migrações para cada processo foi limitado em 3 não trazendo custos significativos para a transferência de processos (ver Tabela 5.10).

**Tabela 5.9** – Número de migrações realizadas durante a variação dos *Thresholds* de transferência e atualização de carga

| Variação Percentual | *Threshold* de Transferência | | *Threshold* de Atualização | |
|---|---|---|---|---|
| | Número de Migrações | Desvio Padrão | Número de Migrações | Desvio Padrão |
| **5** | 84,9013 | 1,0306 | 69,1663 | 0,8222 |
| **10** | 78,8376 | 0,9840 | 69,6355 | 0,8402 |
| **15** | 79,5042 | 0,9417 | 71,4974 | 0,8506 |
| **25** | 77,8446 | 0,9459 | 71,0011 | 0,8356 |
| **35** | 80,3344 | 0,9720 | 70,1267 | 0,8051 |
| **45** | 75,4989 | 0,8860 | 70,4656 | 0,8234 |
| **55** | 71,3458 | 0,8656 | 68,0521 | 0,8284 |
| **65** | 71,5387 | 0,8674 | 71,5924 | 0,8396 |
| **75** | 67,5117 | 0,8202 | 69,7687 | 0,8329 |
| **85** | 64,9386 | 0,7988 | 71,4305 | 0,8476 |
| **95** | 68,5201 | 0,7788 | 70,0065 | 0,8352 |

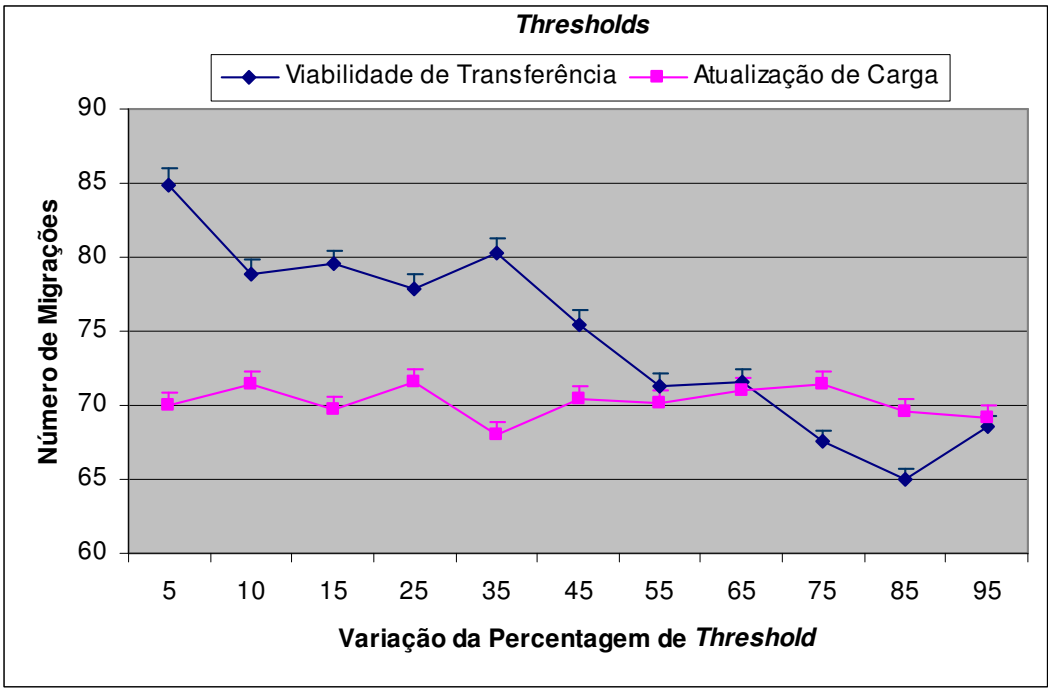


**Figura 5.28** – Número de migrações realizadas durante a variação dos *Thresholds* de transferência e atualização de carga.

**Tabela 5.10** – Tempo médio de resposta (s) em função da variação dos *Thresholds* de transferência e atualização de carga

| Variação Percentual | TMR para Variação Threshold de Transferência | Desvio Padrão | TMR para Variação Threshold de Atualização | Desvio Padrão |
|---|---|---|---|---|
| **5** | 3442,5530 | 87,7670 | 3412,7200 | 86,9710 |
| **10** | 3634,5570 | 92,6210 | 3580,8320 | 91,3000 |
| **15** | 3609,7160 | 92,0410 | 3552,7380 | 90,5940 |
| **25** | 3545,1940 | 90,3930 | 3754,0440 | 95,6730 |
| **45** | 3559,9950 | 90,7930 | 3539,0480 | 90,2360 |
| **55** | 3459,3640 | 88,2010 | 3368,8060 | 85,9230 |
| **65** | 3662,4680 | 93,4130 | 3518,5960 | 89,7180 |
| **75** | 3440,5900 | 87,7530 | 3378,5600 | 86,1460 |
| **85** | 3397,5920 | 86,5730 | 3465,4780 | 88,3710 |
| **95** | 3515,7730 | 89,6260 | 3473,5270 | 88,5790 |

### 5.2.4 Análise da Variação de Computadores no Sistema

Para essa Seção foram fixados os seguintes parâmetros: foram enviados 100 processos que chegavam no sistema através da distribuição de Hiperexponencial com média de 1500s, conforme o modelo de Feitelson (1996). O intervalo de confiança adotado foi de 95%. A taxa de transferência de dados para o *switch* foi de 100 Mbits/s. Todas as simulações foram realizadas para um ambiente heterogêneo de computadores. Para os demais parâmetros foram considerados os valores *default*.

Essas simulações foram realizadas em função da variação dos número de computadores da árvore computacional e, também, da variação do número máximo de computadores por nível. Os resultados obtidos para esse caso de estudo são apresentados nos gráficos das Figuras 5.29 a 5.33, levando-se em consideração a análise do tempo médio de resposta e do número de mensagens geradas no sistema.

Pela Figura 5.29 é possível notar que o número de mensagens geradas por nível no sistema tende a diminuir à medida que novos computadores são adicionados ao sistema. Esse resultado comprova novamente que árvore com muitos níveis e poucos computadores por nível, geram sobrecarga nas buscas em profundidade utilizadas para localizar os computadores ociosos (Mello, 2003).

Por meio desses resultados é possível concluir que a escalabilidade de um sistema estará assegurada à medida que se tem um elevado número de computadores, pois o número de mensagens por nível será menor quanto maior for o sistema. Além disso, o número de computadores por nível somente influenciará no número de mensagens por nível em sistemas de pequeno porte, onde se terá um aumento do número de mensagens geradas por nível.

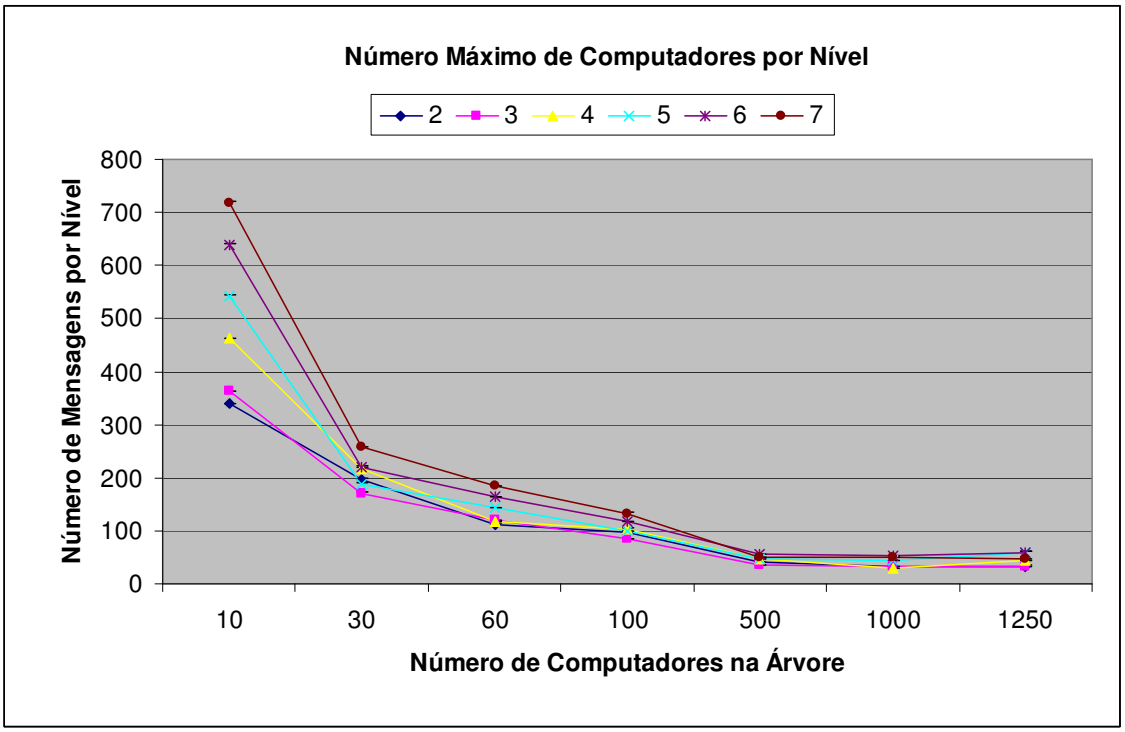


**Figura 5.29** – Número de mensagens por nível geradas em função da variação do número de máximo de computador por nível e na árvore

Na Figura 5.30 é apresentado o número de mensagens geradas por computadores sobrecarregados tende a aumentar à medida que se aumenta o número de computadores na árvore lógica e quanto menor for o número de computadores por nível maior será o número de mensagens de computadores sobrecarregados que buscarão compartilhar sua carga com os outros computadores do sistema.

