Rodrigo Fernandes de Mello

PROPOSTA E AVALIAÇÃO DE DESEMPENHO DE UM ALGORITMO DE BALANCEAMENTO DE CARGA PARA AMBIENTES DISTRIBUÍDOS HETEROGÊNEOS ESCALÁVEIS

Tese apresentada à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte dos requisitos para a obtenção do título de Doutor em Engenharia Elétrica.

ORIENTADORA: Prof. Dra. Maria Stela Veludo de Paiva

São Carlos

2003

AGRADECIMENTOS

Agradeço inicialmente a meu pai e minha mãe internos e eternos.

Agradeço a orientação despendida pela Profa. Dra. Maria Stela Veludo de Paiva.

Agradeço, principalmente, ao meu co-orientador, Prof. Dr. Luis Carlos Trevelin, pois sem seu apoio e colaboração este trabalho não poderia jamais ser realizado.

Agradeço ao Prof. Dr. Adilson Gonzaga pela colaboração na realização deste trabalho.

Agradeço ao Prof. Dr. Gonzalo Travieso pela colaboração nas revisões de artigos resultantes deste projeto.

Agradeço a toda minha família pela colaboração para elaborar este trabalho, principalmente à minha querida esposa Claudia.

Agradeço, também, a todos meus amigos que direta ou indiretamente colaboraram para a realização deste trabalho, em especial a Leonardo Marcus da Silva e Luiz Falaguasta Barbosa.

"Toda teoria é cinza,

só é verde a árvore de dourados frutos,

que é a vida"

Goethe

RESUMO

MELLO, R. F. (2003). Proposta e Avaliação de Desempenho de um Algoritmo de Balanceamento de Carga para Ambientes Distribuídos Heterogêneos Escaláveis. Tese (Doutorado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

Algoritmos de balanceamento de carga são utilizados em sistemas distribuídos para homogeneizar a ocupação dos recursos computacionais disponíveis. A homogeneidade na ocupação do ambiente permite otimizar a alocação de recursos e, conseqüentemente, aumentar o desempenho das aplicações. Com o advento dos sistemas distribuídos de alta escala, fazem-se necessárias pesquisas para a construção de algoritmos de balanceamento de carga que sejam capazes de gerir com eficiência esses sistemas. Essa eficiência é medida através do número de mensagens geradas no ambiente, do suporte a ambientes heterogêneos, do uso de políticas que consomem poucos recursos do sistema, da estabilidade em alta carga, da escalabilidade do sistema e dos baixos tempos de resposta. Com o objetivo de atender as necessidades dos sistemas distribuídos de alta escala, este doutorado propõe, apresenta e avalia um novo algoritmo de balanceamento de carga denominado TLBA (*Tree Load Balancing Algorithm*). Esse algoritmo organiza os computadores do sistema em uma topologia lógica na forma de árvore, sobre a qual são executadas operações de balanceamento de carga. Para validar o TLBA foi construído um simulador que, submetido a testes, permitiu comprovar suas contribuições, que incluem: o baixo número de mensagens geradas pelas operações de balanceamento de carga; a estabilidade em altas cargas; os baixos tempos médios de resposta de processos. Para validar os resultados de simulação, foi construído um protótipo do TLBA. Esse protótipo confirmou os resultados de simulação e, conseqüentemente, as contribuições do algoritmo.

Palavras-chave: balanceamento de carga, computação paralela e distribuída, alto desempenho.

ABSTRACT

MELLO, R. F. (2003). Proposal and Performance Evaluation of a Load Balancing Algorithm for Heterogeneous Scalable Distributed Environments. Ph.D. Thesis – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

Load balancing algorithms are applied in distributed systems to homogenize the occupation of the available computational resources. The homogeneity of the environment occupation allows optimising the resource allocation and consequently, increasing the application performance. With the advent of the large-scale distributed systems, it was necessary to start researching the construction of load balancing algorithms which are able to manage these systems with efficiency. This efficiency is measured through the number of messages generated on the environment; the support to heterogeneous environments and the load balance policies which should spend the minimal resources time; the stability in overloaded situations; the system scalability; and the processes average response times, that should be small. With the aim to achieve the large-scale distributed systems requirements, this Ph.D. proposes, presents and evaluates a new load balancing algorithm named TLBA (*Tree Load Balancing Algorithm*). This algorithm arranges the computers on a logical network topology with a tree format. The load balancing operations are executed over this tree. To evaluate the TLBA algorithm, a simulator was built that was submitted to tests that confirmed the following caracteristics: the small number of messages generated by the load balancing operations; the stability in overloaded situations; the small average processes response times. To validate the simulation results a TLBA prototype was implemented. This prototype confirmed the simulation results and consequently the contributions of the proposed algorithm.

Keywords: load balancing, parallel and distributed computing, high performance.

LISTA DE FIGURAS

FIGURA 1 – EXEMPLO DE *SLIDING-WINDOW* DO ALGORITMO GENÉTICO 24
FIGURA 2 – EXEMPLO DE TOPOLOGIA FÍSICA DE UMA REDE 67
FIGURA 3 - TEMPO MÉDIO DE RESPOSTA DOS PROCESSOS EM FUNÇÃO DO TAMANHO DO SUBCONJUNTO DE COMPUTADORES ANALISADOS PELA POLÍTICA DE LOCALIZAÇÃO DO ALGORITMO LOWEST 79
FIGURA 4 - NÚMERO TOTAL DE MENSAGENS GERADAS NO MEIO DE COMUNICAÇÃO, PARA DIFERENTES TAMANHOS DE SUBCONJUNTO DE COMPUTADORES 80
FIGURA 5 – VARIAÇÃO NOS TEMPOS DE RESPOSTA DE 100 SIMULAÇÕES SUCESSIVAS DO ALGORITMO TLBA E LOWEST MODIFICADO, EM AMBIENTE COMPOSTO POR 7 COMPUTADORES HOMOGÊNEOS 83
FIGURA 6 – VARIAÇÃO NOS TEMPOS DE RESPOSTA DE 100 SIMULAÇÕES SUCESSIVAS DO ALGORITMO TLBA E LOWEST MODIFICADO, EM AMBIENTE COMPOSTO POR 7 COMPUTADORES HETEROGÊNEOS 87
FIGURA 7 – TEMPO MÉDIO DE RESPOSTA DOS PROCESSOS EM FUNÇÃO DO TAMANHO DO SUBCONJUNTO DE COMPUTADORES ANALISADOS PELA POLÍTICA DE LOCALIZAÇÃO DO ALGORITMO LOWEST 89
FIGURA 8 – NÚMERO DE MENSAGENS GERADAS NO AMBIENTE PARA DIFERENTES TAMANHOS DE SUBCONJUNTO DE COMPUTADORES EM UM AMBIENTE COMPOSTO POR 781 COMPUTADORES HOMOGÊNEOS 90
FIGURA 9 – TEMPOS DE RESPOSTA OBTIDOS EM 100 SIMULAÇÕES SUCESSIVAS 93
FIGURA 10 – NÚMERO DE MENSAGENS TROCADAS NAS OPERAÇÕES DE BALANCEAMENTO DE CARGA 97
FIGURA 11 – NÚMERO DE MENSAGENS TROCADAS NAS OPERAÇÕES DE BALANCEAMENTO DE CARGA, UTILIZANDO O ALGORITMO TLBA E LOWEST MODIFICADO 99

LISTA DE TABELAS

TABELA 1 – OCUPAÇÃO DOS PROCESSADORES EM UM AMBIENTE QUE UTILIZA UM ALGORITMO GENÉTICO PARA O BALANCEAMENTO DE CARGA 24
TABELA 2 – SIMULAÇÕES PARA DEFINIR O NÚMERO MAIS INDICADO DE COMPUTADORES ANALISADOS PELA POLÍTICA DE LOCALIZAÇÃO DO ALGORITMO LOWEST 78
TABELA 3 – TEMPOS MÉDIOS DE RESPOSTA DO ALGORITMO TLBA E LOWEST MODIFICADO EM AMBIENTES COMPOSTOS POR 7 COMPUTADORES DE CAPACIDADE DE PROCESSAMENTO HOMOGÊNEA 81
TABELA 4 - MÉDIAS, MEDIANAS E DESVIOS PADRÃO OBTIDOS COMO RESULTADO DAS SIMULAÇÕES DE PRECISÃO NO TEMPO DE RESPOSTA DE PROCESSOS, REALIZADAS SOBRE OS ALGORITMOS TLBA E LOWEST MODIFICADO 82
TABELA 5 – TEMPOS MÉDIOS DE RESPOSTA DO ALGORITMO TLBA E LOWEST MODIFICADO EM AMBIENTES COMPOSTOS POR 7 COMPUTADORES DE CAPACIDADE DE PROCESSAMENTO HETEROGÊNEA 85
TABELA 6 – MÉDIAS, MEDIANAS E DESVIOS PADRÃO OBTIDOS COMO RESULTADO DAS SIMULAÇÕES DE PRECISÃO NO TEMPO DE RESPOSTA DE PROCESSOS, REALIZADAS SOBRE OS ALGORITMOS TLBA E LOWEST MODIFICADO 86
TABELA 7 – RESUMO DOS RESULTADOS OBTIDOS NAS SIMULAÇÕES PARA DEFINIR O NÚMERO MAIS INDICADO DE COMPUTADORES ANALISADOS PELA POLÍTICA DE LOCALIZAÇÃO DO ALGORITMO LOWEST 90
TABELA 8 – TEMPOS MÉDIOS DE RESPOSTA DO ALGORITMO TLBA E LOWEST MODIFICADO EM AMBIENTES COMPOSTOS POR 781 COMPUTADORES DE CAPACIDADE DE PROCESSAMENTO HOMOGÊNEA 91
TABELA 9 – MÉDIAS, MEDIANAS E DESVIOS PADRÃO OBTIDOS COMO RESULTADO DAS SIMULAÇÕES DE PRECISÃO NO TEMPO DE RESPOSTA DOS PROCESSOS, REALIZADAS SOBRE OS ALGORITMOS TLBA E LOWEST MODIFICADO 92
TABELA 10 – TEMPOS MÉDIOS DE RESPOSTA DO ALGORITMO TLBA E LOWEST MODIFICADO EM AMBIENTES COMPOSTOS POR 781 COMPUTADORES DE CAPACIDADE DE PROCESSAMENTO HETEROGÊNEA 95
TABELA 11 – MÉDIAS, MEDIANAS E DESVIOS PADRÃO OBTIDOS COMO RESULTADO DAS SIMULAÇÕES DE PRECISÃO NO TEMPO DE RESPOSTA DOS PROCESSOS, REALIZADAS SOBRE OS ALGORITMOS TLBA E LOWEST MODIFICADO 96

TABELA 12 – TEMPOS MÉDIOS DE RESPOSTA OBTIDOS COMO RESULTADOS DAS EXECUÇÕES DO PROTÓTIPO DO TLBA E DO SIMULADOR EM UM AMBIENTE HOMOGÊNEO 107
TABELA 13 – NÚMERO DE MENSAGENS GERADAS NA REDE DE COMPUTADORES 108
TABELA 14 - TEMPOS MÉDIOS DE RESPOSTA OBTIDOS COMO RESULTADOS DAS EXECUÇÕES DO PROTÓTIPO DO TLBA E DO SIMULADOR EM UM AMBIENTE HETEROGÊNEO 109
TABELA 15 – MENSAGENS TRAFEGANDO NA REDE PARA O AMBIENTE REAL E SIMULADO 109
TABELA 16 – SISTEMA OPERACIONAL 123
TABELA 17 – CARACTERÍSTICAS DO PROCESSADOR 124
TABELA 18 – CARACTERÍSTICAS DE MEMÓRIA 124
TABELA 19 – DEMAIS CARACTERÍSTICAS DE HARDWARE 124
TABELA 20 – SISTEMA OPERACIONAL 125
TABELA 21 – CARACTERÍSTICAS DO PROCESSADOR 126
TABELA 22 – CARACTERÍSTICAS DE MEMÓRIA 126
TABELA 23 – DEMAIS CARACTERÍSTICAS DE HARDWARE 126

SUMÁRIO

RESUMO 1
ABSTRACT 2
1. INTRODUÇÃO 11
2. BALANCEAMENTO DE CARGA 14
2.1 Considerações Iniciais 14
2.2 Taxonomia de Algoritmos de Escalonamento de Processos 14
2.2.1 Classificação Hierárquica 14
2.2.2 Classificação Plana 16
2.3 Balanceamento de Carga 17
2.4 Resumo do Capítulo 18
3. ESTADO DA ARTE NO BALANCEAMENTO DE CARGA 19
3.1 Balanceamento de Carga Hidrodinâmico 19
3.2 Análise de Modelos de Balanceamento de Carga Usando Teoria de Filas 20
3.3 Modelo de Custo-Oportunidade para Alocação de um Processo 21
3.4 Modelo de Análise da Queda de Desempenho para Aplicações que Executam em Clusters de Workstations de Tempo Compartilhado 22
3.5 Estudos de Balanceamento de Carga Dinâmico Usando Algoritmos Genéticos 23
3.6 Impacto da Carga de Trabalho e dos Parâmetros de Sistema nos Mecanismos de Escalonamento de Processos em Clusters 24
3.7 Balanceamento de Carga Randômico Utilizando Duas Opções 25
3.8 Heurísticas de Compartilhamento Dinâmico de Carga Baseada em Predição de Recursos por Processo 26
3.9 Estratégias para o Balanceamento de Carga em Computadores Altamente Paralelos 27
3.10 Conjuntos de Envio e Recebimento de Informações para Ambientes Multiprocessados 28
3.11 Um Estudo Experimental de Desempenho em Algoritmos de Balanceamento de Carga 28
3.12 Algoritmos Microeconômicos para o Balanceamento de Carga em Sistemas Computacionais Distribuídos 30

3.13 Busca por Computadores Ociosos em Sistemas Distribuídos Baseados em Workstations 31
3.14 Distribuição de Carga em Sistemas Distribuídos sobre Redes Locais 32
3.15 Predição do Uso de Recursos por Processo: Um Estudo de Medidas sobre UNIX 34
3.16 Escalonamento Paralelo em Tempo de Execução para Balanceamento de Carga em Processadores 35
3.17 Outros Estudos Relacionados 36
3.18 Análise dos Estudos 36
3.19 Resumo do Capítulo 39
4. PRINCIPAIS SISTEMAS DE BALANCEAMENTO DE CARGA 40
4.1 Considerações Iniciais 40
4.2 V-System 40
4.3 Sprite 41
4.4 Condor 42
4.5 Stealth 43
4.6 PBS 43
4.7 Messiahs 44
4.8 Mosix 45
4.9 Resumo do Capítulo 45
5. APLICAÇÃO DE ALGORITMOS DE BALANCEAMENTO DE CARGA EM SISTEMAS DE COMPUTAÇÃO PARALELA DISTRIBUÍDA 46
5.1 Considerações Iniciais 46
5.2 Clusters 46
5.3 Grid Computing 47
5.4 Sistemas Distribuídos baseados em Redes de Estações 48
5.5 Resumo do Capítulo 48
6. TÉCNICAS DE AVALIAÇÃO DE DESEMPENHO DE SISTEMAS COMPUTACIONAIS 49
6.1 Considerações Iniciais 49
6.2 Técnicas de Avaliação de Desempenho 49
6.3 Resumo do Capítulo 50

7. PROPOSTA DE UM ALGORITMO DE BALANCEAMENTO DE CARGA PARA AMBIENTES DISTRIBUÍDOS HETEROGÊNEOS ESCALÁVEIS 52
7.1 Considerações Iniciais 52
7.2 Objetivos 54
7.3 Modelo Formal do TLBA 55
7.3.1 Política de Informação 56
7.3.2 Política de Localização 61
7.3.3 Política de Transferência 62
7.3.4 Política de Seleção 64
7.3.5 Outras Considerações sobre o Algoritmo TLBA 65
7.4 Contribuições 68
7.5 Resumo do Capítulo 68
8. CONSTRUÇÃO DE UM EMULADOR DO TLBA 70
8.1 Considerações Iniciais 70
8.2 Objetivos do Emulador do TLBA 70
8.3 Implementação do Emulador 70
8.4 Resumo do Capítulo 72
9. CONSTRUÇÃO E AVALIAÇÃO DOS RESULTADOS OBTIDOS DE UM SIMULADOR DO ALGORITMO TLBA 73
9.1 Considerações Iniciais 73
9.2 Objetivos da Simulação 73
9.3 Métricas de Análise 74
9.4 Considerações sobre os Simuladores do Algoritmo TLBA e Lowest Modificado 75
9.5 Propostas de Avaliação usando Simuladores 75
9.6 Simulações para Analisar o Desempenho do TLBA em um Pequeno Sistema Homogêneo 76
9.6.1 Política de Localização do Algoritmo Lowest Modificado 76
9.6.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado 80
9.6.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado 82

9.7 Simulações para analisar o desempenho do TLBA em um pequeno sistema heterogêneo 83
9.7.1 Política de Localização do Algoritmo Lowest Modificado 84
9.7.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado 84
9.7.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado 85
9.8 Simulações para analisar o desempenho do TLBA em um grande sistema homogêneo 87
9.8.1 Política de Localização do Algoritmo Lowest Modificado 87
9.8.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado 91
9.8.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado 92
9.9 Simulações para analisar o desempenho do TLBA em um grande sistema heterogêneo 93
9.9.1 Política de Localização do Algoritmo Lowest Modificado 93
9.9.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado 94
9.9.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado 95
9.10 Simulações para Analisar o Número de Mensagens Trocadas nas Operações de Balanceamento de Carga 96
9.11 Simulações para Analisar o Número de Mensagens Trocadas nas Operações de Balanceamento de Carga em Função do Número de Computadores por Nível da Árvore do Algoritmo TLBA 98
9.12 Resumo do Capítulo 100
10. IMPLEMENTAÇÃO, AVALIAÇÃO E ANÁLISE DE UM PROTÓTIPO DO TLBA 101
10.1 Considerações Iniciais 101
10.2 Objetivo da Implementação 101
10.3 Implementação 101
10.3.1 Componentes do Protótipo 101

10.3.2 Aspectos de Configuração 103
10.3.3 Considerações 104
10.4 Características e Limitações do Protótipo do Algoritmo TLBA 104
10.5 Avaliação dos resultados de execução do protótipo TLBA 105
10.6 Resumo do Capítulo 110
11. APLICABILIDADE DO ALGORITMO TLBA 111
12. CONCLUSÕES E TRABALHOS FUTUROS 112
REFERÊNCIAS 113
APÊNDICE A – Configuração do Ambiente Homogêneo Usado para Executar o Protótipo 123
APÊNDICE B – Configuração do Ambiente Heterogêneo Usado para Executar o Protótipo 125
APÊNDICE C – Artigos Publicados e Aceitos para Publicação 127
APÊNDICE D – Códigos-Fontes do Emulador, Simulador e do Protótipo do Algoritmo TLBA 128

1. INTRODUÇÃO

A disponibilidade de microprocessadores de baixo custo e a evolução das redes de computadores viabiliza economicamente o desenvolvimento dos sistemas distribuídos. Nesses sistemas os processos executam sobre os computadores de uma rede, comunicando-se para a realização de um mesmo trabalho computacional. A distribuição desses processos pode ser feita de forma manual ou automatizada, por meio de um algoritmo de balanceamento de carga.

Algoritmos de balanceamento de carga buscam distribuir de forma eqüitativa a carga de processos entre todos os computadores do sistema. Krueger e Livny [1] demonstram que os métodos de balanceamento de carga reduzem as médias e desvios padrão dos tempos de resposta dos processos. Baixos tempos de resposta compreendem menor tempo de execução de processos e, conseqüentemente, maior desempenho.

Algoritmos de balanceamento de carga envolvem as políticas de: transferência, seleção, localização e informação [2, 3]. A política de transferência classifica um computador como emissor ou receptor de processos de acordo com sua ocupação. A política de seleção define o processo que deve ser transferido do computador mais ocupado para o mais ocioso. A política de localização define que computador deve receber ou enviar um processo para transferência. Um computador servidor ou emissor oferece processos, pois está carregado; um computador receptor requer processos, pois está ocioso. A política de informação define quando e como as informações sobre a ocupação dos computadores são atualizadas no sistema. Estas informações são utilizadas pela política de localização.

Diversos métodos de balanceamento de carga foram propostos [2, 15, 17-33, 38-42]. Segundo Shivaratri, Krueger e Singhal [2], estes algoritmos podem ser subdivididos em grupos de acordo com suas características. Esses grupos são: iniciados pelo receptor, iniciados pelo servidor, simetricamente iniciados, estáveis iniciados pelo

servidor e estáveis iniciados simetricamente. Desses algoritmos, o que apresenta melhor desempenho é o estável simetricamente iniciado.

Recentemente Mello, Paiva e Trevelin, durante estudos sobre clusters e sistemas distribuídos [4-7], propuseram outras técnicas para balanceamento de carga. Concluíram que escolhendo um índice de carga baseado no comportamento dos processos e na capacidade de processamento dos computadores do ambiente, seria possível obter melhores resultados para algoritmos iniciados pelo servidor [8]. Ainda em [8] os autores apresentam um estudo comparativo do algoritmo Lowest, utilizando um novo índice de carga que é baseado no comportamento dos processos e capacidade de processamento dos computadores, e do algoritmo estável simetricamente iniciado. Segundo os resultados obtidos em [8], o algoritmo Lowest, utilizando um índice de carga baseado na análise da ocupação de CPU e memória, obtém maior desempenho que um índice gerado a partir do comprimento da fila de processos. Essas análises permitem concluir que a escolha de um índice de carga adequado modifica o comportamento do algoritmo de balanceamento de carga.

Da análise dos algoritmos acima foi possível obter os principais aspectos que qualificam um algoritmo de balanceamento de carga. Estes aspectos são: a estabilidade no número de mensagens geradas no ambiente, que não devem sobrecarregar o meio de comunicação; o suporte a ambientes compostos por computadores de capacidade de processamento heterogênea; a atualização das informações de carga do ambiente, sem afetar o desempenho das aplicações; o baixo custo na escolha do computador ideal para executar um processo recebido pelo sistema; a estabilidade em alta carga; a escalabilidade do sistema; e baixos tempos de resposta. O estudo e a análise de cada um desses aspectos permitiu propor um novo algoritmo de balanceamento de carga, denominado TLBA (*Tree Load Balancing Algorithm*) [9].

Este projeto de doutorado apresenta a proposta e avaliação de desempenho do algoritmo TLBA, um algoritmo de balanceamento de carga voltado para ambientes heterogêneos de alta escalabilidade. Este algoritmo agrega contribuições significantes na queda do número de mensagens geradas pelas operações de balanceamento de carga, na estabilidade em altas cargas e, principalmente, nos baixos tempos médios de resposta dos processos, que implicam em alto desempenho.

Este trabalho é subdividido nos seguintes capítulos: 2) os conceitos de balanceamento de carga; 3) o estado da arte em algoritmos de balanceamento de carga; 4) os sistemas de balanceamento de carga existentes; 5) os tipos de sistemas distribuídos que se beneficiam de algoritmos de balanceamento de carga; 6) as principais técnicas de avaliação de desempenho; 7) a proposta do algoritmo TLBA; 8) um emulador construído para analisar o comportamento do TLBA; 9) resultados obtidos através de um simulador do TLBA e do algoritmo Lowest modificado; 10) a implementação e resultados obtidos de um protótipo do TLBA; 11) a aplicabilidade do TLBA; 12) as conclusões e trabalhos futuros relacionados ao TLBA; finalmente, são apresentadas as referências e os apêndices.

2. BALANCEAMENTO DE CARGA

2.1 Considerações Iniciais

Um algoritmo de balanceamento de carga é, antes de tudo, um algoritmo de escalonamento de processos com algumas características específicas. O principal resultado dessas características é a obtenção da distribuição eqüitativa de carga sobre os recursos computacionais de um ambiente distribuído. Essa distribuição eqüitativa maximiza a utilização dos recursos e permite obter alto desempenho [1, 2, 3, 10-13].

Para compreender melhor o termo balanceamento de carga e outros termos utilizados para classificar algoritmos de escalonamento de processos, as seções a seguir apresentam a taxonomia de Casavant e Kuhl [14]. Essa taxonomia destaca-se pela sua abrangência de classificação e aceitação pela comunidade.

2.2 Taxonomia de Algoritmos de Escalonamento de Processos

A taxonomia de Casavant e Kuhl [14] enumera e define os termos utilizados para qualificar os algoritmos de escalonamento de processos. Essa taxonomia é subdivida em duas classificações: a hierárquica e a plana. Tais classificações são apresentadas a seguir.

2.2.1 Classificação Hierárquica

A classificação hierárquica divide os algoritmos de escalonamento de processos em dois tipos: locais e globais. Os algoritmos locais são caracterizados por executar sobre um único processador ou computador, enquanto que os globais, por executar sobre vários processadores ou computadores.

O escalonamento de processos global é segmentado em dois tipos: o estático e o dinâmico. No escalonamento estático as alocações de recursos são definidas antes da execução das aplicações. Essa forma de escalonamento é fixa e não pode ser modificada durante a execução das aplicações. O escalonamento dinâmico define a alocação de recursos durante a execução das aplicações.

O escalonamento estático é definido na fase de implementação das aplicações. Para definir as alocações estáticas, os projetistas envolvidos devem ter conhecimentos suficientes sobre o comportamento dos processos. Esta forma de escalonamento é de implementação complexa e restrita, dado que necessita de informações do sistema antes de sua execução.

No escalonamento dinâmico as decisões de alocação de recursos são tomadas durante a execução dos processos. Para tomar essas decisões, os algoritmos de escalonamento realizam análises de comportamento do sistema. Essas análises podem priorizar a execução de um processo, a ocupação de determinado recurso ou outro aspecto. Esta forma de escalonamento oferece bom desempenho para ambientes onde o comportamento prévio dos processos é desconhecido.

Apesar do benefício de não requerer informações sobre o comportamento prévio dos processos, os algoritmos de escalonamento dinâmicos apresentam uma limitação significante. Estes algoritmos, quando mal projetados, podem sobrecarregar o sistema durante a obtenção e análise dinâmica de informações utilizadas na tomada de decisões de alocação de processos.

Os algoritmos de escalonamento estático são subdivididos em ótimos e sub-ótimos. Os algoritmos de escalonamento dinâmicos são subdivididos em dois tipos: fisicamente distribuídos e não distribuídos. Este projeto de doutorado trabalha com decisões dinâmicas de alocação de processos, por isto somente algoritmos desse tipo são considerados.

Os algoritmos de escalonamento dinâmicos e fisicamente distribuídos são compostos de módulos, que executam sobre cada um dos processadores ou computadores. Estes módulos descentralizam a tomada de decisões de escalonamento. Os algoritmos de escalonamento dinâmicos e fisicamente não distribuídos têm um

módulo central, localizado em um processador ou computador, que toma todas as decisões de escalonamento.

Os algoritmos de escalonamento dinâmicos e fisicamente distribuídos são subdivididos em cooperativos e não cooperativos. Os cooperativos tomam decisões levando em consideração informações capturadas dos módulos, que executam em outros computadores ou processadores. Nesse caso, os módulos trabalham em conjunto para um objetivo comum do sistema. Nos algoritmos não cooperativos, os módulos tomam decisões sem considerar informações dos demais. Nesse caso, os módulos concorrem entre si para obter alto desempenho para suas próprias aplicações.