Com esses resultados também é possível de se observar que quando o número de processos no sistema está próximo ao número de computadores na árvore lógica há uma diminuição do tráfego de mensagens de computadores sobrecarregados. Assim sendo, o sistema apresenta um melhor desempenho se o número de processos for equivalente ao número de computadores, ou melhor, se todos os computadores possuírem uma carga igualmente distribuída ou balanceada.

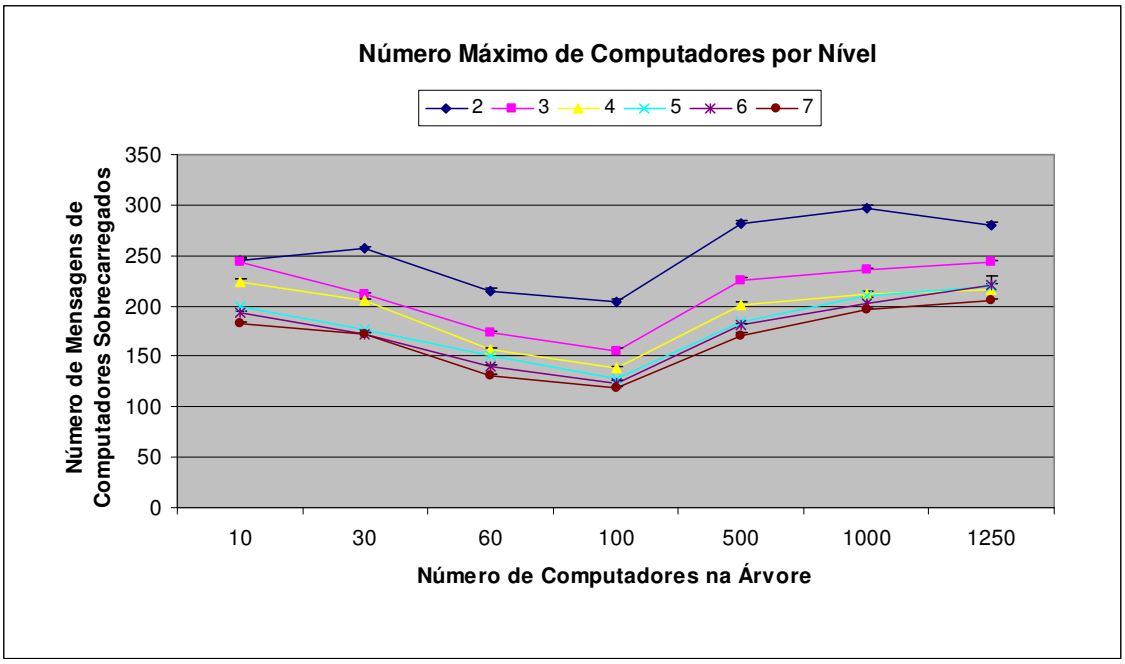


**Figura 5.30** – Número de mensagens geradas por computadores sobrecarregados em função da variação do número de máximo de computadores por nível e na árvore

Na Figura 5.31 é apresentado o tempo médio de resposta para um sistema onde se tem um aumento do número de computadores por nível e no número de computadores da árvore lógica. Através desse gráfico é possível concluir que o TLBA não é recomendado para sistemas de pequeno porte, ou seja, redes formadas por poucos computadores ou que tenham uma grande concentração de computadores por nível.

A aplicação do TLBA para esse tipo de sistema irá piorar o seu desempenho aumentando o número de mensagens de balanceamento de carga e terá um aumento no tempo médio de resposta dos processos.

Pelas Figuras 5.32 e 5.33, pode-se notar uma estabilização do número de mensagens de atualização e de balanceamento de cargas a partir do momento em que o número de computadores é maior que o número de processos submetidos ao sistema.

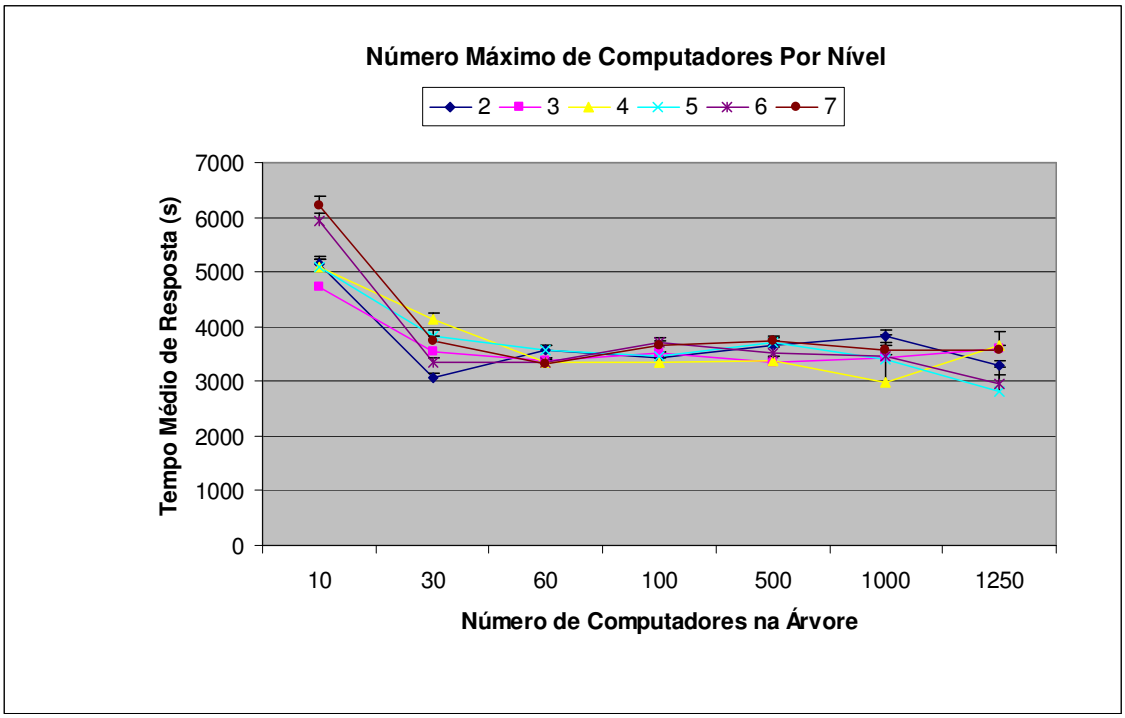


**Figura 5.31** – Tempo médio de resposta (s) em função da variação do número de máximo de computador por nível e na árvore

Os comportamentos apresentados nos gráficos das Figuras 5.32 e 5.33 são em virtude de que a partir do momento que o número de computadores supera o número de processos presentes no sistema não há mais necessidade de migração de processos e, por conseguinte, não há mais necessidade de balanceamento de carga no sistema.

Por outro lado, é por esse motivo também que se deve fixar o número de migração de processos entre os computadores, pois neste caso novas migrações somente viriam a prejudicar o sistema e não agregariam benefícios de desempenho em sua execução. Além disso, sistemas desse tipo não apresentam uma sobrecarga significante para que as políticas de balanceamento atuem de forma a melhor distribuir os recursos entre os computadores.