2.2.2 Classificação Plana

Além da classificação hierárquica, Casavant e Kuhl [14] definiram outros termos para a classificação de algoritmos de escalonamento, que não se encaixam na classificação hierárquica. Estes termos, apresentados a seguir, fazem parte da classificação plana.

1) Algoritmo adaptativo - um algoritmo é definido como adaptativo quando pode alterar seu comportamento de acordo com a situação atual do sistema. Os algoritmos adaptativos podem redefinir, dinamicamente, regras e parâmetros das políticas de escalonamento buscando obter maior desempenho.
2) Atribuição Inicial – neste tipo de algoritmo, os processos iniciam e finalizam sua execução no processador ou computador ao qual foram atribuídos. Esses processos não realizam migração (ou transferência).
3) Reatribuição Dinâmica – estes algoritmos utilizam a preempção, para interromper a execução de processos e a migração, para redistribuir os processos entre processadores ou computadores do ambiente.
4) Compartilhamento de Carga - compartilhar carga significa evitar a requisição de serviços a processadores ou computadores sobrecarregados, enquanto existem outros ociosos.
5) Balanceamento de Carga – balancear a carga significa manter as cargas homogeneamente distribuídas sobre todos os recursos de um ambiente. O balanceamento de carga utiliza-se da técnica de reatribuição dinâmica.

6) Licitação – refere-se às etapas de negociação entre processadores ou computadores para distribuir a carga.
7) Escalonamento Probabilístico – operações de escalonamento são geradas randomicamente, de acordo com uma função de distribuição de probabilidades. Esta função seleciona os computadores ou processadores que devem receber processos.

2.3 Balanceamento de Carga

Após identificar o termo balanceamento de carga (seção 2.2), pode-se resumir um algoritmo de escalonamento de processos, que oferece suporte a balanceamento de carga, ou simplesmente um algoritmo de balanceamento de carga, como o responsável por submeter processos ou tarefas para os processadores e computadores de um ambiente visando distribuir, de forma eqüitativa, a ocupação dos recursos computacionais disponíveis, aumentando, conseqüentemente, o desempenho das aplicações [1, 2, 3, 10, 15, 16].

Um algoritmo de balanceamento de carga deve distribuir a ocupação ou carga, de maneira compatível com a capacidade de cada recurso do ambiente e evitar que estes se tornem sobrecarregados em detrimento da ociosidade de outros. Os componentes que definem o funcionamento de um algoritmo de balanceamento de carga são:

1) A política de transferência – esta política determina se o estado de um computador permite que este participe de uma operação de transferência de processos. Em suma, a política de transferência identifica computadores emissores e receptores de processos e transfere processos. Os emissores, também conhecidos como servidores, são computadores sobrecarregados. Os receptores são computadores ociosos.
2) A política de seleção – tem como tarefa selecionar o processo que deve ser transferido para outro processador ou computador do ambiente. A transferência ou migração tem como objetivo redistribuir a carga sobre um ambiente.
3) A política de localização – busca encontrar o melhor computador receptor de processos para o melhor emissor, ou vice-versa. O melhor emissor é definido

como o computador mais sobrecarregado do sistema. O melhor receptor, como o computador mais ocioso do sistema.

4) A política de informação – define quais informações e como estas são utilizadas para definir a carga dos processadores ou computadores (recursos computacionais analisados e como estes são analisados), de que forma as informações trafegam no ambiente e a periodicidade de captura das informações. Essas informações são utilizadas para obter um único índice de carga, que identifica a ocupação de cada processador ou computador do ambiente. Esses índices são utilizados pela política de localização para encontrar emissores e receptores.

Boas políticas de um algoritmo de balanceamento de carga são aquelas que aumentam o desempenho das aplicações e não influenciam, ou pouco influenciam, a carga do sistema. Bons algoritmos de balanceamento de carga utilizam poucos recursos computacionais do ambiente e, mesmo nestas condições, conseguem alocar recursos de forma otimizada e homogênea, oferecendo, conseqüentemente, alto desempenho para as aplicações.

2.4 Resumo do Capítulo

Este capítulo apresentou os algoritmos de escalonamento de processos e sua classificação hierárquica e plana segundo Casavant e Kuhl [14]. Após identificar o termo balanceamento de carga nessa taxonomia, foram apresentadas as políticas que compõem um algoritmo de balanceamento de carga.

As classificações apresentadas neste capítulo permitem definir o algoritmo proposto neste projeto de doutorado como um algoritmo de escalonamento de processos global, dinâmico, fisicamente distribuído, cooperativo e de balanceamento de carga.

3. ESTADO DA ARTE NO BALANCEAMENTO DE CARGA

Este capítulo apresenta os principais trabalhos na área de balanceamento de carga, os quais foram utilizados como base deste projeto de doutorado.

3.1 Balanceamento de Carga Hidrodinâmico

Em [17] é apresentada uma técnica de balanceamento de carga baseada na migração de processos ou tarefas entre processadores heterogêneos vizinhos, denominada balanceamento de carga hidrodinâmico. Essa técnica utiliza conceitos de vizinhança dos processadores, desconsiderando a necessidade de um coordenador global, o que aumenta a escalabilidade e a disponibilidade do sistema.

O balanceamento de carga hidrodinâmico é desenvolvido para ambientes compostos de processadores autônomos, conectados por uma rede de comunicação. Esse ambiente é modelado como um grafo indireto $G = (N, E)$, em que N representa o conjunto de processadores heterogêneos assíncronos e E descreve as conexões entre estes processadores.

No balanceamento de carga hidrodinâmico, cada processador n_i é associado a um atributo $c_i > 0$, o qual define sua capacidade total de processamento. Um segundo atributo, o *load*, $l_i >= 0$ reflete a carga de trabalho atual de um processador no sistema. Considerando-se que o valor destes atributos varia ao longo do tempo, eles devem ser expressos na forma $c_i(t)$ e $l_i(t)$, onde t é o instante de leitura da informação.

Para simplificar o entendimento sobre o balanceamento de carga hidrodinâmico, suponha-se um sistema onde cilindros representam os processadores n_i, o conteúdo líquido destes cilindros é a carga l_i de cada processador. A capacidade de processamento total c_i de um processador n_i corresponde ao total de líquido que seu cilindro pode

armazenar. Os canais de comunicação entre processadores são representados por canais de troca de líquido entre cilindros.

O balanceamento de carga hidrodinâmico assume que o equilíbrio do sistema é atingindo pela troca de líquidos através dos canais que interligam os processadores. O equilíbrio é atingido quando as alturas das colunas de líquido dos cilindros são iguais, independentemente do diâmetro de cada cilindro. O diâmetro do cilindro representa a capacidade dos processadores. Após o equilíbrio do sistema, nenhum líquido flui entre os cilindros.

Essa técnica compara a tarefa de balanceamento de carga a operações sobre energia potencial, utilizando os conceitos de massa e altura dos líquidos contidos nos cilindros. Esse paralelo permite aos pesquisadores utilizar funções de energia potencial global e funções de transformação da energia potencial em cinética para atingir equilíbrio no sistema.

3.2 Análise de Modelos de Balanceamento de Carga Usando Teoria de Filas

Em [18] é apresentado um algoritmo de balanceamento de carga centralizado, que é modelado e avaliado através de teoria de filas. A política de informação desse algoritmo é baseada no conceito de um quadro de avisos central, que armazena informações de carga do sistema. Essas informações são atualizadas periodicamente. A política de localização desse algoritmo utiliza a informações do quadro de avisos para encontrar um computador receptor de processos.

O modelo desse algoritmo de balanceamento de carga utiliza uma função Poisson para a distribuição de probabilidades de chegada de processos no sistema. Após a chegada de um processo, a política de localização entra em ação e define sobre qual computador esse processo deve executar. Utilizando esse modelo de filas foram definidas equações que avaliam o comportamento deste algoritmo de balanceamento de carga centralizado.

3.3 Modelo de Custo-Oportunidade para Alocação de um Processo

Em [19] é proposto um modelo de custo-oportunidade para alocação de recursos em ambientes distribuídos heterogêneos. O objetivo desse modelo é minimizar a queda de desempenho de processos através de uma política de informação baseada em índices de carga que analisam os recursos de CPU e memória.

No modelo custo-oportunidade, os processos comportam-se como clientes que chegam para ser atendidos por um sistema de filas. Esses processos são inicialmente alocados ao computador que os iniciou. Aos processos estão associados os seguintes atributos:

1) O tempo de chegada, a(j);
2) A quantidade de memória que o processo requer, definida como m(j). Esse atributo é conhecido na chegada do processo ao sistema;
3) O número de segundos de CPU que ele requer t(j). Atributo desconhecido na chegada do processo ao sistema;

A política de informação desse modelo cria um índice de carga único, baseado nas ocupações de CPU e memória. No cálculo desse índice são considerados os consumos de cada processo em execução. Para gerar esse índice é necessário, primeiramente, definir a carga ou ocupação do recurso de CPU ($l_{cpu}(t,i)$) apresentada na eq.(1) e a de memória ($l_m(t,i)$) apresentada na eq.(2), onde: t é o instante de análise de carga; i é o computador analisado; J(t,i) é o conjunto de processos do computador i no instante t; m(j) é a ocupação de memória do processo j.

$$l_{cpu}(t,i) = |J(t,i)| \tag{1}$$

$$l_m(t,i) = \sum m(j), j \in J(t,i) \tag{2}$$

Tendo-se definido a carga sobre os recursos de CPU e memória, apresenta-se o índice único de carga, $l_c(t,i)$, na eq.(3), onde: t é o instante de análise de carga; i é o computador analisado; l_m é a carga de memória; l_{cpu} é a carga de CPU; τ é o fator de queda de desempenho; r_m é a capacidade total, em bytes, do recurso de memória. Esse índice considera um fator τ de queda de desempenho em situações onde não há memória

suficiente para os processos. A queda de desempenho ocorre pelas falhas de página e *swap* em disco.

$$l_c(t,i) = l_{cpu}, \quad se \quad l_m(t,i) < r_m(i) \quad e \quad l_c(t,i) = l_{cpu} * \tau, \quad caso \quad contrário. \qquad (3)$$

Em suma, o objetivo desse modelo custo-oportunidade é obter um índice de carga único que represente, o mais próximo possível do real, a ocupação de CPU e memória no sistema.

3.4 Modelo de Análise da Queda de Desempenho para Aplicações que Executam em Clusters de Workstations de Tempo Compartilhado

Em [21] é apresentado um modelo para calcular a queda de desempenho (*slowdown*) imposto por aplicações em clusters multiusuários de tempo compartilhado. São analisados três tipos de queda de desempenho: a queda de desempenho local, que sintetiza o efeito da contenção por processamento (CPU) em um computador; a queda de desempenho na comunicação, que sintetiza o efeito da contenção de canais de comunicação entre computadores; a queda de desempenho agregada, que determina o efeito da contenção de tarefas paralelas, causada por outras aplicações que executam sobre o cluster.

Os tipos de queda de desempenho são analisados com o objetivo de se obter o tempo de execução de processos submetidos a sistemas com tais contenções. Utilizando-se os três tipos de queda de desempenho apresentados são obtidos fatores que, multiplicados pelo tempo de execução de um processo em um ambiente dedicado, permitem predizer o tempo de execução desse processo em um ambiente distribuído concorrente. O tempo de execução dos processos em um ambiente dedicado é previamente conhecido.

Este modelo assume que os fatores de queda de desempenho dependem das características do sistema e de seu comportamento durante a execução. Por isso, esses fatores são calculados em tempo de execução, depois são multiplicados pelo tempo de execução de cada processo em ambiente dedicado, obtendo-se, finalmente, uma predição do término da execução dos processos.

3.5 Estudos de Balanceamento de Carga Dinâmico Usando Algoritmos Genéticos

Em [22] são utilizados algoritmos genéticos para definir um novo algoritmo de balanceamento de carga. Os algoritmos genéticos são utilizados para a busca baseada nos princípios da evolução e da genética natural. Nesses algoritmos são definidas gerações, que são coleções de criaturas artificiais. Em cada nova geração um conjunto destas criaturas é criado usando informações das gerações anteriores.

A técnica proposta nesse trabalho descreve um ambiente onde existe um número fixo de tarefas, cada uma com seu identificador e tamanho. Essas tarefas são randomicamente geradas e associadas a processadores. O balanceamento de carga proposto é executado por um algoritmo genético descentralizado, que tem como objetivo encontrar a população de soluções possíveis, ou seja, alocações possíveis para os processos do sistema.

Cada geração é representada por uma *sliding-window*, a qual permite identificar processos iniciados em um mesmo intervalo de tempo. Os processos são as criaturas artificiais submetidas ao algoritmo de balanceamento de carga para escalonamento. Após a alocação destes processos inicia-se uma nova *sliding-window*, que representa uma nova geração de processos. O balanceamento de carga é iniciado quando um processador ocioso é detectado.

Para simplificar a compreensão sobre o funcionamento desse algoritmo genético de escalonamento, considere-se um sistema em que, no seu estado atual, os processadores encontram-se ocupados, de acordo com a tabela 1. A figura 1 apresenta um exemplo de *sliding-window* que contém 10 processos para escalonamento.

Os processos de uma nova *sliding-window*, tal como os representados pela figura 1, são submetidos ao algoritmo genético. O algoritmo balanceia a carga por meio das unidades de tempo que cada processador fica ocupado. Dessa forma, cada processador tem um número semelhante de unidades de tempo para processar.