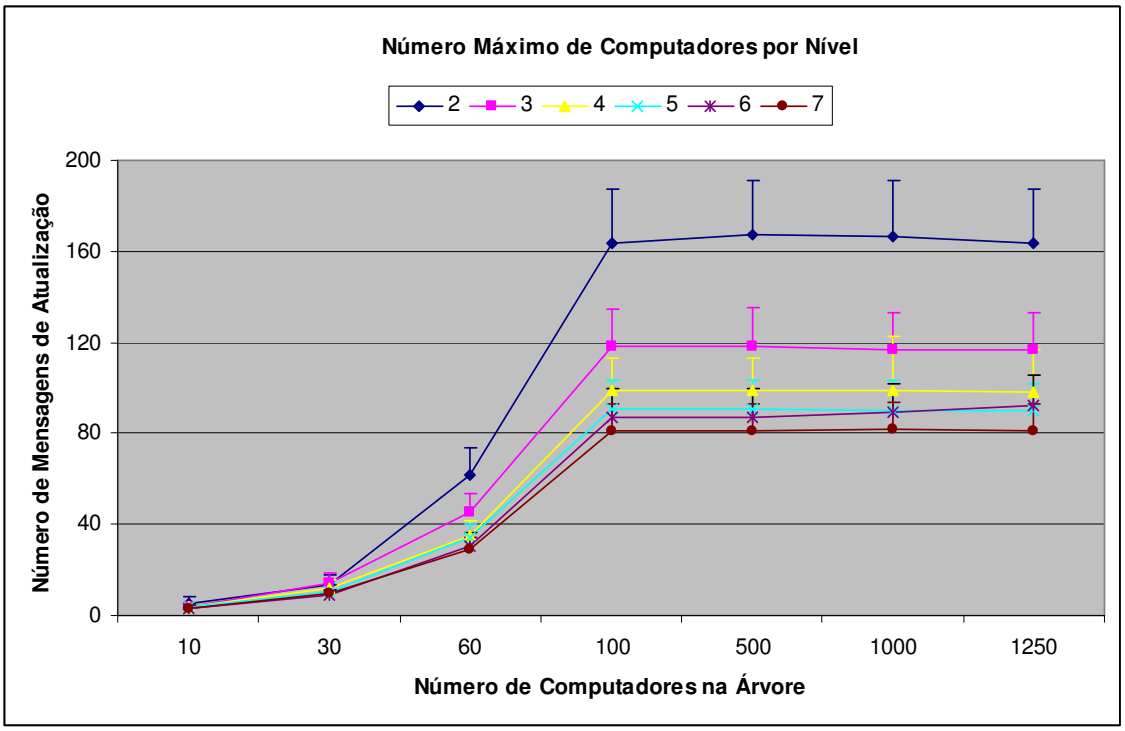


**Figura 5.32** – Número de mensagens de atualização geradas em função da variação do número de máximo de computador por nível e na árvore

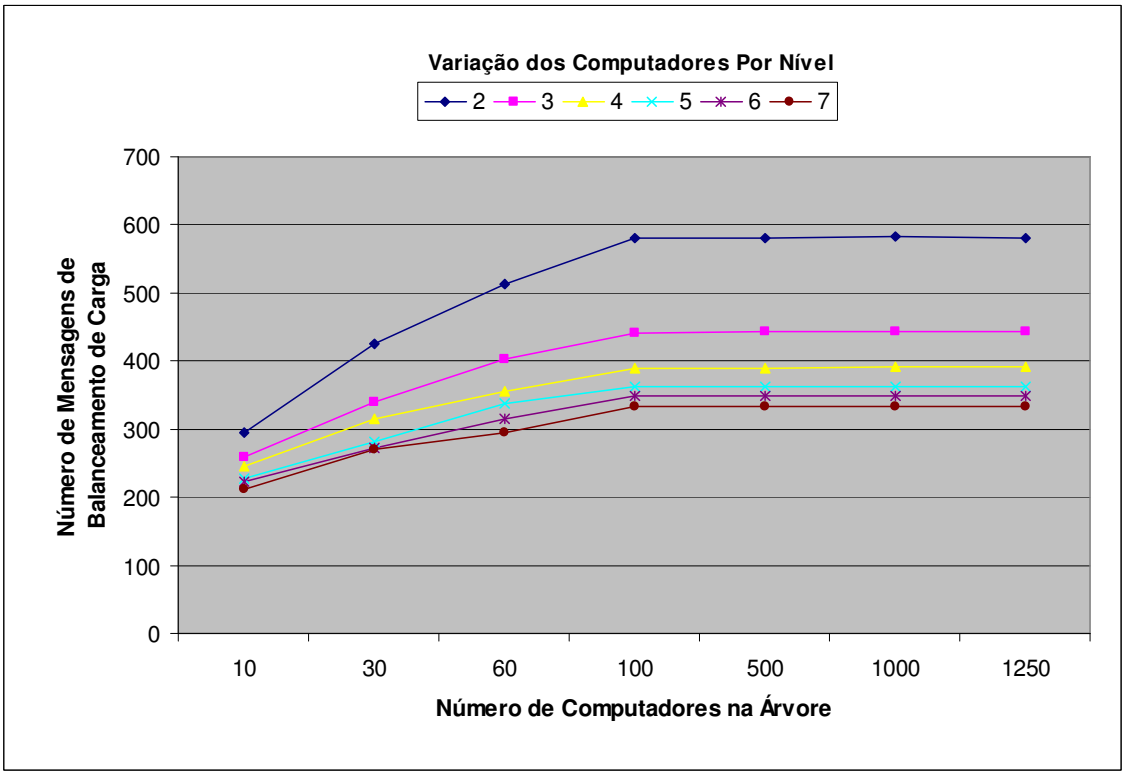


**Figura 5.33** – Número de mensagens de balanceamento de carga em função da variação do número de máximo de computador por nível e na árvore

### 5.2.5 Análise da Variação do Número de Migrações

A Tabela 5.11 apresenta os resultados obtidos do número mensagens geradas no sistema, da média de migrações realizadas por ciclo de simulação e do tempo médio de resposta dos processos em função da variação do número de migrações permitidas para um ambiente heterogêneo, composto de um total de 7 computadores, sendo no máximo 2 computadores por nível de computadores, com 300 processos que chegavam através de uma distribuição de Hiperexponencial com média de 1500s. Os demais parâmetros forma considerados como os valores *default*.

A partir desses valores, foi possível a elaboração dos gráficos apresentados nas Figuras 5.34 e 5.35. Como pode ser visto pela Figura 5.34, a variação do número de migrações não causa grandes impactos no número de mensagens geradas no sistema. Como já foi visto anteriormente, esse aumento do número de mensagens está mais relacionado à quantidade de processos que são submetidos para execução, à quantidade de computadores e o número máximo de computadores por níveis do sistema.

Por outro lado, o aumento excessivo do número de migrações permitidas não agrega benefícios de desempenho na execução dos processos, uma vez que se tem um significante aumento do tempo médio de resposta de cada processo (Figura 5.35). Por isso a limitação do número de migrações entre 3 e 5, é uma boa escolha para melhorar o tempo de execução dos processos e do desempenho do sistema quando se está utilizado o TLBA. Essa solução do número máximo de transferência por processo é apresentada e discutida por Shivaratri, Krueger e Singhal (1992).

Pelo gráfico da Figura 5.35 fica evidente que o balanceamento de cargas através de um determinado número de migrações de processos traz uma melhora significativa no desempenho do sistema que pode ser de aproximadamente 61%, como é visto para os casos de não se ter nenhuma migração no sistema ou de se ter até no máximo 5 migrações por processo.

**Tabela 5.11** – Tempo médio de resposta (s), número de migrações realizadas e número de mensagens geradas no sistema devido à variação do número máximo de migrações permitidas