TABELA 1 – Ocupação dos processadores em um ambiente que utiliza um algoritmo genético para o balanceamento de carga

Processador	Processo em Execução - identificador do processo (unidades de tempo consumidas)	Fila de Processos	Comprimento total da fila em unidades de tempo
1	6(1)	5(2), 3(4)	7
2	8(6)		6
3	7(0)		0
4	1(8)	9(5)	13
5	4(8)		8
	Total de carga no sistema		34

11(9)	12(8)	13(7)	14(9)	15(17)	16(18)	17(4)	18(11)	19(17)	20(5)

FIGURA 1 – Exemplo de *sliding-window* do algoritmo genético

3.6 Impacto da Carga de Trabalho e dos Parâmetros de Sistema nos Mecanismos de Escalonamento de Processos em Clusters

Em [23] são analisadas as estratégias de escalonamento de processos em clusters computacionais baseadas na técnica do co-escalonamento. Nessa técnica, os processos são alocados simultaneamente em seus respectivos processadores. A esses processos é atribuído um mesmo *quantum* de processamento. Após a expiração desse tempo, ocorre uma sincronização entre os processadores. Essa sincronização decide sobre as próximas operações de escalonamento de processos.

O maior problema da técnica de co-escalonamento é a constante sincronização entre processadores, para definir os escalonamentos que serão feitos. A técnica proposta nesse trabalho comprova que, através de interfaces de rede e protocolos de comunicação, pode-se minimizar os impactos da sincronização entre computadores. Essa nova técnica é denominada mecanismo de co-escalonamento dinâmico.

As interfaces de rede e protocolos permitem que os processos troquem mensagens, mesmo estando inativos. Ao enviar ou esperar por uma mensagem, o contexto pode ser chaveado e o processo entrar em inatividade. Durante operações de entrada e saída, outros processos podem ocupar os processadores. Desta forma os processadores podem se sincronizar, através de processos que utilizam mensagens em rede.

Dois métodos de co-escalonamento dinâmico são apresentados nesse trabalho: o Spin Yield e o Periodic Boost.

No método Spin Yield um processo, que envia mensagem, diminui sua prioridade e aumenta a prioridade de outro processo. Desta forma, processos com menos mensagens pendentes tendem a ter maior prioridade. Quando uma mensagem chega ao sistema, o escalonador é chamado para colocar em execução os processos com maior prioridade.

No método Periodic Boost é criada uma *thread* no nível do *kernel* para analisar as filas de mensagens e modificar a prioridade dos processos com base no número de mensagens pendentes. Quando o escalonador tornar-se ativo, ele coloca os processos com maior prioridade em execução. O Periodic Boost não chama o escalonador a cada recepção de mensagem. Por isso, ele ocupa menos recursos do sistema e garante maior estabilidade.

3.7 Balanceamento de Carga Randômico Utilizando Duas Opções

Em [24] é apresentado um modelo de filas para estudar e avaliar o comportamento da política de localização de um algoritmo de balanceamento de carga iniciado pelo servidor, semelhante ao Lowest. Nesse modelo, cada computador apresenta uma fila com política FIFO (*first-in first-out*) de atendimento e os processos chegam ao sistema segundo uma função Poisson de distribuição de probabilidades.

O modelo proposto tem uma política de localização onde o computador, que inicia processos, seleciona um subconjunto de computadores, analisa suas respectivas cargas e escolhe o mais ocioso para receber o processo iniciado. São apresentadas análises dessa política para diferentes tamanhos de subconjunto. Essas análises comprovam que subconjuntos de dois elementos diminuem exponencialmente o tempo de espera dos processadores nas filas do sistema e, conseqüentemente, aumentam expressivamente o desempenho dos processos. Ao aumentar o tamanho desses subconjuntos, a queda no tempo de espera torna-se um fator constante.

Esse modelo permite concluir que políticas de localização baseadas na análise de dois computadores aumentam exponencialmente o desempenho na execução de aplicações em um ambiente distribuído. Acima de dois elementos, os subconjuntos agregam ganhos constantes e não tão expressivos no desempenho das aplicações. De

outro lado, esse aumento no tamanho dos subconjuntos causa acréscimo no número de mensagens geradas no meio de comunicação, podendo sobrecarregá-lo.

3.8 Heurísticas de Compartilhamento Dinâmico de Carga Baseada em Predição de Recursos por Processo

Em [25] são apresentadas técnicas para o compartilhamento dinâmico de carga entre processadores, baseadas na predição dos recursos necessários para cada processo. São apresentados quatro métodos que seguem tais técnicas: MINQ, MINRT, DMINQ e FDMINQ.

O método MINQ determina a carga dos processadores através de predição da ocupação de CPU e vazão de entrada e saída necessária pelos processos. Os processadores com menor carga recebem processos iniciados no sistema.

O método MINRT utiliza a predição de ocupação de recursos para estimar os tempos de resposta dos processos em cada um dos processadores do sistema. Após essa etapa, o método associa processos ao processador com menor tempo de resposta, ou seja, o que oferece maior desempenho.

O método DMINQ é uma versão distribuída do MINQ, em que uma cópia do método executa em cada processador do sistema. FDMINQ é uma variação de DMINQ, onde são filtrados os processos com menor ocupação para que sejam executados localmente.

Os quatro métodos funcionam através de dois módulos, um para escalonar e outro para predizer. Os dois módulos estão contidos em um componente, denominado escalonador central. Quando um processo chega ao escalonador central, seu identificador é enviado para o módulo preditor, que analisa a ocupação do processo gerando valores de predição para cada recurso do sistema. Os valores de predição são enviados para o escalonador, que aloca o processo no sistema. Ao término da execução de um processo, seus dados de ocupação de recursos são capturados pelo preditor, que os utiliza como base de conhecimento para futuras operações de predição.

3.9 Estratégias para o Balanceamento de Carga em Computadores Altamente Paralelos

Em [26] são apresentadas as seguintes técnicas de balanceamento de carga para ambientes multiprocessados: Sender Initiated Diffusion, Receiver Initiated Diffusion, Hierarchical Balancing, Gradient Model e Dimension Exchange.

A técnica Gradient Model é baseada no conceito de que os processadores mais ociosos informam seus processadores vizinhos sobre seu estado. O processador sobrecarregado responde a tal mensagem enviando uma porção de sua carga para o processador ocioso. Esse método, iniciado pelo receptor de processos [2], cria uma matriz virtual de processadores, onde o gradiente é definido de acordo com a carga dos processadores vizinhos [27].

A técnica Sender Initiated Diffusion baseia-se no conceito de que os processadores sobrecarregados requisitam a informação de carga de seus vizinhos. Os vizinhos com carga abaixo de um determinado *threshold* receberão parte da carga do processador mais carregado. Este método é iniciado pelo servidor de processos [2].

A técnica Receiver Initiated Diffusion é baseada no conceito de que os processadores ociosos requisitam a informação de carga de seus vizinhos (método iniciado pelo receptor). Os vizinhos com carga acima de um determinado limite recebem mensagens para dividir suas cargas com os processadores ociosos.

A técnica Hierarchical Balancing organiza um sistema em uma árvore binária, descentralizando o processo de balanceamento de carga. Os processadores de um nível recebem informações de carga do nível imediatamente seguinte, tomando como base o fato de que a raiz da árvore é o primeiro nível. Para atingir o balanceamento global de carga são enviadas mensagens ascendentes pela árvore. Essas mensagens contêm informações sobre a carga dos processadores. Quando um nível da árvore detecta uma situação de desbalanceamento em seus níveis imediatamente seguintes, uma operação de balanceamento de carga é criada nesse ramo da árvore, para balanceá-lo. Essa estrutura em árvore minimiza a sobrecarga no meio de comunicação. Essa técnica é aplicada somente para sistemas multiprocessados compostos por árvores simétricas.

A técnica Dimension Exchange foi criada para executar em arquiteturas hipercubo. Nessa técnica há um processador para cuidar do balanceamento de carga em cada dimensão do hipercubo, visando atingir o balanceamento global.

3.10 Conjuntos de Envio e Recebimento de Informações para Ambientes Multiprocessados

Observando as evoluções nos estudos de Ni, Xu e Gendreau em [28] e Ryou e Wong em [29], Suen e Wong realizaram novas pesquisas para otimizar a migração de processos e minimizar o número de mensagens em uma rede de processadores [30]. Nesses estudos foram aplicados conceitos de planos de projeção finita para a criação de grupos que enviam e recebem informações da carga do sistema. Esses estudos resultaram em grupos, bem dimensionados, de emissão e recepção de mensagens. O tamanho obtido para esses grupos, também chamado de conjuntos, é de $k = \sqrt{N}$, onde k é o tamanho do grupo e N é o número de processadores na rede.

Em [31] Tiemeyer e Wong, baseados nos estudos de Ryou, Wong e Suen, propuseram uma nova técnica de formação de grupos de envio e recepção de mensagens de carga do sistema. Essa técnica foi baseada no conceito de grupos isolados [32], onde cada processador define seu grupo para atualizar informações de carga.

Utilizando grupos isolados, um processador envia suas informações de carga para um grupo de processadores e recebe informações atualizadas de outros processadores. Desta forma, as informações de carga são propagadas por todo o ambiente sem haver grande número de mensagens.

3.11 Um Estudo Experimental de Desempenho em Algoritmos de Balanceamento de Carga

Em [16] são analisados algoritmos de balanceamento de carga em sistemas distribuídos fracamente acoplados. Os algoritmos analisados apresentam dois componentes: o LIM (*Load Information Manager*) e o LBM (*Load Balance Manager*). O LIM é responsável pela política de informação (seção 2.3), que monitora a carga dos computadores do sistema e calcula índices únicos de carga. O LBM é responsável pela

política de localização, que busca pelo melhor computador do ambiente para executar processos iniciados. Os algoritmos analisados e seus funcionamentos são apresentados a seguir:

1) Disted – Neste algoritmo os computadores calculam, periodicamente, seus índices de carga. Caso um índice seja significantemente diferente do último índice calculado, este é enviado para todos os LIMs dos computadores do sistema. Quando um processo é iniciado, o LBM utiliza as informações do LIM local para localizar o computador receptor de processos. Esse computador é o que apresenta menor índice de carga. Se a carga de todos os computadores estiver acima de um limite, o processo é executado no computador que o iniciou.
2) Global – Este algoritmo define um LIM mestre que recebe todos os índices de carga do sistema, enquanto que os demais computadores executam LIMs escravos. Os escravos calculam o índice de carga local e o submetem ao mestre, quando detectam mudanças significativas na carga. Periodicamente, o LIM mestre envia um vetor contendo os índices de carga para todos os LIMs escravos. Os processos iniciados são submetidos ao LBM que analisa, por meio das informações do LIM local, em qual computador o processo deve ser executado. Caso todos computadores estejam sobrecarregados, o processo é executado localmente.
3) Central – Neste algoritmo há um LBM e um LIM mestre. Ambos centralizam a tomada de decisões de balanceamento de carga. O LIM mestre recebe os índices de carga enviados pelos LIMs escravos. O LBM mestre recebe requisições para alocar processos no sistema e utiliza as informações do LIM mestre para tomar essas decisões.
4) Lowest – Este algoritmo troca 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 a sua carga e seleciona o mais ocioso, que deve receber o processo iniciado. Este método difere do Disted, Global e Central, pois o número de mensagens trocadas no sistema independe do número de computadores [16].
5) Random – Este algoritmo não manipula nenhuma informação sobre a carga dos computadores do sistema. Quando uma requisição para execução de um processo chega ao sistema, é definido randomicamente um computador para executá-lo.

A métrica de análise comparativa dos algoritmos apresentados foi efetuada em termos de tempo médio de resposta dos processos. 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 esta técnica, mais carregados.

Esses estudos levam à conclusão de que o desempenho dos algoritmos é muito próximo, com exceção do Random. Conclui-se, também, que os algoritmos de atualização periódica de informações de carga, tais como o Global, Central e Disted, fazem a melhor alocação de processos. Contudo, consomem muitos recursos do sistema, sobrecarregando, conseqüentemente, o meio de comunicação.

A partir de todas as análises efetuadas, foi possível concluir também que algoritmos centralizados, tais como o Global, respondem melhor para ambientes onde a carga é constantemente alterada, enquanto algoritmos distribuídos, tais como o Lowest, impõem menor sobrecarga no sistema e atingem maior escala. Finalmente, conclui-se que o algoritmo Lowest é o mais indicado para ambientes distribuídos.

3.12 Algoritmos Microeconômicos para o Balanceamento de Carga em Sistemas Computacionais Distribuídos

Em [33] é descrita uma técnica de balanceamento de carga baseada nos conceitos da microeconomia, que são caracterizados pela competição descentralizada para alocação de recursos. Os recursos são globais e disputados por todos os processos. Nessa técnica, cada processo recebe um capital inicial quando entra no sistema, que é aplicado na obtenção de recursos.

Na técnica microeconômica de balanceamento de carga, os processadores vendem sua capacidade de CPU e largura de banda dos canais de comunicação. Esses processadores definem preços para seus recursos. O objetivo dos processadores é ter o maior lucro possível. No algoritmo de balanceamento de carga, todas as tomadas de decisões são acionadas por investimentos que os processos realizam durante sua alocação no sistema. Os investimentos buscam maximizar o desempenho dos processos. Para realizar os investimentos, os processos avaliam as seguintes políticas para a compra de recursos:

1) Por preço – em que um processo quer ser servido pelo menor preço possível.
2) Por tempo de serviço –em que um processo busca ser atendido o mais rápido possível.
3) Por tempo de serviço e preço – em que um processo considera o tempo de atendimento e o preço pago pelos recursos.

A conclusão obtida nessa pesquisa é que há possibilidade de aplicar um modelo microeconômico na construção de um algoritmo de balanceamento de carga, pois foram comprovados ganhos de desempenho em relação a um ambiente sem balanceamento de carga. Contudo, esta técnica não agrega contribuições significativas de desempenho quando comparada aos demais algoritmos.

3.13 Busca por Computadores Ociosos em Sistemas Distribuídos Baseados em Workstations

Em [34] são apresentadas algumas técnicas de balanceamento de carga para ambientes compostos de *workstations*. Essas técnicas visam selecionar e atribuir processos aos computadores mais ociosos do ambiente. Duas técnicas são propostas:

1) Técnica para sistemas de escalonamento centralizado – utiliza coleta periódica de informações dos computadores do sistema, tal como o algoritmo Central [16]. Essas informações são centralizadas em um único computador, que recebe requisições para alocação de processos.
2) Técnica para sistemas de escalonamento descentralizado – visa atingir alta disponibilidade, evitando pontos centralizados que refletem em falhas no sistema. Quando um computador inicia um processo, ele envia uma mensagem para um grupo *multicast* requisitando informações de carga e aguarda as N primeiras respostas. Essas respostas são utilizadas para encontrar o computador mais ocioso do sistema, o qual deve receber o processo iniciado.

Através dos resultados obtidos nessa pesquisa concluiu-se que o desvio padrão na carga média de um ambiente, utilizando a técnica descentralizada, foi de 4% ao considerar as 3 primeiras respostas do grupo *multicast*, e de 2% ao considerar as 6 primeiras respostas. Esses desvios permitem comprovar que há um bom balanceamento.

Ainda foram realizadas comparações de escalabilidade das duas técnicas. Nessas comparações o sistema descentralizado suportou centenas de computadores interconectados, enquanto que o centralizado, milhares. Em suma, a conclusão final foi que a técnica centralizada oferece maior desempenho. Contudo, apresenta limitações para alta disponibilidade. A técnica descentralizada apresenta suporte à alta disponibilidade. Porém, tem pior desempenho.

3.14 Distribuição de Carga em Sistemas Distribuídos sobre Redes Locais

Em [2] são classificados e analisados os seguintes tipos de algoritmos de distribuição e balanceamento de carga:

1) Algoritmos iniciados pelo emissor – algoritmos onde a distribuição de carga é iniciada pelos computadores sobrecarregados, que visam transferir parte de sua carga para computadores ociosos. Há três tipos de algoritmos iniciados pelo emissor:
    a. Randômico – neste algoritmo nenhuma informação sobre a carga dos computadores é considerada. Todo processo iniciado é transferido para um computador escolhido randomicamente. Ao detectar um aumento de carga, o computador realiza uma escolha randômica e transfere parte de seus processos para o computador escolhido como destino. Para evitar migrações freqüentes define-se um número máximo de migrações por processo.
    b. Threshold – nesta política, o algoritmo de balanceamento de carga seleciona randomicamente um computador e o adiciona em um vetor de computadores disponíveis. Após essa etapa, a carga do computador adicionado é analisada. Se exceder a um determinado limite, esse computador é definido como sobrecarregado e procura-se por outro. Caso contrário, o computador recebe o processo iniciado. Todos os computadores do sistema têm seu próprio vetor. O vetor tem um tamanho limite. Caso o vetor seja preenchido por computadores sobrecarregados, o processo é executado localmente. Este algoritmo não garante a escolha do computador mais ocioso do sistema, mas

garante a escolha de um computador onde o número de processos na fila esteja abaixo de um limite.

c. Shortest – neste algoritmo cada computador apresenta um vetor de tamanho pré-definido. Quando um computador inicia um processo, ele seleciona um subconjunto de computadores e os adiciona no vetor. Após esta etapa, são realizadas comparações de carga e opta-se pelo computador mais ocioso. Se o computador mais ocioso estiver mais carregado que o computador que iniciou o processo, o processo é executado localmente.

2) Algoritmos iniciados pelo receptor – algoritmos onde as migrações de processos são iniciadas pelos computadores ociosos. Para isso, é analisado um algoritmo estudado por Livny e Melman [36], e Eager, Lazowska e Zahorjan [37]. Nesse algoritmo, um computador ocioso seleciona computadores randomicamente e os adiciona em um vetor local. Após essa etapa, a carga dos computadores é analisada. Caso esteja alta, o computador ocioso requisita parte da carga. O vetor tem um tamanho pré-definido. Se o vetor for preenchido e não for localizado nenhum computador sobrecarregado, nenhuma atitude é tomada. Mais tarde, essas operações são reiniciadas.
3) Algoritmos iniciados simetricamente – iniciam a migração de processos através dos emissores e receptores de processos. Estes algoritmos agregam as vantagens dos dois métodos anteriores, ou seja, em baixas cargas, os algoritmos iniciados pelo emissor apresentam melhores resultados; em altas cargas, os algoritmos iniciados pelos receptores apresentam melhores resultados. De outro lado, estes algoritmos também agregam as desvantagens dos outros dois tipos de algoritmos. Os algoritmos iniciados pelos emissores, quando em altas cargas, geram instabilidade na seleção randômica de computadores. Os algoritmos iniciados pelos receptores necessitam de implementações mais complexas de preempção e migração de processos, pois é necessário um maior número de mensagens para sincronizar essas operações.
4) Algoritmos adaptativos estáveis simetricamente iniciados – este tipo de algoritmo define grupos de computadores no ambiente, de acordo com seus respectivos estados de carga. Os grupos são: emissores de processos, receptores de processos e computadores com carga média. Em cada

computador do sistema são mantidos três vetores. Cada vetor armazena computadores segundo seus grupos. Um computador é considerado emissor se estiver com carga acima de um limite superior; receptor, se estiver abaixo de um limite inferior; com carga média, se estiver entre tais limites. Este algoritmo agrega a vantagem de ser iniciado pelo receptor em ambientes sobrecarregados, iniciado pelo emissor em ambientes de baixa carga e iniciado simetricamente em ambientes de carga média.

5) Algoritmos adaptativos estáveis iniciados pelo emissor – estes algoritmos assemelham-se aos adaptativos estáveis iniciados simetricamente. Contudo, apresentam algumas diferenças que são:
    a. Um computador selecionado e adicionado em um vetor é avisado sobre essa ação.
    b. Ao se tornar receptor de processos, um computador informa aos computadores de seu vetor sobre seu novo estado.

Essa pesquisa analisa os tempos médios de resposta de processos submetidos aos algoritmos, concluindo que os estáveis iniciados simetricamente 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.

3.15 Predição do Uso de Recursos por Processo: Um Estudo de Medidas sobre UNIX

Em [38] é apresentada uma técnica para a predição da ocupação de recursos computacionais através de análises do comportamento dos processos. Essas análises são baseadas nas informações de tempo de ocupação de CPU, entrada e saída e uso de memória, obtidos no início e no final da execução de cada processo. Para realizar essa análise, são mantidos históricos dessas informações e montados diagramas de estados. Utilizando-se esses diagramas são definidas as probabilidades de transições entre estados. Os estados representam a mudança na ocupação de recursos.

Para criar diagramas de estados que consigam predizer com propriedade o comportamento dos processos, foi realizado um estudo do número de execuções

necessárias para predizer uma situação futura. Conclui-se que executando 100 vezes um mesmo processo, o erro médio em sua predição comportamental é muito baixo.

Com o objetivo de capturar os tempos de ocupação dos recursos do sistema foi utilizado o sistema operacional UNIX BSD 4.3, que oferece uma estrutura de dados com tais informações. Para capturar os instantes de início e fim da execução dos processos, as chamadas de sistema *fork* e *_exit* foram alteradas [38]. Essas chamadas permitem, respectivamente, definir o momento exato do início e final da execução de um processo.

A conclusão obtida nessa pesquisa é que 80% dos erros de predição na ocupação dos processos variam menos de 0,5% na ocupação real dos recursos. Estes resultados confirmam a possibilidade de predizer o comportamento de processos.

3.16 Escalonamento Paralelo em Tempo de Execução para Balanceamento de Carga em Processadores

Em [39] são analisados três algoritmos de balanceamento de carga para sistemas multiprocessados. Esses algoritmos trabalham com o balanceamento de tarefas entre processadores utilizando um número médio de tarefas por processador. Caso um processador apresente um número de tarefas acima da média, esse processador migra parte de suas tarefas para outro ocioso.

Os três algoritmos apresentados neste trabalho são definidos para as seguintes topologias de processadores: topologia em árvore, hipercubo e *mesh*.

Para a topologia em árvore, o algoritmo analisa o número de tarefas totais do sistema e as subdivide por subárvore, até chegar nos processadores folha.

Para a topologia hipercubo, são propostos os algoritmos DEM (não é esclarecido o significado da sigla) e CWA (*Cube Walking Algorithm*).

No algoritmo DEM, os pequenos domínios são balanceados primeiramente e então, combinados com os domínios maiores, até que todo o sistema esteja balanceado. Nenhuma informação global sobre o sistema é coletada. Quando um processador encontra-se sobrecarregado, esse processador migra suas tarefas para os vizinhos.

O algoritmo CWA captura as informações de carga dos subcubos. Cada processador tem um vetor com o número de tarefas por subcubo. Conseqüentemente, o nível mais alto do hipercubo tem o número total de tarefas do sistema. Esse número de tarefas é utilizado para calcular o número médio de tarefas por processador. Após essa etapa, as tarefas são divididas em números iguais por subcubo, até chegar nos processadores que os compõem.

Para a topologia *mesh*, é proposto o algoritmo MWA (*Mesh Walking Algorithm*), que encontra a diferença de carga, expressa pelo número de processos, em cada linha que compõe a topologia. Uma linha é formada por processadores interconectados. As linhas com mais processos realizam transferências para as linhas com menos processos, balanceando a carga do sistema.

3.17 Outros Estudos Relacionados

Além dos estudos apresentados nas seções anteriores, existem outros. Contudo, esses trabalhos não trazem colaborações significativas para este projeto de doutorado. Alguns desses trabalhos incluem:

1) Alocação de Tarefas em Redes de Processadores [40] – trata da alocação de tarefas em redes de processadores.
2) *Clustering* de Tarefas e Escalonamento para Arquiteturas de Memória Distribuída Paralela [41] – trata da divisão de programas em tarefas paralelas, através de grafos de precedência.
3) Multicomputadores Baseados em Redes: Uma Arquitetura Prática de Supercomputadores [35] – apresenta o projeto Néctar, que desenvolveu um multicomputador de alto desempenho usando redes de computadores. O objetivo é apresentar o funcionamento de redes de computadores baseadas em *switches crossbar* de alta velocidade, protocolos para alta largura de banda e baixa latência de comunicação.

3.18 Análise dos Estudos

Os principais trabalhos na área de balanceamento de carga foram apresentados neste capítulo. Esses trabalhos resumem esforços relacionados à otimização na

ocupação de recursos do ambiente, visando aumento no desempenho das aplicações. A principal técnica apresentada para aumentar o desempenho foi a utilização de algoritmos de balanceamento de carga.

Apesar de existir diversos trabalhos na área de balanceamento de carga, pouco se tem agregado a eles desde os trabalhos propostos por Zhou e Ferrari [16], Theimer e Lantz [34], e Shivaratri, Krueger e Singhal [2]. Várias análises ainda comprovam que esses últimos são os principais trabalhos da área [2, 15, 17-33, 38-42].

Grande parte dos trabalhos recentes preocupa-se em criar uma nova abordagem para abstrair as operações de balanceamento de carga e compará-las a outras atividades. Nesses trabalhos há maior preocupação com a forma do que com os resultados de desempenho. Alguns desses trabalhos incluem: o balanceamento de carga hidrodinâmico [17], que cria paralelos das operações de balanceamento de carga ao cálculo de energia potencial e cinética em cilindros contendo água; os estudos de balanceamento de carga dinâmico, usando algoritmos genéticos [22]; a técnica microeconômica [33].

Há ainda outros trabalhos que buscam aumento de desempenho através do balanceamento de carga. Contudo, sua contribuição é pequena. São exemplos destes trabalhos: a análise de modelos de balanceamento de carga usando teoria de filas [18]; os conjuntos de envio e recebimento de informações para ambientes multiprocessados [31]; a busca por computadores ociosos em sistemas distribuídos baseados em *workstations* [34].

Existem outros trabalhos mais relevantes que visam resolver pontos específicos do balanceamento de carga. Alguns destes trabalhos são: o modelo de custo-oportunidade para alocação de processos [19]; o modelo de *slowdown* para aplicações que executam em clusters de *workstations* de tempo compartilhado [21]; o balanceamento de carga randômico utilizando duas opções [24]; as heurísticas de compartilhamento dinâmico de carga baseada em predição de recursos por processo [25]; as estratégias para o balanceamento de carga para computadores altamente paralelos [26]; um estudo experimental de desempenho em algoritmos de balanceamento de carga [16]; distribuição de carga em sistemas distribuídos de redes locais [2]; predição do uso de recursos por processo [38]; escalonamento paralelo em tempo de execução para balanceamento de carga em processadores [39].

A análise dos trabalhos apresentados permitiu concluir que a grande maioria utiliza como base os estudos de Zhou e Ferrari [16], Theimer e Lantz [34], e Shivaratri, Krueger e Singhal [2]. A mesma base conceitual é adotada por este projeto de doutorado. Contudo, agregaram-se outros pontos que foram obtidos através da análise dos trabalhos apresentados neste capítulo.

Os questionamentos que direcionaram o projeto de doutorado incluem: o uso de árvores de balanceamento de carga tal como as de ambientes multiprocessados [39] em ambientes distribuídos fracamente acoplados; análise da escalabilidade de uma árvore; comportamento de árvores compostas por computadores de capacidade de processamento heterogênea; utilização de *thresholds* para evitar grande número de mensagens trafegando pelo meio de comunicação.