| Número Migrações Permitidas | Tempo Médio de Resposta (s) | Média de Migrações Processos | Mensagens Por Nível | Mensagens Comp. Sobre-carregado | Mensagens de Atualização | Mensagens Balancea-mento |
|---|---|---|---|---|---|---|
| **0** | 17696,5790 ± 451,2310 | 0,0000 ± 0 | 685,4053 ± 1,2508 | 17,2979 ± 0,1027 | 6,1001 ± 1,9997 | 733,4125 ± 1,5326 |
| **1** | 12463,2220 ± 317,6500 | 96,7080 ± 1,3744 | 807,6632 ± 2,2110 | 281,3457 ± 3,7684 | 3,3574 ± 1,3801 | 716,6233 ± 1,3879 |
| **3** | 10854,2760 ± 276,7640 | 95,9360 ± 1,7434 | 802,6688 ± 2,5566 | 280,1967 ± 4,4451 | 4,1110 ± 1,6711 | 707,0300 ± 1,3568 |
| **5** | 10819,7530 ± 276,0030 | 101,3277 ± 2,1270 | 839,6952 ± 3,1066 | 301,9862 ± 5,6836 | 3,5561 ± 1,8365 | 759,8480 ± 1,3339 |
| **10** | 12106,6190 ± 308,8190 | 104,3299 ± 2,0916 | 1158,5962 ± 4,9533 | 356,3572 ± 6,4486 | 4,1134 ± 2,4010 | 763,4237 ± 2,0103 |
| **20** | 11096,8850 ± 283,0220 | 89,3931 ± 1,9050 | 802,6500 ± 2,8627 | 269,7191 ± 5,1212 | 3,1008 ± 1,8242 | 718,4802 ± 1,3745 |
| **25** | 12002,9930 ± 306,0800 | 105,1838 ± 2,3158 | 858,7313 ± 3,5163 | 320,9329 ± 6,3985 | 3,7517 ± 2,3953 | 778,7779 ± 1,2012 |
| **30** | 12409,2600 ± 316,5090 | 83,1103 ± 2,0850 | 806,0720 ± 2,9945 | 236,8377 ± 5,4148 | 4,5634 ± 1,9999 | 756,7429 ± 1,4368 |
| **35** | 12327,6060 ± 314,4050 | 99,0169 ± 2,4184 | 837,1420 ± 3,7232 | 302,1754 ± 6,7201 | 3,4346 ± 2,3643 | 754,6741 ± 1,3954 |
| **40** | 12036,9710 ± 306,9280 | 67,3453 ± 1,7725 | 782,6779 ± 2,7273 | 213,3671 ± 4,7290 | 3,8540 ± 2,0675 | 734,1346 ± 1,3024 |
| **45** | 11984,3270 ± 305,7040 | 106,9661 ± 2,4443 | 849,9881 ± 3,6195 | 325,5229 ± 6,6722 | 4,1914 ± 2,3503 | 756,2619 ± 1,4163 |
| **50** | 11293,4860 ± 287,9240 | 101,0340 ± 1,9528 | 821,0140 ± 2,8379 | 302,2024 ± 5,1254 | 3,1495 ± 1,7882 | 722,6760 ± 1,3530 |
| **70** | 11848,8890 ± 302,1720 | 122,8306 ± 2,3263 | 1217,2011 ± 5,2563 | 407,6172 ± 7,0259 | 3,6883 ± 2,6925 | 800,4960 ± 1,8194 |
| **100** | 13286,0040 ± 338,8700 | 107,1923 ± 2,4011 | 827,2207 ± 3,5135 | 308,6769 ± 6,1789 | 4,6356 ± 2,4966 | 727,1288 ± 1,5956 |
| **150** | 13060,4140 ± 332,9060 | 69,7448 ± 1,7158 | 795,4987 ± 2,6907 | 222,1944 ± 4,7213 | 3,7361 ± 1,8863 | 751,0668 ± 1,2157 |
| **200** | 13697,0150 ± 348,9860 | 117,6892 ± 2,5148 | 1148,2066 ± 4,9549 | 344,1302 ± 6,5415 | 4,8840 ± 2,7334 | 759,2956 ± 1,7423 |
| **250** | 10971,6000 ± 279,6640 | 63,0488 ± 1,5457 | 1046,9099 ± 3,4599 | 211,8949 ± 4,3728 | 4,1029 ± 1,7662 | 740,3671 ± 1,5409 |
| **300** | 11823,9910 ± 301,5400 | 118,7009 ± 2,3493 | 849,3880 ± 3,5962 | 353,3639 ± 6,4616 | 4,3370 ± 2,0343 | 727,0750 ± 1,5818 |
| **350** | 10806,1800 ± 275,5350 | 88,0996 ± 1,9279 | 804,4083 ± 2,8589 | 270,0204 ± 5,1087 | 3,4687 ± 1,7326 | 721,3275 ± 1,3026 |
| **500** | 11667,7750 ± 297,4560 | 119,6084 ± 2,3746 | 835,1897 ± 3,4277 | 343,0096 ± 6,0418 | 3,2491 ± 2,3376 | 710,1206 ± 1,4834 |

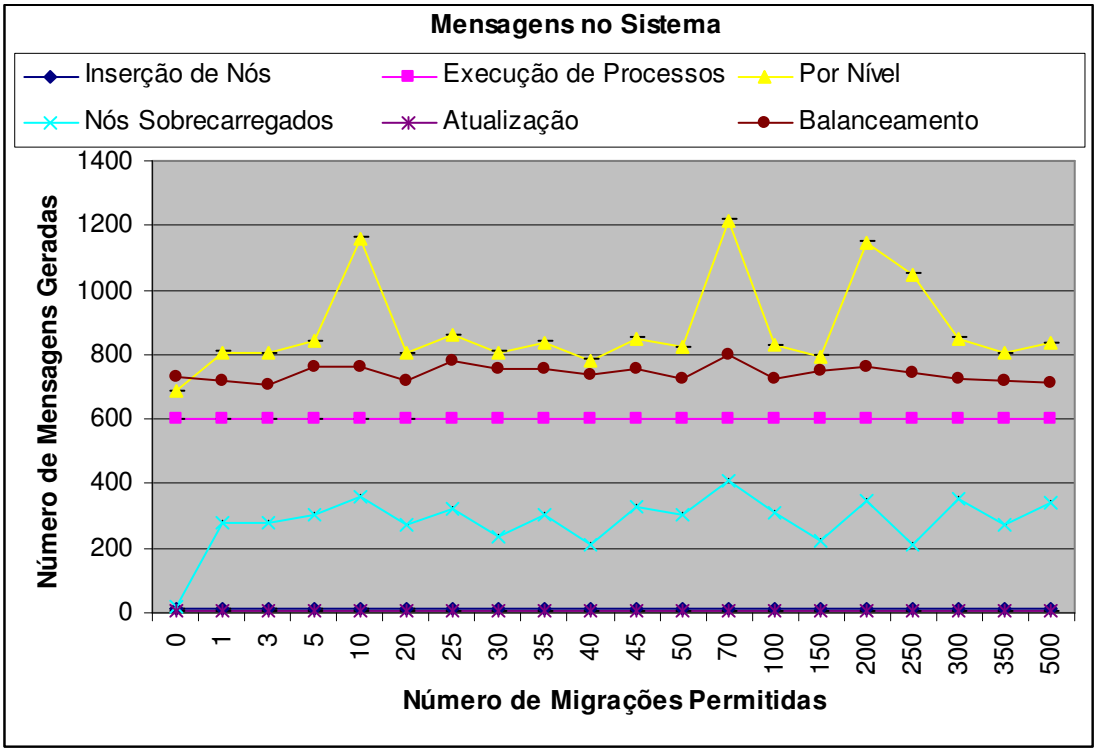


**Figura 5.34** – Número de mensagens geradas em função da variação do número de migrações permitidas

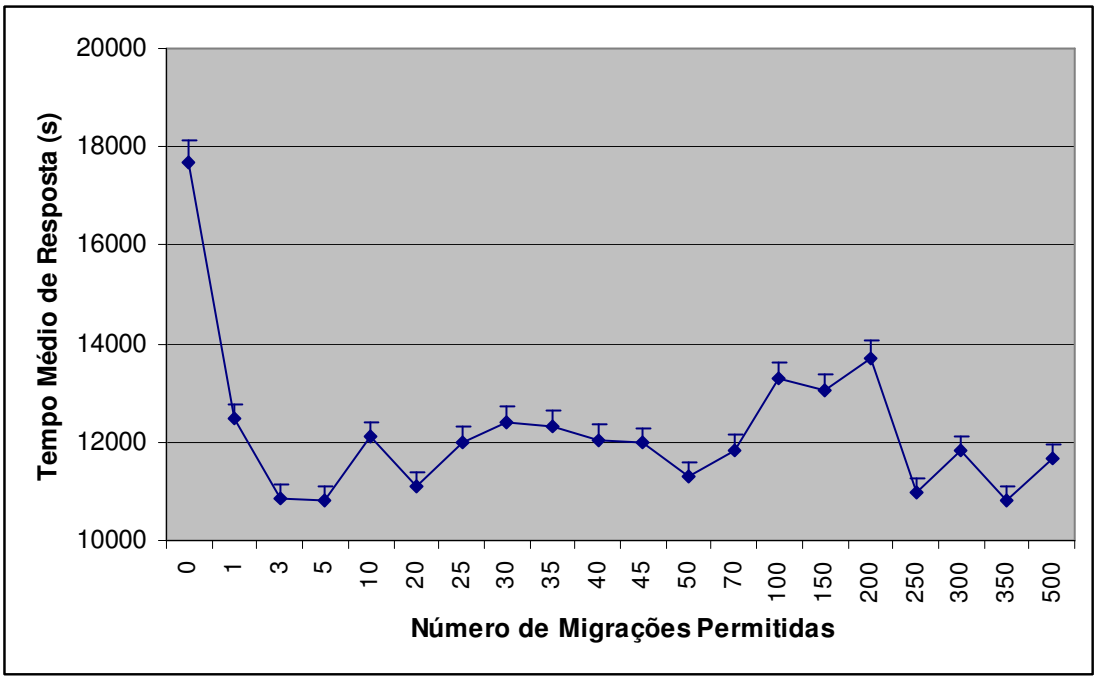


**Figura 5.35** – Tempo médio de resposta (s) em função da variação do número máximo de migrações permitidas

### 5.2.6 Análise de Sistemas Homogêneos e Heterogêneos

As simulações dessa Seção permitem analisar os impactos apresentados a um sistema composto por computadores homogêneos e heterogêneos. As simulações propostas são subdividas nos seguintes grupos: pequenos, médios e grandes sistemas homogêneos e heterogêneos.

O número de computadores para os sistemas variou de 10, 100, 500, 1000 e 5000, onde os dois primeiros casos são considerados como sistemas de pequeno porte, os dois seguintes como de médio porte e acima de 5000 computadores como de grande porte.

O número de processos submetidos foi igual a 300 e chegavam ao sistema através da distribuição de Hiperexponencial com média de 1500s. O número de computadores variou, porém foi adotado o número máximo de 4 computadores por nível para todos os casos e o tipo de escalonamento *First-come, first-served*.

Para sistemas homogêneos foi adotada a configuração de um computador **ASUS A7N8X Deluxe Motherboard, AMD Athlon (TM) XP 3000+1 da Advanced Micro Devices**, com aproximadamente 960 MHz e memória de 1024 Mbytes.

Já para os sistemas heterogêneos a capacidade computacional desses computadores variou aleatoriamente em função do número de computadores. Atualmente, o simulador possui 52 computadores disponíveis para simulação e apresentam em 80% dos casos capacidades computacionais e memórias diferentes um dos outros. No entanto, o usuário está livre para adicionar qualquer nova configuração que se faça necessário, pois existe uma ferramenta no simulador que auxilia na adição de novos computadores e, conseqüentemente, novas configurações.

A Tabela 5.12 apresenta os resultados dos tempos médios de resposta para simulações em ambientes homogêneos e heterogêneos. Pode-se notar que com o aumento do número de computadores no sistema existe uma melhora no tempo médio de resposta para ambos ambientes. Essa é uma das características importantes do TLBA que é recomendado para ambientes com um elevado número de computadores.

No entanto, como pode ser visto na Tabela 5.13 ele é mais recomendado para sistemas heterogêneos por apresentar em média um menor número de mensagens geradas para esse sistema, quando comparados aos ambientes homogêneos.

O aumento do número de computadores e a diminuição da variância entre os valores obtidos mostram que o TLBA pode ser aplicado em ambientes altamente escaláveis e sua estabilidade estará garantida por suas políticas de balanceamento de cargas.

**Tabela 5.12 –** Tempo médio de resposta (s), variância e desvio padrão para ambientes homogêneos e heterogêneos

| Número de Compu-tadores | AMBIENTES HOMOGÊNEOS | | | AMBIENTES HETEROGÊNEOS | | |
|---|---|---|---|---|---|---|
| | Tempo Médio de Resposta (s) | Variância (s) | Desvio Padrão (s) | Tempo Médio de Resposta (s) | Variância (s) | Desvio Padrão (s) |
| **10** | 7543,0240 | 67593419,3790 | 192,3460 | 7732,1970 | 78047973,6640 | 197,2000 |
| **100** | 2919,3320 | 3307481,9080 | 74,4320 | 3440,8230 | 7280377,9270 | 87,7730 |
| **500** | 2779,0980 | 2653977,8110 | 70,8310 | 3495,7530 | 4285556,3800 | 89,0850 |
| **1000** | 2778,0980 | 2853977,8110 | 71,1284 | 3538,2440 | 4372803,3540 | 90,1550 |
| **5000** | 2775,2670 | 2744443,7330 | 72,7634 | 3586,2680 | 4412478,3801 | 79,3134 |

**Tabela 5.13** – Número de Mensagens geradas para sistemas homogêneos e heterogêneos

| Número Compu-tadores | Mens. Por Nível | Mens. Comp. Sobrecarregado | Mens. de Atualização | Mens. de Balanceamento de Carga |
|---|---|---|---|---|
| **SISTEMAS HOMOGÊNEOS** | | | | |
| | 1263,7858 | 552,4231 | 4,6077 | 718,6480 |
| **10** | ± 3,5562 | ± 4,6959 | ± 2,0297 | 1,0085 |
| | 278,2246 | 664,0536 | 125,7956 | 1164,5745 |
| **100** | ± 5,3024 | ± 8,6095 | ± 50,9341 | 1,0101 |
| | 101,0533 | 452,5085 | 761,6238 | 1388,0000 |
| **500** | ± 2,4749 | ± 8,8451 | ± 102,7539 | ± 0 |
| | 63,7214 | 555,0435 | 761,7656 | 1388,0000 |
| **1000** | ± 1,2350 | ± 8,6745 | ± 102,8070 | ± 0 |
| | 55,3860 | 696,1333 | 7092,6333 | 1388,0000 |
| **5000** | ± 3,6349 | ± 24,4420 | ± 1294,8325 | ± 0 |
| **SISTEMAS HETEROGÊNEOS** | | | | |
| | 1056,4933 | 272,7787 | 2,9601 | 689,0010 |
| **10** | ± 2,4886 | ± 3,1296 | ± 1,5136 | ± 0,9433 |
| | 274,2629 | 635,1577 | 105,2963 | 1174,7492 |
| **100** | ± 3,0698 | ± 5,9061 | ± 29,1021 | ± 0,7657 |
| | 101,3695 | 455,2296 | 770,0518 | 1390,0000 |
| **500** | ± 2,5043 | ± 8,5351 | ± 104,0671 | ± 0 |
| | 63,8601 | 552,4294 | 773,9275 | 1390,0000 |
| **1000** | ± 1,2629 | ± 8,4351 | ± 105,1539 | ± 0 |
| | 64,3916 | 732,9667 | 7386,4000 | 1399,0000 |
| **5000** | ± 4,4141 | ± 22,0985 | ± 369,32 | ± 0 |

## 5.3 Considerações Gerais

Os estudos realizados nesse capítulo confirmam os resultados apresentados por Mello (2003) e fornecem novas informações sobre o número de mensagens geradas no sistema, número de migrações e tempos médio de resposta dos processos em função dos casos estudados.

Como base nesses estudos foi possível verificar que:

- o novo simulador foi validado através dos estudos do tempo médio de resposta dos processos, do número de mensagens geradas durante o balanceamento e do percentual utilizado do concentrador de rede (*switch*) e seus resultados foram muito próximos aos apresentados por Mello (2003), assegurando sua confiabilidade e funcionamento;
- o novo simulador comparado com o antigo oferece uma maior diversidade de recursos onde o usuário pode configurar os parâmetros de simulação mais facilmente, visualizar seus resultados após cada ciclo de simulação, gerar gráficos, salvar e abri-los em outros *softwares*;
- o novo simulador oferece bases estatísticas mais sólidas e foi desenvolvido envolvendo novos conceitos e estudos;
- sistemas com um baixo número de computadores e um elevado número de processos estará sobrecarregado em virtude do número de mensagens e terá um maior tempo médio de resposta para cada processo;
- sistemas com um elevado número de computadores e poucos níveis na árvore computacional, proporcionam um alto número de mensagens por nível. Neste caso, existe uma maior sobrecarga nas buscas em profundidade utilizadas para localizar os computadores ociosos e árvores com poucos níveis geram maior número de mensagens entre níveis e diminuem a escalabilidade do ambiente;
- sistemas bem dimensionados, geram um menor número de mensagens e uma menor sobrecarga no meio de comunicação, quando em fase de balanceamento de cargas e transferência de processos;

- a escalabilidade de um sistema estará assegurada à medida que se tem um elevado número de computadores, pois o número de mensagens por nível será menor quanto maior for o sistema. Além disso, o número de computadores por nível somente influenciará no número de mensagens por nível em sistemas de baixo porte, onde se terá um aumento do número de mensagens geradas por nível;
- um sistema apresenta um melhor desempenho se o número de processos for equivalente ao número de computadores, ou seja, se todos os computadores possuírem uma carga igualmente distribuída ou balanceada;
- a variação dos *Thresholds* de transferência e atualização, pouco contribui para a melhoria no desempenho com relação à diminuição do número de mensagens geradas e no tempo médio de resposta dos processos. O aumento no *Threshold* de transferência contribui apenas para evitar um número excessivo no número de migrações de processo e, conseqüentemente, evitar com que o sistema seja prejudicado por esse excesso de migrações.
- a limitação do número de migrações entre 3 e 5, é uma boa escolha para melhorar o tempo de execução dos processos e do desempenho do sistema quando se está utilizado o TLBA;
- o TLBA é altamente recomendado para sistemas de grande porte e heterogêneos uma vez que fornece um melhor tempo médio de resposta, um menor número de mensagens e a diminuição no valor da variância garante sua estabilidade.

Capítulo

6

# 6 CONCLUSÕES

## 6.1 Considerações Finais

Este trabalho teve como objetivo a elaboração de um novo simulador para o TLBA. Ele permite o estudo da variação de seus parâmetros de balanceamento de carga, aplicação em diversos ambientes homogêneos e heterogêneos, análise dos resultados obtidos e impactos no sistema devido às mudanças dos parâmetros de balanceamento de carga.