Os questionamentos acima apresentados permitiram, de forma empírica, a observação de algumas contribuições que um algoritmo de balanceamento de carga, organizado tal como uma árvore, teria para ambientes distribuídos fracamente acoplados e escaláveis. Essas contribuições, que motivaram o desenvolvimento deste projeto de doutorado, compreenderam:

1) A criação de pequenos grupos, identificados pelos níveis da árvore, por onde as informações de carga trafegavam. Essa característica criaria grupos de tamanho limitado, que minimizariam a sobrecarga causada pelo número de mensagens no meio de comunicação.
2) O aumento da escalabilidade em sistemas baseados em Clusters, *Network of Workstations* e *Grid Computing* através do formato em árvore, que permitiria criar pequenos grupos e, conseqüentemente, gerar poucas mensagens no meio de comunicação.
3) Estabilidade, utilizando um valor de *threshold* para analisar a troca de informações de carga entre níveis da árvore. Essas informações somente seriam atualizadas entre níveis, ou seja, entre componentes dos grupos, quando ocorressem alterações significativas na ocupação dos computadores.
4) Estabilidade na transferência de processos, que deveria ser realizada em circunstâncias onde há garantias mínimas de ganho de desempenho.

Durante a pesquisa, as contribuições acima foram comprovadas através de simulações e da criação de um protótipo.

3.19 Resumo do Capítulo

Este 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. O estudo e análise desses trabalhos permitiram construir a proposta do algoritmo de balanceamento de carga TLBA, apresentado e avaliado neste projeto de doutorado.

4. PRINCIPAIS SISTEMAS DE BALANCEAMENTO DE CARGA

4.1 Considerações Iniciais

Este capítulo apresenta os principais sistemas que utilizam balanceamento de carga para oferecer alto desempenho a ambientes distribuídos. Tais sistemas são exemplos de soluções práticas que surgiram de pesquisas na área.

4.2 V-System

Este sistema de balanceamento de carga utiliza uma política de definição de estados para os computadores. Os estados estão associados às cargas dos computadores. Caso um computador modifique seu estado, ele o envia através de um *broadcast* para os demais computadores. Cada computador armazena o estado dos demais em memória. Pode-se configurar o sistema para que cada computador atualize somente um conjunto de computadores, minimizando o número de mensagens trocadas no meio de comunicação [2].

Para calcular o estado de cada computador, há um processo que, periodicamente, captura informações de ocupação de CPU, memória, características de hardware e gera um índice de carga único para todo o sistema. A política de transferência do V-System permite que somente processos recém-iniciados sejam migrados. A política de localização desse sistema define um computador como receptor de processos se estiver entre os N computadores mais ociosos. Caso contrário, o computador é um emissor de processos.

A política de localização é ativada na inicialização de processos. Nessa situação, o computador que iniciou o processo analisa a carga dos demais. Essas informações de

carga estão presentes em sua memória. Se o computador, que iniciou o processo, estiver entre os *N* computadores mais ociosos, ele executa o processo. Caso contrário, um dos computadores mais ociosos é escolhido randomicamente. Após selecionar randomicamente um dos computadores, lhe é requisitado seu estado. Caso sua ociosidade seja confirmada, ele recebe o processo iniciado. Caso contrário, as informações da memória local são atualizadas e um novo computador é selecionado randomicamente. A eleição randômica de um computador ocioso diminui a probabilidade de que vários computadores escolham, ao mesmo tempo, o mesmo receptor de processos.

4.3 Sprite

O sistema de balanceamento de carga Sprite [2] foi construído para trabalhar sobre *workstations*. Neste sistema, a política de informação utiliza um servidor central que captura o estado dos computadores do sistema. Quando um computador torna-se um receptor potencial de processos, ele avisa o computador central sobre sua mudança de estado.

A política de localização também é centralizada. Nessa política, as requisições para busca do melhor computador do sistema devem ser enviadas para o computador central, que toma as decisões de escalonamento.

A política de seleção do Sprite é manual e os processos devem ser escolhidos pelos usuários para que sejam executados remotamente. O computador que iniciou um processo é definido como emissor.

A política de transferência do Sprite não é completamente automática. Nela, um computador é identificado como um emissor logo que um usuário inicia um processo. Ao iniciar o processo o usuário envia, automaticamente, uma requisição para o computador central, que inicia uma busca randômica por um computador receptor de processos. Um computador é considerado receptor caso esteja sem ações de mouse e teclado há mais de 30 segundos e seu número de processos seja menor que o do computador que iniciou o processo.

Caso um computador receptor de processos não seja encontrado, a política de transferência inicia uma busca pelo número total de processos remotos que cada usuário

tem no ambiente. Se um usuário tem mais processos remotos que seu limite, seu processo de maior custo computacional é retirado de execução. Esse processo retorna para o computador do usuário e continua a execução local. O computador que teve recursos liberados recebe o processo iniciado no sistema.

O Sprite considera que todos os computadores do sistema podem ser utilizados por seus respectivos usuários, dando preferência aos mesmos. Ele evita migrar um processo para um computador que tenha um usuário trabalhando. Caso isto ocorra, o processo deve retornar para o computador onde foi iniciado.

4.4 Condor

O sistema de balanceamento de carga Condor [42, 43, 44] foi projetado para alocar processos de longa duração em redes de *workstations*. Nesse sistema existe um computador dedicado à tarefa de controlar o balanceamento de carga no ambiente, denominado controlador.

As políticas de seleção e transferência do Condor são similares à do Sprite, onde os usuários definem manualmente os processos que executam remotamente. Estes processos de execução remota são adicionados em uma fila local em cada computador.

A política de atualização de informações do Condor é periódica. Em intervalos de dois minutos o controlador obtém informações de carga de cada computador do sistema.

O Condor busca preservar os direitos do usuário local de utilizar seus recursos. Com este propósito, o escalonador local de cada computador realiza testes, em intervalos de 30 segundos, para saber se há usuários conectados. Se houver, o escalonador salva o estado do processo e o retira de execução. Se esse usuário continuar ativo por mais de 5 minutos, o processo é transferido para o computador que o iniciou. O processo somente será novamente transferido se o controlador encontrar um computador receptor de processos. Computadores receptores são aqueles em que seus usuários encontram-se inativos por mais de 12,5 minutos.

A política de localização do Condor utiliza um índice para cada computador do ambiente. Quando um processo é transferido para um computador, o índice deste é incrementado. Quando o controlador detecta que há algum usuário no computador destino, seu índice é decrementado. Para todo processo iniciado no ambiente, o

controlador busca o computador com maior índice, pois tem maior probabilidade de não haver usuário ativo.

4.5 Stealth

O Stealth [42, 45, 46] é voltado para o balanceamento de carga em ambientes compostos por redes de *workstations*, nas quais os usuários locais têm privilégio sobre os recursos de seus computadores. A política de seleção desse sistema de balanceamento de carga é automática.

A política de transferência do Stealth permite a transferência de processos para computadores com usuários ativos. Essa política foi baseada em estudos realizados pelos projetistas do Stealth, que concluíram que computadores com usuários ativos ficam ao menos 50% do tempo ociosos. Para atender aos usuários locais e execução de processos remotos, o Stealth cria um esquema de prioridades, em que o escalonador local aloca todos os recursos que necessita e, depois disso, os processos remotos podem alocar a fatia de recursos disponíveis.

Antes de transferir um processo, o Stealth considera um histórico de resultados da transferência de processos, que apresentavam a mesma ocupação de CPU e memória, nas mesmas condições de carga do sistema. A política de transferência do Stealth transfere processos remotos quando estes ficam muito tempo sem recursos.

4.6 PBS

O sistema de balanceamento de carga PBS (*Portable Batch System*) [42, 47] foi projetado para suportar várias políticas de escalonamento sobre ambiente heterogêneo. Seu objetivo é ser flexível e permitir que projetistas implementem o balanceamento de carga segundo suas necessidades. O PBS é composto pelos seguintes componentes:

1) Servidores de processos – são responsáveis por gerir filas de processos. Este componente recebe processos para execução e os submete aos servidores de execução ou a outros servidores de processos.
2) Servidores de execução – são responsáveis por controlar todos os processos de cada computador do ambiente. O servidor de execução é responsável por iniciar processos, monitorar e controlar o uso de recursos.

3) Monitores de recursos – obtêm a disponibilidade dos recursos e informação de uso do computador onde executam.
4) Escalonadores – manipulam informações sobre o estado dos processos contidos nas filas dos servidores de processos e sobre a disponibilidade de recursos capturada pelos monitores de recursos. Os escalonadores decidem onde cada processo deve executar.

A distribuição de responsabilidades entre os componentes do PBS permite que os projetistas definam onde cada componente será executado, flexibilizando a implementação do balanceamento de carga sobre o sistema distribuído. Essa flexibilidade é interessante, mas segundo os estudos de Henderson [20], ela compromete o desempenho.

O PBS oferece um conjunto de técnicas para que os projetistas possam definir suas próprias políticas de escalonamento de processos. Essas políticas podem ser definidas através de escalonadores escritos em:

1) Scripts de escalonamento em BASL (*Batch Scheduling Language*), linguagem desenvolvida para o PBS.
2) TCL (*Tool Command Language*).
3) Linguagem C usando as bibliotecas do PBS.

4.7 Messiahs

O sistema de balanceamento de carga Messiahs [42, 48] é baseado em princípios de autonomia que visam criar um sistema flexível, que pode ser adaptado para diferentes algoritmos de escalonamento de processos dinâmicos ou estáticos. Os princípios de autonomia do Messiahs compreendem:

1) Autonomia de projeto – o hardware e sistema operacional de cada computador pode ser projetado independentemente da arquitetura dos demais.
2) Autonomia de comunicação – cada computador pode tomar decisões independentemente de troca de informações de carga. Essa autonomia é parcial, pois nunca poderia ser total em um sistema distribuído.
3) Autonomia administrativa – cada computador define a política de alocação de seus recursos, independentemente da política dos demais.

4) Autonomia de execução – cada computador decide se atende ou não a uma requisição para executar um processo. Os computadores têm o direito de parar a execução de um processo que tenham previamente aceito.

4.8 Mosix

O sistema de balanceamento de carga Mosix [89] é baseado em algoritmos adaptativos de compartilhamento de recursos. Esse sistema executa sobre ambientes Linux compostos de computadores de capacidade de processamento homogênea.

O Mosix tem como principais contribuições o suporte a migração preemptiva de processos, ao alto desempenho, a execução de aplicações legadas e a hardwares de baixo custo.

4.9 Resumo do Capítulo

O objetivo deste capítulo foi apresentar os principais sistemas de balanceamento de carga, que foram projetados para oferecer alto desempenho a ambientes distribuídos.

O algoritmo TLBA, proposto e avaliado neste projeto de doutorado, pode ser utilizado como núcleo de um sistema de balanceamento de carga, tais como os apresentados neste capítulo.

5. APLICAÇÃO DE ALGORITMOS DE BALANCEAMENTO DE CARGA EM SISTEMAS DE COMPUTAÇÃO PARALELA DISTRIBUÍDA

5.1 Considerações Iniciais

Este capítulo apresenta três tipos de sistemas de computação distribuída, que buscam atingir alto desempenho e compartilhamento de recursos. Esses sistemas podem utilizar algoritmos de balanceamento de carga para otimizar a ocupação dos recursos e diminuir o tempo médio de resposta dos processos.

5.2 Clusters

A construção de sistemas de alto desempenho teve mudanças significativas após o advento dos sistemas paralelos multiprocessados. Anteriormente acreditava-se que o aumento de desempenho poderia ser obtido com o desenvolvimento de processadores mais eficientes [10].

A década de 1990 trouxe novos conceitos para a construção de sistemas computacionais de alto desempenho. Esses conceitos estão relacionados ao uso de redes de computadores e à aplicação de técnicas de paralelismo para obter alto desempenho. Os fatores que direcionaram os estudos de alto desempenho para redes de computadores incluem: o aumento da capacidade dos computadores uniprocessados; o aumento da largura de banda nas redes de computadores; o desenvolvimento de novos protocolos de comunicação; a facilidade de interconexão de computadores em redes; o alto custo das máquinas paralelas [10].

Os novos conceitos adquiridos na década de 1990 direcionaram os estudos para um tipo de sistema de alto desempenho, denominado Cluster. Clusters são tipos de sistemas paralelos e distribuídos, compostos por uma coleção de computadores interconectados em rede local, que atuam cooperativamente na resolução de um mesmo problema computacional. Esse conjunto de computadores aparenta, para o usuário final, um único recurso computacional [10, 49-57].

As principais contribuições dos clusters são: o desempenho computacional; a expansão dos componentes de software do sistema; a redundância de hardware e software; o acesso concorrente a múltiplos recursos; a escalabilidade; a flexibilidade de configuração; o uso de hardware de baixo custo; o baixo custo para atingir desempenho semelhante a computadores multiprocessados; a alta disponibilidade [49-57].

Apesar de oferecer os benefícios anteriormente citados os clusters apresentam algumas limitações. A principal limitação está relacionada ao crescimento incremental, onde a rede de comunicação gera gargalos para interligar muitos computadores. Os softwares não são corretamente modelados para se adaptarem a este crescimento [49-57].

5.3 Grid Computing

Grid computing é um tipo de sistema paralelo e distribuído, composto por uma coleção de recursos computacionais interconectados, que se comunicam através de redes não locais. Esse tipo de sistema surge como uma opção para o compartilhamento de recursos em larga escala [58, 59, 60].

Para compartilhar os recursos, são necessárias políticas de flexibilidade, segurança, coordenação de compartilhamento e participação dinâmica de recursos no ambiente. Essas políticas resultam em suportes para autenticação única no ambiente, gerência de permissões e acesso a recursos, procura por recursos distribuídos e outras funcionalidades [58, 59, 60].