O processo de simulação é fundamental para a obtenção dos resultados, otimizando o tempo de execução de todo o processo e flexibilizando a variação dos parâmetros do projeto.

Uma vez que a simulação é extremamente útil para uma primeira visão do sistema foi implementado o simulador para análise do TLBA (TLBASim).

O TLBASim apresentou uma interface gráfica coerente com as atuais práticas de desenvolvimento, além de diversos recursos adicionais para a introdução de conceitos básicos na área de balanceamento de cargas e sistemas distribuídos, tornando-o uma ferramenta adequada para fins didáticos.

Seus recursos de tabela e gráficos facilitam a interpretação de simulações e análise de resultados. O recurso de diagramação da árvore lógica auxilia a análise do algoritmo e a verificação da montagem da árvore computacional em tempo real de simulação.

O TLBASim comparado com o antigo simulador fornece uma diversidade de recursos adicionais, além de estar fundamentado em bases estatísticas

mais sólidas e ter incorporado novas teorias e estudos apresentados na literatura. Graças a ele é possível o estudo de todos parâmetros do TLBA de acordo com suas políticas de balanceamento de carga.

Através das análises desses parâmetros, verificou-se que a alteração de alguns parâmetros não traz benefícios significativos ao sistema (alteração dos *Thresholds* e número de migrações), enquanto que em outros (número de níveis de computadores, número de computadores por nível, número de processos no sistema), influenciam diretamente no número de mensagens geradas e, conseqüentemente, no tempo de resposta final da execução dos processos no sistema.

Para os *Thresholds* foi possível verificar que eles pouco influenciam no tempo final de resposta dos processos e no número de mensagens geradas no sistema. Porém, o aumento do *Threshold* minimiza o número de migrações a serem realizadas e, em alguns casos, pode melhorar o desempenho do sistema.

Pode-se constatar que o número máximo de migrações permitidas deve estar entre 3 e 5 para se ter uma melhor tempo médio de resposta para os processos e um menor número de mensagens de balanceamento no sistema.

Os resultados confirmam a aplicação do TLBA para grandes sistemas homogêneos ou heterogêneos, uma vez que melhorou o tempo médio de resposta se comparado a sistemas sem migração.

Para sistemas com um pequeno número de computadores na árvore lógica e com poucos níveis apresentaram um elevado número de mensagens e um aumento no tempo médio de resposta quando da aplicação do TLBA.

Este estudo permitiu a análise do TLBA para um conjunto de parâmetros de balanceamento de carga e para sistemas homogêneos e heterogêneos e contribui de forma a priorizar os parâmetros a serem considerados para um melhor desempenho do TLBA para sistema distribuídos.

## 6.2 Trabalhos Futuros

Como sugestão de trabalhos futuros a serem realizados neste simulador pode-se citar os seguintes tópicos:

- implementação de outros algoritmos de balanceamento de carga e comparação com TLBA;
- aprimoramento e melhoria das funcionalidades deste simulador (impressão, múltiplas janelas etc);
- utilização das interfaces gráficas para implementação de um sistema real distribuído utilizando-se o TLBA em uma rede de computadores e comunicação através de *sockets* TCP/IP.
- implementação de novas abordagens e conceitos apresentados na literatura como, por exemplo, análise do tipo de aplicação: *CPU-Bound, I/O-Bound, Network-Bound e Memory-Bound*;
- criação de novas janelas com serviços de *Web browsers* permitindo o monitoramento *online* de sistemas pertencentes a outras redes onde o TLBA estaria sendo executando e fornecendo recursos suficientes para o usuário monitorar uma determinada rede de computadores.
- integração de novas técnicas de balanceamento de cargas que viessem a surgir e proporcionassem melhorias no algoritmo;

# REFERÊNCIAS

AIZA, T.S. (2003). *Avaliação do uso de statecharts estocásticos e queuing statecharts em um ambiente de simulação automático*. Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2003.

ANÁLISE das diferentes capacidades computacionais através de Benchmarks. (2004). Disponível em:<http://www.specs.org>. Acesso em: 5 maio 2004.

BANKS, J. (1998). Principles of Simulation. In: _____. *Handbook of simulation: principles, methodology, advances, applications, and practice*. New York: Willey; Engineering Management Press, p.3-30.

BANKS, J. et al (2001) *Discrete-event system simulation*. 3rd ed. Upper Saddle River: Prentice-Hall. (International Series).

BERGLUND, M. J. (1986). An introduction to the V-system. *IEEE Micro*, v.6, n.4, p.35–52, Aug.

BRANCO, K.R.L.J. (2002). *Índices de carga e desempenho em ambientes paralelos/distribuídos – modelagem e métricas*. 260p. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-18052005-163302/>. Acesso em: 9 ago. 2005.

BRENT, R. (2002). *Classes estatísticas*. Disponível em: <http://www.wiseowl.com/downloads/downloads.aspx>. Acesso em: 28 out. 2005.

BRUSCHI, S.M. (1997). *Extensão do ASiA para simulação de arquitetura de computadores.* 88p. Dissertação (Mestrado), Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 1997.

_____. (2002). *ASDA – um ambiente de simulação distribuída automático.* 227p. Tese (Doutorado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2002.

CARSON, J.S. (2002) Model verification e validation. In: WINTER SIMULATION CONFERENCE, 31, 2002, Piscataway.*Proceedings…* Piscataway: IEEE. p.52-18.

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

CENTRO DE COMPUTAÇÃO DA UNICAMP (2005). *Treinamento: introdução ao PVM.* Campinas: UNICAMP/CCUEC; MCT-FINEP. Apostila. Disponível em: <ftp://ftp.cenapad.unicamp.br/cursos/PVM/ apostila.ps.gz>, Acesso em: 1 ago. 2005.

CHANCHIO, K.; SUN, X.H (2001). *A protocol design of comunication state transfer for distributed computing.* New York: IEEE Computer Society Press, p.715-718.

CHENGJIANG, L.; SANLI, L. (1994). Strategy and Simulation of Adaptive RID for Distributed Dynamic Load Balancing in Parelle Systems. In: INTERNATIONAL SYMPOSIUM ON PARALLEL ARCHITECTURES, ALGORITHMS AND NETWORKS, 1994, New York. *Proceedings…* New York: IEEE Computer Society Press. p.406-412.

CHIANG, S.H.; MANSHARAMANI, R.K. e VERNON, M.K. (1994). Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In: SIGMETRICS CONFERENCE OF MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 1994, New York, p.33-44.

CODE PROJECT:the - free source code and tutorials (2005). Disponível em: <http://www.codeproject.com>. Acesso em: 29 oct. 2005.

CODEGURU:the – number one developer site (2005). Disponível em: <http://www.codeguru.com>. Acesso em: 18 oct. 2005.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. (1994). *Distributed systems concepts and design*, 2nd ed. Harlow: Addison-Wesley.

COUTO, C.L.S. (2000). *Um estudo comparativo de benchmarks paralelos*. 175p. Dissertação (Mestrado) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2000.

CROVELLA, M. E. (1994). *Performance prediction and tuning of parallel programs*, PhD Thesis - University of Rochester, Rochester, 1994.

DANDAMUDI S. et al (1996). *Hierarchical load sharing policies for distributed systems.* Ottawa: Carleton University. (Technical Report SCS –96–1).

DUBROVSKY, A.; FRIEDMAN, R.; SCHUSTER, A. (1996). *Load balancing in a distributed shared memory system*. Disponível em: <http://www.cs.technion.ac.il/~assaf/publications/alex.doc>. Acesso em: 21 Apr. 2004.

DUNCAN, R. (1990). A survey of parallel computer archictectures. *IEEE Computer*, New York, v.23, n.2, p.5-16, Feb.

EAGER, D.L.; LAZOWSKA, E.D.; ZAHORJAN, J. (1986a). A comparison of receiver-initiated and sender-initiated adaptative load sharing. *Performance Evaluation*, Amsterdam, v.6, n.1, p.53-68, Mar.

_____. (1986b). Dynamic load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, v. SE-12, n.5, p.662-675, May.

EVANS, D.J. et al (1993). Dynamic load balancing using task-transfer probabilities. *Parallel Computing*, New York, v.19, n.8, p.897-916, Aug.

FEITELSON, D.G. (1996). *Packing schemes for gang scheduling*. Disponível em: <http://www.cs.huji.ac.il/labs/parallel/workload/models.html#_feitelson96>. Acesso em: 20 mar. 2006.

FEITELSON, D.G. (2001). Metrics for parallel job scheduling and their convergence. In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 7, 2001, Cambridge, *Proceedings…* Berlin: Springer. p.188-205. (Lectures notes in computer science, 2221).

_____. (2006). *Parallel workloads archive Feitelson '96*. Disponível em: <http://www.cs.huji.ac.il/labs/parallel/workload/models.html>. Acesso em: 20 mar. 2006.

FEITELSON, D.G.; JETTE, M.A. (1997) Improved utilization and responsiveness with gang scheduling. In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 1997, Geneva. *Proceedings…* Berlin: Springer. p.238-261. (Lectures notes in computer science, 1291).

FEITELSON, D.G.; RUDOLPH, L. (1995). Parallel job scheduling: issues and approaches. In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 1995, Santa Barbara. *Proceedings…* Berlin: Springer. p.1-18. (Lectures notes in computer science, 949).

_____. (1996). Toward convergence in job schedulers for parallel supercomputers, In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 1996, Honolulu. *Proceedings…* Berlin: Springer. p.1-26. (Lectures notes in computer science, 1162).

_____. (1998). Metrics and benchmarking for parellel job scheduling. In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 1998, Orlando. *Proceedings…* Berlin: Springer. p.1-24. (Lectures notes in computer science, 1459).

FEITELSON, D.G.; WEIL, A. M. (1998). Utilization and predictability in scheduling the ibm sp2 with backlling. INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM, 12., 1998, Orlando. *Proceedings…* Berlin: Springer. p.542-546. (Lectures notes in computer science, 1388).

FLYNN, M.J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, New York, v.C-21, n. 9, p.948-960, Sept.

FLYNN, M.J.; RUDD, K.W. (1996). Parallel Architectures. *ACM Computing Surveys*, v.28, n.1, p. 67-70, Mar.

FOX, G.C. e FURMANSKI, W. (1997). *Java for parallel cComputing and as a general language for scientific and engineering simulation and modelling*. New York: Syracuse University.

FRANCÊS, C.R.L. (2001a). *Statecharts estocásticos e queuing statecharts: novas abordagens para avaliação de desempenho baseadas em especificação statecharts*. 156p. Tese (Doutorado) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2001.

FRANCÊS, C.R.L. et al (2001b). Statecharts estocásticos e queuing statecharts: novas abordagens para avaliação de desempenho baseadas em especificação statecharts. . In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE *SOFTWARE*, 15., 2001, Rio de Janeiro. *Anais...* Rio de Janeiro: UFRJ. p.316–331.

GHARE, G.; LEUTENEGGER, S. T. (1999). The effect of correlating quantum allocation and job size for gang scheduling. In: JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, 1999, San Juan. *Proceedings...* Berlin: Springer. p.91-110. (Lectures notes in computer science, 1659).

GNU GENERAL PUBLIC LICENSE (2004). Disponível em: <http://www.gnu.org/licenses/gpl.html>. Acesso em: 8 feb. 2004

GUIMARÃES, C.C. (1980). *Princípios de sistemas operacionais*. Rio de Janeiro: Campus. p.195-210.

HALCHOR-BALTER, M.; DOWNEY, A. (1997). Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, v.15, n.3, p.256-285, Aug.

HU, L.; GORTON, I. (1997). *Performance evaluation for parallel systems: a survey*. Sydney: Department of Computer Systems, School of Computer Science and Engineering, University of NSW. (Technical Report UNSW-CSE-TR-9797).

HWANG, K.; XU, Z. (1998). *Scalable parallel computing – technology, architecture, programming*. Boston: WCB/McGrawHill.

ICARDI, D. (2005). *Projeto sourcegrid para apresentação de dados em tabela.* Disponível em: <www.codeproject.com>. Acesso em: 12 sept. 2005.

JAIN, R. (1991). *The art of computer systems performance analysis:* techniques for experimental design, measurement, simulation, and modeling. New York: John Wiley.

JAVA Technology (2004). Disponível em: <http://java.sun.com>. Acesso em: 9 oct. 2004.

JOHNSON, J.T. (2005). *TSWizard – a wizard framework for .NET*. Disponível em: <http://www.codeproject.com/cs/miscctrl/tswizard.asp>. Acesso em: 28 oct. 2005.

KRUEGER, P.; LAI, T.H. e RADIYA, V.A. (1991). Processor allocation vs. job scheduling on hypercube computers. In: 11th INTERNATIONAL CONFERENCE OF DISTRIBUTED COMPUTING SYSTEMS, 1991, *Proceedings…*, IEEE. p.394-401.

KUCHEN, H.; WAGENER, A. (1990). *Comparison of dynamic load balancing strategies*. Disponível em: <http://fiachra.ucd.ie/~david/loadbalancingpapers.html>, Acesso em: 5 sept. 2005.

KUNZ, T. (1991). The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, New York, v. 17, n.7, p.725-734, July.

LAINE, J.M. (2003). *Desempenho de Modelos para Predição de Desempenho de Programas Paralelos MPI*. 100p. Dissertação (Mestrado em Engenharia) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2001.

LEUTENEGGER, S.T. e VERNON, M.K. (1990). The performance of multiprogrammed multiprocessor scheduling policies. In: SIGMETRICS CONF. MEASUREMENT & MODELING OF COMPUT. SYST., 1990, p.226-236.

LI, K. C. (2001). *Análise e Predição de Desempenho de Programas Paralelos em Redes de Estações de Trabalho*. 113p. Tese (Doutorado) – Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica, Universidade de São Paulo, São Paulo, 2001.

LIVNY, M.; MELMANN, M. (1982). Load balancing in homogeneous broadcast distributed systems. *ACM Sigmetrics Performance Evaluation*, New York, v.11, n.1, p.47-55, Spring.

LO, V.; MACHE, J.; WINDISCH, K. (1998). A comparative study of real workload traces and synthetic workload models for parallel job scheduling. In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 1998, Orlando. *Proceedings…* Berlin: Springer. p.25-46. (Lectures notes in computer science, 1459).

LOH, P.K.K. et al (1996). How network topology affects dynamic load balancing. *IEEE Parallel & Distributed Technology*, v.4, n.3, p.25-35, Fall.

LUO, W. (2004). *Projeto utilização para elaboração do Ambiente de Trabalho e Gerenciamento das Telas Gráficas*. Disponível em: <http://www.myxaml.com/wiki/>. Acesso em: 15 sept. 2005.

LÜLING, R.; MONIEN, B. (1993). A dynamic distributed load balancing algorithm with provable good performance. In: ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTETURES, 10., 1993, Velen. *Proceedings…* New York: ACM. p.164-172

LÜLING, R.; MONIEN, B.; RAMME, F. (1993). *Load balancing in large networks: a comparative study*. Alemanha: Universidade de Paderborn. (Relatório técnico).

MACDOUGALL, M.H. (1987). *Simulating computer systems techniques and tools*. Cambridge: MIT Press.

MACIEL, P.R.M.; LINS, R.D.; CUNHA, P.R.F. (1996). Introdução às Redes de Petri e Aplicações. In: ESCOLA DE COMPUTAÇÃO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 10., 1996, Campinas. *Introdução...* Campinas: UNICAMP.

MAJUMDAR, S.; EAGER, D.L. e BUNT, R. B. (1988). *Scheduling in multiprogrammed parallel systems*. IN: SIGMETRICS CONF. MEASUREMENT & MODELING OF COMPUT. SYST., 1988, *Proceedings...* p. 104-113.

MARTINS, A.P.; COENTRO, T.R.L. (2005). *Comparação de multiprocessadores de memória partilhada e de transferência de mensagens*. Disponível em: <http://mega.rnl.ist.utl.pt/~ic-micro/uP_01_02_artigo_sintese_3.pdf>. Acesso em: 9 ago. 2005.

MEIRA JR., W. (1995). *Modeling Performance of Parallel Programs. Rochester*, New York: Computer Science Department, The University of Rochester. (Technical Report 589).

MELAB, N.; DEVESA, N.; LECOUFFE, M.P.; TOURSEL, B. (1996). Adaptive Load Balancing of Irregular Applications. In: INTERNATIONAL WORKSHOP ON PARALLEL ALGORITHMS FOR IRREGULARY STRUCTURED PROBLEMS, 3., 1996, Santa Barbara. *Proceedings...* Berlin: Springer. p. 337-338. (Lecture notes in computer science, 1117).

MELLO, R. F. (2003). *Proposta e Avaliação de Desempenho de um Algoritmo de Balanceamento de Carga para Ambientes Distribuídos Heterogêneos Escaláveis*. 130p. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

_____. (2005). *Website do Instituto de Matemática e Ciências da Computação*, Universidade de São Paulo. Disponível em: <http://www.icmc.usp.br/~mello/proj.html>. Acesso em: 4 abr. 2005.