O termo compartilhamento de recursos refere-se ao acesso direto a computadores, softwares e dados. Os recursos devem ser compartilhados seguindo políticas que definem os usuários, suas respectivas permissões e sob quais condições ocorrem estes compartilhamentos.

Uma das aplicações de *Grid Computing* compreende a obtenção de alto desempenho. Neste sentido os *Grids* buscam por algoritmos de balanceamento de carga que aloquem seus recursos da forma mais otimizada possível.

5.4 Sistemas Distribuídos baseados em Redes de Estações

O objetivo de um Sistema Distribuído baseado em Rede de Estações (*Network of Workstations* - NOW) é aumentar o desempenho de uma aplicação através do uso dos recursos computacionais ociosos de uma rede de computadores [87, 88]. Segundo Dandamudi e Piotrowski [87] os recursos interconectados em uma rede de computadores ficam em torno de 80% do tempo ociosos. Durante este período os recursos podem ser utilizados por aplicações que necessitam de maior desempenho. Nesse tipo de sistema os usuários locais têm privilégio sobre os recursos de seus computadores.

5.5 Resumo do Capítulo

Este capítulo apresentou três tipos de sistemas de computação paralela e distribuída, que podem utilizar algoritmos de balanceamento de carga para aumentar seu desempenho. Esses algoritmos aumentam o desempenho, otimizando a ocupação de recursos do ambiente.

6. TÉCNICAS DE AVALIAÇÃO DE DESEMPENHO DE SISTEMAS COMPUTACIONAIS

6.1 Considerações Iniciais

A avaliação comportamental de um sistema computacional tem grande importância para a analisar sua viabilidade econômica, seu desempenho e a segurança de suas operações. Para avaliar um sistema utilizam-se técnicas que estimam seu comportamento em diferentes situações. Tais técnicas de avaliação de desempenho fornecem resultados numéricos que permitem comparar diferentes soluções para um mesmo problema. Este capítulo apresenta as principais técnicas de avaliação de desempenho existentes.

6.2 Técnicas de Avaliação de Desempenho

As técnicas de avaliação de desempenho podem ser divididas em elementares e indiretas [61-66]. As elementares são aplicadas diretamente sobre o sistema sendo, portanto, necessário que o sistema esteja previamente construído para ser submetido a tais testes. As técnicas indiretas permitem a avaliação de um sistema antes de sua construção, constituindo-se em uma importante ferramenta para projetar sistemas.

As técnicas elementares são subdivididas em monitoramento e *benchmark*. O monitoramento é a captura de dados do sistema durante sua execução. Esses dados capturados são posteriormente analisados. O *benchmark* consiste na aplicação de uma carga específica de trabalho sobre o sistema. Essa técnica permite quantificar a capacidade do sistema de executar determinado tipo de operação.

As técnicas indiretas são subdivididas em analítica e simulação. A analítica consiste no desenvolvimento de equações que, através de probabilidade, demonstram o comportamento de um sistema. Essa técnica é aplicada em sistemas de baixa complexidade. A simulação é utilizada para solucionar sistemas com alto grau de complexidade. Na simulação é construído um modelo formal, que permite avaliar os pontos essenciais do sistema. Esse modelo é submetido a experimentos que, corretamente analisados, resultam no comportamento do sistema [67]. Um modelo formal não precisa representar todo o sistema. Contudo, deve representar as principais características a serem avaliadas.

As técnicas indiretas utilizam ferramentas analíticas para analisar o comportamento de um sistema. Essas ferramentas compreendem as Cadeias de Markov, Redes de Petri e Teoria das Filas.

As Cadeias de Markov modelam os sistemas de acordo com estados e suas probabilidades de transições. Uma transição para um estado depende somente do estado atual [68].

As Redes de Petri são baseadas em lugares, transições, arcos dirigidos e marcas. Os lugares correspondem a condições do sistema. As transições correspondem a eventos que movem as marcas de um lugar para outro. Os arcos dirigidos representam as possíveis transições entre lugares. As marcas correspondem ao estado atual de um sistema [69].

As teorias de filas são compostas por clientes, servidores e filas. Os clientes são tarefas que chegam ao sistema. Os servidores são responsáveis por atender os clientes de uma fila. As filas são criadas para que os clientes aguardem seu atendimento. A chegada de clientes e o processo de atendimento pelos servidores seguem funções de distribuição de probabilidades, as quais definem o comportamento do sistema [70].

6.3 Resumo do Capítulo

Este capítulo apresentou as principais técnicas de avaliação de desempenho. Este projeto de doutorado aplica duas destas técnicas, uma indireta e outra elementar. Ambas são utilizadas para avaliar o desempenho do algoritmo de balanceamento de carga TLBA.

A técnica indireta utilizada foi a de simulação, que permitiu avaliar e obter refinamentos do TLBA durante a fase de projeto. Essa técnica foi escolhida por ser a mais indicada para sistemas de alta complexidade.

A técnica elementar utilizada foi o *benchmark*, que foi aplicado sobre um protótipo do algoritmo TLBA e permitiu comprovar os resultados obtidos em simulação.

7. PROPOSTA DE UM ALGORITMO DE BALANCEAMENTO DE CARGA PARA AMBIENTES DISTRIBUÍDOS HETEROGÊNEOS ESCALÁVEIS

7.1 Considerações Iniciais

Conforme apresentado no capítulo 3 foram propostos diversos trabalhos na área de balanceamento de carga dentre os quais destacam-se os de Zhou e Ferrari [16], Theimer e Lantz [34], e Shivaratri, Krueger e Singhal [2]. Trabalhos mais recentes foram desenvolvidos, mas não se constituem em contribuições relevantes quanto ao desempenho [2, 15, 17-33, 38-42].

Zhou e Ferrari [16] avaliam cinco algoritmos de balanceamento de carga iniciados pelo servidor, ou seja, pelo computador mais carregado. Esses algoritmos são: o Disted, Global, Central, Random e Lowest. No Disted, quando um computador percebe a alteração em sua carga, emite mensagens para os demais, informando sua ocupação atual. No Global há um computador que centraliza todas as informações de carga dos computadores do ambiente. Esse centralizador envia *broadcasts* para o ambiente, atualizando os demais. No Central, assim como no Global, um computador central recebe todas as informações de ocupação do sistema, mas não atualiza os demais. Esse centralizador decide sobre a alocação de recursos no ambiente. No Random, nenhuma informação sobre a carga do ambiente é manipulada. Nesse algoritmo um computador é randomicamente selecionado para receber um processo a iniciado. No Lowest a informação de carga é enviada sob demanda. Quando um computador inicia um processo, ele requisita e analisa as cargas de um pequeno conjunto de computadores e submete o processo ao mais ocioso.

Analisando os algoritmos Disted, Global, Central, Random e Lowest, Zhou e Ferrari concluíram que todos aumentam o desempenho em relação a um sistema sem balanceamento de carga. O algoritmo Lowest apresenta o maior desempenho dentre esses algoritmos, pois gera mensagens sob demanda para transferência de processos, causando menor sobrecarga no meio de comunicação. O Lowest permite o crescimento incremental do sistema, pois o número de mensagens geradas no ambiente não depende do número de computadores.

Theimer e Lantz [34] implementaram algoritmos similares ao Central, Disted e Lowest. Contudo, analisaram tais algoritmos em sistemas compostos por um maior número de computadores, em torno de 70. Para o Disted e Lowest foram criados grupos de recepção e emissão de processos. A comunicação desses grupos foi feita utilizando-se um protocolo *multicast*, minimizando a troca de mensagens entre os computadores. Computadores abaixo de uma carga limite participaram do grupo de recepção de processos. Acima dessa carga, participam do grupo de emissão de processos.

Theimer e Lantz recomendam algoritmos descentralizados tais como o Lowest e Disted, pois não geram pontos únicos de falha, como ocorre no Central. O Central apresenta melhor desempenho para redes pequenas e médias, sendo degradado em ambientes compostos por muitos computadores.

Nas análises, Theimer e Lantz concluíram, também, que algoritmos que utilizam como índice de carga o comprimento da fila de processos, assumem a homogeneidade do sistema, o que não reflete a ocupação real dos computadores. Esse tipo de índice degrada o desempenho dos algoritmos de balanceamento de carga [8, 34].

Shivaratri, Krueger e Singhal [2] analisam algoritmos iniciados pelo servidor, pelo receptor, simetricamente iniciados, adaptativos simetricamente iniciados e adaptativos iniciados pelo servidor. Nesses estudos foi considerado como índice de carga o comprimento na fila de processos em espera pela CPU. Segundo os autores, essa medida foi escolhida por ser simples e, portanto, consumir menos recursos ao ser obtida.

Shivaratri, Krueger e Singhal concluíram que os algoritmos iniciados pelo receptor apresentam maior desempenho que os algoritmos iniciados pelo servidor, como o Lowest. Contudo, em suas conclusões, o algoritmo que apresentou o maior desempenho foi o estável simetricamente iniciado. Esse algoritmo preserva um histórico

das informações de carga trocadas no sistema e toma ações de transferência de processos utilizando essas informações.

As análises dos estudos acima apresentados conduziram à proposta de um algoritmo de balanceamento de carga para ambientes distribuídos heterogêneos escaláveis, denominado TLBA (*Tree Load Balancing Algorithm*).

Durante o desenvolvimento do algoritmo TLBA, foram analisados os índices de carga existentes. Desta análise resultou um novo índice de carga baseado na ocupação de CPU e memória. Este novo índice contribuiu para o aumento de desempenho e para a estabilidade no balanceamento de carga [8].

Esse novo índice de carga foi aplicado ao algoritmo Lowest. Em seguida, o Lowest foi comparado ao algoritmo estável simetricamente iniciado, proposto por Shivaratri, Krueger e Singhal [2]. O Lowest utilizando este novo índice de carga teve aumento de desempenho, o que demonstrou que a escolha do índice influencia na qualidade das operações de balanceamento [8].

As seções a seguir apresentam a proposta do algoritmo de balanceamento de carga TLBA, projetado para ambientes distribuídos altamente escaláveis.

7.2 Objetivos

O objetivo deste projeto de doutorado é propor e avaliar o desempenho de um novo algoritmo de balanceamento de carga para sistemas distribuídos heterogêneos altamente escaláveis, denominado TLBA (*Tree Load Balancing Algorithm*) [9]. Este algoritmo foi proposto com base nos principais estudos da área [2, 15-34, 38-42].

As contribuições esperadas do algoritmo TLBA incluem o aumento de desempenho na execução de aplicações, estabilidade na distribuição de carga dos computadores que compõem o ambiente distribuído, e queda no número de mensagens que trafegam durante as operações de balanceamento de carga. As seções e os capítulos a seguir apresentam detalhes sobre o TLBA.

7.3 Modelo Formal do TLBA

O algoritmo de balanceamento de carga para ambientes distribuídos altamente escaláveis, denominado TLBA (*Tree Load Balancing Algorithm*) [9], foi projetado para executar na forma de módulos, sobre cada um dos computadores do sistema. O nome TLBA foi dado a esse algoritmo porque ele cria uma topologia lógica [71], no formato de árvore, que interconecta os computadores do sistema.

O termo árvore pode ser formalizado através de teoria de grafos. Uma árvore é definida como um grafo conexo sem ciclos. Seja $G = (X, U)$ um grafo, onde X é o conjunto de vértices e U é o conjunto de arcos que interconectam um par ordenado. Esse grafo apresenta $n > 2$, sendo n o número máximo de elementos do conjunto de vértices, e satisfaz as seguintes propriedades [72]:

1) G é conexo e sem ciclos;
2) G é sem ciclos e tem $n - 1$ arestas;
3) G é conexo e tem $n - 1$ arestas;
4) G é sem ciclos e por adição de uma aresta, somente um ciclo é criado;
5) G é conexo, mas deixa de sê-lo se uma aresta é suprimida;
6) Todo par de vértices de G é unido por uma e somente uma cadeia simples.

Após definir o termo árvore através de teoria de grafos, faz-se necessário definir os níveis que compõem uma árvore. Seja o mesmo grafo orientado $G = (X, U)$ apresentado anteriormente, é possível definir uma partição N do conjunto X tal que:

$N = \{N0, N1, ..., Nr\}$

Onde os elementos $N0, N1, ..., Nr$ são os níveis de uma árvore ordenados pela inclusão de seus $R^{-1}(xi)$ como segue:

1) $N0 = \{ xi / R^{-1} (xi) = \{\}\}$
2) $N1 = \{ xi / R^{-1} (xi)\ C\ N0\}$
3) $N2 = \{ xi / R^{-1} (xi)\ C\ (N0\ U\ N1)\}$
4) $Nr = \{ xi / R^{-1} (xi)\ C\ U\ (de\ k=0\ a\ r-1)\ Nk\}$

$R^{-1}(xi)$ representa o conjunto de vértices antecessores de um elemento xi do conjunto de vértices X. Todo vértice xi somente tem antecedentes nos níveis anteriores.

O último nível da árvore é representado por $R^{+}(Nr) = \{\}$, sendo Nr o maior nível e $R+(Nr)$ o conjunto de sucessores dos vértices do nível Nr.

O termo árvore e seus níveis são apresentados para formalizar a topologia lógica de interconexão criada pelo algoritmo de balanceamento de carga TLBA. Esse algoritmo organiza os computadores do sistema como vértices de uma árvore. Esses vértices são distribuídos pelos níveis que sempre apresentam antecessores em todo nível R^{-1} anterior. A topologia de interconexão criada pelo TLBA é dita lógica, pois não depende da topologia da física da rede de computadores [71]. Outra característica desta topologia lógica é que os níveis podem conter qualquer número de computadores e um computador xi, ou vértice, tem somente um antecessor, mas pode ter qualquer número de sucessores, representados pelo conjunto $R^{+}(xi)$. Portanto, a árvore pode ser assimétrica.