MELLO, R.F.; SENGER, L.J. (2004). A new migration model based on the evaluation of processes load and lifetime on heterogeneous. *Computing Environments*. Submitted to SBAC-PAD 2004. (No prelo)

MELLO, R.F.; TREVELIN, L.C.; PAIVA, M.S. (2003a). Comparative Analysis of the Prototype and the Simulator of a New Load Balancing Algorithm for Heterogeneous Computing Environments. Aceito para publicação no International Journal of High Performance Computing and Network. (No prelo)

_____. (2003b). Comparative Study of the Server-Initiated Lowest Algorithm using a Load Balancing Index Based on the Process Behavior for

Heterogeneous Environment. Aceito para publicação no Journal of Networks, Software Tools and Application. (No prelo).

MORCILLO, A.M. (2004). *Amostras e populações*. Disponível em: <http://www.fcm.unicamp.br/centros/ciped/mp639/amostras%20e%20populacoes.pdf>. Acesso em: 11 out. 2004.

MU´ALEM, A.W.; FEITELSON, D.G. (2001). Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backlling. *IEEE Transactions on Parallel and Distributed Systems*, v.12, n.6, p.529.543, Feb.

MÜLLENDER, S.J (1993). *Distributed Systems*. Addison-Wesley. 2nded. New York: Addison-Wesley.

NAIK, V.K. (1985). *Performance of NAS paralle benchmark LU on IBM SP systems*. New York: IBM. (*IBM Research Report*, RC 20046).

NETRON PROJECT (2005). *Diagramação da Árvore Computacional*. Disponível em: <http://netron.sourceforge.net/wp/> ou <http://www.codeproject.com/cs/miscctrl/TreeDiagrams.asp>. Acesso em: 28 out. 2005.

OED, W.; MERTENS, B. (1981). Characterization of Computer System Workload. *Computer Performance*, v.2, n.2, p.77-83, June.

OLIVEIRA, R.S.; CARISSIMI, A.S. e TOSCANI, S.S. (2001). *Sistemas operacionais*. 2.ed. Porto Alegre: Sagra-Luzzato. Disponível em:<http://www.inf.ufrgs.br/~asc/livro/secao94.pdf>. Acesso em: 3 Sept. 2005.

_____. (2005). *Organização de sistemas operacionais convencionais e de tempo real.* Cap.8. Disponível em: <http://www.lcmi.ufsc.br/~romulo/discipli/cad-cbtisa/SO-Jai2002-pagina.pdf>. Acesso em: 9 ago. 2005.

OMÊNA, M. (2004). *Clusters e supercomputação*. Disponível em: <http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=1362>. Acesso em: 15 out. 2005.

PATTERSON, D.A.; HENNESSY, J.L. (2000). *Organização e Projeto de Computadores*. 2.ed. Rio de Janeiro: LTC

PEGDEN, C.D.; SHANNON, R.E.; SADOWSKI, R.P. (1995). *Introduction to Simulation using SIMAN*. 2nd ed. New York: MacGraw-Hill International.

PETRI, C.A. (1962). *Kommunikation mit automaten*. Bonn: Institut für Instrumentelle Mathematik. (Technical Report RADC-TR-65-3770.

PFISTER, G. F. (1998). *In search of clusters*. Upper Saddle River: Prentice-Hall.

PITANGA, M. (2003). *Computação em cluster: o estado da arte da computação*. Rio de Janeiro: Brasport.

SANTANA, M.J. (1990) *An advanced filestore architecture for a multiple lan distributed computing system*. PhD Thesis - University of Southampton, Southampton, 1990.

SANTANA, R.H.C. (1990). *Perfomance evaluation of lan-based file servers*. PhD Thesis - University of Southamptom, Southampton, 1990.

SANTANA, R.H.C. et al. (1994). *Técnicas para avaliação de desempenho de sistemas computacionais*, São Carlos: ICMC/USP. (Notas didáticas, 14).

SCOFIELD, W.C.L. (2002). *Uma implementação híbrida e distribuída do problema job-shop através dos algoritmos GRASP e genético*. Monografia apresentada ao Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto, Ouro Preto, 2002.

SHAY, W.A., *Sistemas operacionais.* São Paulo: Makron Books. p.238-302.

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

SHIRAZI, B.A.; HURSON, A.R.; KAVI, K.M (1995). *Scheduling and load balancing in parallel and distributed systems.* Los Alamitos: IEEE Computer Society.

SHIVARATRI, N.G., KRUEGER, P.; SINGHAL, M. (1992). Load distributing for locally distributed systems. *IEEE Computer*, New York, v.25, n.12, p.33-44, Dec.

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

SKILLICORN, D.B.; TALIA, D. (1998) Models and Languages for Parallel Computation. *ACM Computing Surveys*, New York, v.30, n.2, p.123-169, June.

SOARES, L.F.G. (1992). *Modelagem e Simulação Discreta de Sistemas.* São Paulo: Campus.

SONG, J.; CHOO, H.K.; LEE, K.M. (1997). Application-Level Load Migration and its Implementation on top of PVM. *Concurrency: practice and experience*, v.9, n.1, p.1-19, Jan.

SOURCEFORGE.NET (2005). Disponível em: <http://sourceforge.net>. Acesso em: 18 oct. 2005.

SOUSA, P.B.M. (2001). *WVM:* uma ferramenta para processamento distribuído na Web. 197p. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal do Ceará, Fortaleza. Disponível em: <http://www.mcc.ufc.br/disser/PauloBenicio.pdf>, Acesso em: 31 ago. 2005.

SPOLON, R. (2001). *Um método para a avaliação de desempenho de protocolos de sincronização otimistas para simulação distribuída*. 201p. Tese (Doutorado) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2001.

SUN MICROSYSTEMS (2004). Disponível em: <http://www.sun.com>. Acesso em: 9 oct. 2004.

TALBY, D.; FEITELSON, D. G.; RAVEH, A. (1999). Comparing logs and models using the co-plot method. In: FEITELSON, D.G.; RUDOLPH, L. (ED.). *For parallel processing*. New York: Springer Verlag. p.43-66. (Lecture note in computer science, 1659).

TANENBAUM, A.S. (1992). *Operating systems* – design and implementation. Englewood Cliffs: Prentice-Hall. p.80-87.

_____. (1996). *Computer Networks.* 3thed. Englewood Cliffs: Prentice-Hall.

THEIMER, M.M.; LANTZ, K.A. (1989). Finding Idle Machines in a Workstation-Based Distributed System. *IEEE Transactions on Software Engineering*, v.15, n.11, p.1444-1458, Nov.

VAN DER STEEN, A.J.; DONGARRA, J.J. (1996). *Overview of recent supercomputer*. Tennessee: Department of Computer Science, University of Tennessee. (Technical Report UT-CS-96-235).

XU, C.; LAU, F.C.M. (1997). *Load balancing in parallel computers:* theory and practice. Boston: Kluwer Academic.

XU, J.; HWANG, K. (1993). Heuristics methods for dynamic load balancing in a message-passing multicomputer. *Journal of Parallel and Distributed Computing*, San Diego, v.18, n..1, p.1-13, May.

WEICKER, R.P. (1990). An overview of common benchmarks. *IEEE Computers*, New York, v.23, n.12, p.65-75, Dec.

WILLEBEEK-LEMAIR, M.H.; REEVES, A.P. (1993). Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, New York, v.4, n.9, p.979-993, Sept.

WU, M. (1997). On runtime parallel scheduling for processor load balancing. *IEEE Transactions on Parallel and Distributed Systems*, New York, v.8, n.2, p.173-186, Feb.

YAMIN, A.C. (1994). *Um ambiente para a exploração de paralelismo na programação em lógica*. 212p. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 1994.

YAN, Y.; ZHANG, X. (1996). An effective and practical model for parallel computing on non-dedicated heterogeneous NOW. *Journal of Parallel and Distributed Computing*, San Diego, v.38, n.1, p.63-80, Oct.

YOKOKURA, A.Y. (2005). *Estudo de viabilidade da implantação de técnicas de cluster ao projeto servidor de estações de trabalho (SET)*. Trabalho Conclusão de Curso (Bacharel em Ciência da Computação) - Universidade Federal do Pará, Belém, 2005. Disponível em:<http://www.cultura.ufpa.br/informatica/TCCs/tcc2005/alex_yokokura.pdf>. Acesso em: 9 ago. 2005.

YOURDON, E. (1990). *Análise estruturada moderna*. Rio de Janeiro: Campus.

ZANDY, V.C. (2004). *CKPT – A process checkpoint library*. Disponível em: <http://www.cs.wisc.edu/~zandy/ckpt>. Acesso em: 17 apr. 2004.

ZEDGRAPH (2005). *A flexible charting library for .NET*. Disponível em: <http://www.codeproject.com/csharp/ZedGraph.asp>. Acesso em: 28 oct. 2005.

ZHOU, S.; FERRARI, D. (1987). *An experimental study of load balancing performance*. Berkeley: Universidade da California/Computer Science Division (EECS). (Report No. UCB/CSD 87/336, PROGRES Report No. 86.8).