Para participar da árvore lógica de interconexões, cada computador deve negociar com um componente denominado Broker. Nesta etapa de negociação o Broker define em que nível da árvore um computador será inserido, qual seu antecessor e quais seus sucessores. Em seguida o Broker lhe envia 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.

O Broker insere computadores na árvore de acordo com suas capacidades computacionais relativas, que são utilizadas para manter a árvore balanceada. Essa capacidade é obtida através de um *benchmark* executado pelo computador durante a etapa que requer sua participação no sistema.

Cada computador participante do sistema executa um componente chamado *Peer*. Esse componente negocia com o Broker para inserir um computador na árvore e implementa as políticas de balanceamento de carga do algoritmo TLBA, as quais são detalhadas nas seções seguintes.

7.3.1 Política de Informação

A política de informação define quais os recursos utilizados para mensurar a carga dos computadores do sistema, como essas medidas de carga trafegam no ambiente para

a tomada de decisões de escalonamento e se essas medidas são geradas periodicamente ou sob demanda.

As medidas de carga são conhecidas como índice de carga. Os índices de carga permitem medir e comparar a carga de computadores, mesmo que estes apresentem constituição distinta de hardware. O índice de carga é utilizado para localizar computadores ociosos e sobrecarregados, dando suporte para a reatribuição dinâmica de processos (seção 2.2.2), que busca a distribuição eqüitativa na carga dos computadores do sistema.

O índice de carga utilizado por grande parte dos estudos de balanceamento de carga [2] é o comprimento da fila de processos em espera pela CPU. Essa técnica não funciona corretamente em ambientes onde os computadores têm capacidade de processamento heterogênea e os processos não têm ocupação semelhante [8, 9].

Para observar a limitação da técnica de índice de carga baseado no comprimento da fila de processos, suponha-se um ambiente composto de dois computadores. Em dado instante o primeiro computador apresenta dois processos de baixa ocupação em sua fila e o segundo, um processo que ocupa cerca de 99% da CPU. Nessa técnica de cálculo do índice de carga, o primeiro computador apresenta índice igual a 2 e o segundo igual a 1. O computador com menor índice recebe novos processos iniciados no ambiente [2]. Neste caso, o segundo computador irá receber o processo iniciado. Contudo, ele não tem recursos disponíveis. Enquanto isso, o primeiro computador encontra-se semiocioso.

O índice de carga baseado no comprimento da fila de processos foi analisado durante a elaboração deste projeto de doutorado. Através dessa análise foi proposta e refinada uma nova técnica de cálculo de índice de carga baseada na análise de CPU e memória, a qual apresentou melhores resultados que a primeira [8].

A nova técnica de cálculo do índice de carga, baseada na ocupação de CPU e memória, foi adotada pelo algoritmo TLBA e é detalhada a seguir.

Considere-se a ocupação de um recurso computacional $O_r(t,i)$, onde i identifica o computador analisado, r o recurso analisado (CPU ou memória principal) e t o instante de leitura da ocupação.

Considere-se a eq.(4) como a percentagem de ocupação de recurso de CPU onde, λ é o *quantum* total de cada processo no sistema, μ é o tempo relativo de λ em que a CPU manteve-se utilizada e p é o número de processos do sistema.

$$O_{cpu} = \frac{\left(\sum_{i=0}^{p} \frac{\mu_i}{\lambda_i} \right)}{p} \tag{4}$$

Considere-se a eq.(5) como 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.

$$O_{mem} = \frac{\left(\sum_{i=0}^{p} \sigma_i \right)}{\theta} \tag{5}$$

O índice de carga (I_{comp}) é definido pela eq.(6), onde O_{CPU} é a ocupação percentual do recurso de CPU e O_{MEM} a ocupação percentual do recurso de memória principal.

$$I_{comp} = \frac{(O_{cpu} + O_{mem})}{2} \tag{6}$$

Cada computador participante da árvore do sistema executa um componente denominado *Peer* que, periodicamente, analisa a ocupação do recurso de CPU e memória principal e em seguida calcula o índice de carga (I_{comp}). Esse evento periódico tem intervalos definidos pelo administrador do sistema.

Os índices de carga calculados por um computador α localizado no último nível *Nk* de uma árvore, são propagados para seu antecessor β localizado no nível *Nk-1*. O computador β armazena em memória seu índice de carga e de seus sucessores. O computador β, 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, atualizando as informações do sistema.

Deve-se notar que computadores que apresentam sucessores na árvore submetem o somatório de seu índice e dos índices de carga de seus sucessores ao seu antecessor. Portanto, um computador β, que apresenta sucessores, submete um índice para seu antecessor, que resume sua carga e de seus sucessores. Esses índices, propagados até a raiz da árvore, são utilizados pela política de localização (seção 7.3.2) do algoritmo de balanceamento de carga TLBA.

Essa técnica periódica de geração e atualização de informação de carga pode sobrecarregar o meio de comunicação através do grande número de mensagens [2]. Para evitar essa sobrecarga, a política de informação do algoritmo TLBA define que um computador somente envia o índice de carga para seu antecessor quando ocorrem mudanças significativas na ocupação dos recursos. Essa técnica garante a estabilidade do algoritmo [2].

Para analisar mudanças significativas em sua carga, um computador compara seu índice de carga anterior (I_{comp-1}) ao atual (I_{comp}), conforme descrito nas eq.(7) e eq.(8) onde: δ, conhecido como *threshold*, é o percentual adicionado ou removido de I_{comp-1} para analisar a alteração no índice de carga atual (I_{comp}), V_s é a variação superior no índice de carga e V_i é a variação inferior no índice de carga.

$$V_s = I_{comp-1} + \left(I_{comp-1} * \delta\right) \tag{7}$$

$$V_i = I_{comp-1} - \left(I_{comp-1} * \delta\right) \tag{8}$$

A alteração do índice de carga atual I_{comp} em relação ao índice anterior I_{comp-1} é confirmada através da estrutura condicional apresentada na eq.(9). Confirmada a mudança significativa na ocupação, o índice de carga atualizado é enviado para o computador antecessor na árvore. O δ das eq.(7) e eq.(8) influenciam na variação percentual mínima da eq.(9) para que um índice de carga I_{comp} seja considerado alterado em relação ao seu anterior I_{comp-1}.

Tão logo o índice tenha sido enviado para o computador antecessor, ele torna-se I_{comp-1} e será utilizado nas próximas comparações de carga. Quando a mudança de ocupação não é confirmada, o índice I_{comp} é descartado e I_{comp-1} continua com o mesmo valor. Desta forma, fica garantido que um antecessor sempre seja atualizado em

circunstâncias de mudança de carga. Esse valor é utilizado pela política de localização (seção 7.3.2).

Um computador que apresenta sucessores submete para seu antecessor um índice de carga que unifica seu índice de carga ao de seus sucessores. Computadores desse tipo submetem esse índice unificado em duas circunstâncias: quando recebe um índice I_{comp} atualizado de um sucessor e quando comprova alteração de seu índice I_{comp}. Um computador que não tem sucessores somente submete seu índice I_{comp}.

$$se \quad I_{comp} > V_s \quad ou \quad I_{comp} < V_i \tag{9}$$

Sabe-se que o índice de carga I_{comp} de um computador α representa sua ocupação percentual. Portanto, varia de 0 a 100% ocupado. O complemento desse índice é a capacidade ociosa de cada computador, definida como $!I_{comp}$ e apresentada na eq.(10). Quando um computador apresenta sucessores, o cálculo torna-se diferente, conforme descrito a seguir.

$$!I_{comp} = 100 - I_{comp} \tag{10}$$

Todo computador α que apresenta sucessores envia para seu antecessor β seu índice de carga I_{comp} e um valor τ, que representa o número total de computadores presentes na subárvore de α. Este número é calculado realizando o somatório de computadores que sucedem α mais uma unidade, que representa o próprio computador α. O computador β tem disponível o índice de carga I_{comp} de seu sucessor α, também definido como $I_{comp\text{-}\alpha}$, além do valor τ. Assim sendo, β pode definir o complemento do índice de carga I_{comp} para toda a subárvore a partir de seu sucessor α através da eq.(11), onde: $!I_{comp\text{-}\alpha}$ representa o complemento do índice de carga de todos os computadores presentes na subárvore, ou ramo, a partir de α; $I_{comp\text{-}\alpha}$ representa o somatório dos índices de carga de todos os computadores presentes na subárvore de α.

$$!I_{comp-\alpha} = (\tau * 100) - I_{comp-\alpha} \tag{11}$$

O complemento do índice de carga de um computador α que apresenta sucessores reflete a capacidade ociosa de todos os computadores contidos na subárvore de α. A

subárvore de α compreende o computador α, seus sucessores do nível Nk, Nk+1, Nk+2 e os seguintes, até não haver mais sucessores.

O índice de carga I_{comp}, apresentado nesta seção, representa a ocupação dos recursos de cada computador do ambiente. Tendo acesso a esses índices, pode-se localizar computadores ociosos e sobrecarregados e, conseqüentemente, balancear a carga do ambiente. O algoritmo TLBA utiliza esses índices para localizar computadores ociosos e realizar transferências de processos que distribuem a carga pelo ambiente, balanceando-o.

7.3.2 Política de Localização

A política de localização de um algoritmo de balanceamento de carga busca encontrar os computadores receptores e emissores ou servidores de processos. Esses computadores sincronizam-se para transferir carga e balancear o sistema (política de transferência - seção 7.3.3). O melhor emissor de processos é definido como o computador mais sobrecarregado do sistema. O melhor receptor, como o computador mais ocioso do sistema.

No algoritmo TLBA, a política de localização busca somente por computadores receptores de processos, que recebem novos processos iniciados no ambiente, ou parte da carga de um computador emissor de processos. A localização do melhor receptor é ativada em duas circunstâncias: quando uma requisição para executar um novo processo é submetida ao Broker; quando um computador sobrecarregado pretende transferir parte de sua carga.

Para localizar o melhor computador receptor de processos, ou seja, o computador mais ocioso, o TLBA inicia uma busca em profundidade, analisando os complementos dos índices de carga ($!I_{comp}$) dos computadores (eq.(11)). Essas informações são mantidas na memória principal, o que permite acesso em alta velocidade, colaborando para o desempenho das buscas.

Toda operação de busca é iniciada no computador raiz, localizado no primeiro nível da árvore, e propagada até encontrar o computador mais ocioso. Para compreender essas buscas considere-se um computador α, o qual compara o complemento de seu índice de carga ao complemento dos índices de seus sucessores. Caso um sucessor tenha

maior complemento, ou seja, menor carga, este recebe uma mensagem para propagar a busca. Caso contrário, o computador α é definido como o mais ocioso do sistema.

A política de localização do algoritmo TLBA privilegia localizar computadores mais próximos das folhas, ou seja, computadores presentes no último nível *Nr* da árvore. Essa característica da política de localização é notada através da eq.(11), onde o somatório de todos os índices de carga dos computadores da subárvore de um computador α é subtraído da capacidade total desses computadores. Desta forma, em situações de baixa carga no ambiente, as mensagens de busca em profundidade tendem a percorrer todos os níveis e localizar computadores receptores próximos do nível *Nr*. 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.

Uma vez localizado o computador mais ocioso do ambiente, definido como o melhor receptor de processos, seu endereço é enviado para um dos dois destinos: primeiro destino, o Broker, caso este tenha recebido uma requisição para executar um novo processo no ambiente e tenha ativado a política de localização. Nesse caso, o Broker transfere o novo processo para o receptor. Segundo destino, o computador sobrecarregado, caso este tenha ativado a política de localização para transferir parte de sua carga (veja política de transferência – seção 7.3.3).

7.3.3 Política de Transferência

A política de transferência é responsável por definir quais computadores participam da transferência de um processo e como ela é realizada. Nessa política há a abordagem do computador emissor e a do computador receptor de processos.

No algoritmo TLBA a política de transferência pode ser ativada em duas circunstâncias: a primeira é quando uma requisição, para executar um processo, chega ao Broker; a segunda é 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 (seção 7.3.2) 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 o receptor, 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, que é representado na eq.(11), em que $!I_{comp\text{-}médio\text{-}\alpha}$ representa o complemento do índice de carga de todos os computadores presentes na subárvore ou ramo, a partir de α, sendo α neste caso a raiz; $I_{comp\text{-}\alpha}$ representa o somatório dos índices de carga de todos os computadores presentes na subárvore de α; e τ representa o número de computadores presentes na subárvore de α. τ é calculado através do somatório de computadores que sucedem α mais uma unidade, que representa o próprio computador α.

O complemento médio do índice de carga de um computador α reflete a média de capacidade ociosa dos computadores contidos na subárvore de α. Nessa segunda circunstância da política de transferência, α é definido como o computador raiz e, portanto, $!I_{comp\text{-}médio\text{-}\alpha}$ retorna a média percentual livre por computador contido na árvore.

Após obter o complemento médio do índice de carga do computador raiz ($!I_{comp\text{-}médio\text{-}\alpha}$, sendo α a raiz), o computador sobrecarregado obtém o limite superior do complemento médio do índice de carga do computador raiz (L_{raiz}), aplicando a eq.(12), em que $!I_{comp\text{-}médio\text{-}\alpha}$ representa o complemento médio da subárvore de α; δt, conhecido como *threshold* de transferência, é o percentual, configurável, utilizado para analisar a viabilidade de transferência de processos de um computador sobrecarregado para outro, ocioso.

$$L_{\alpha} = !I_{comp-médio-\alpha} - \left(!I_{comp-médio-\alpha} * \delta t\right) \tag{12}$$

O computador sobrecarregado utiliza L_{raiz}, obtido na eq.(12), em que α é o computador raiz, para avaliar se a capacidade média ociosa da árvore torna viável uma transferência de processo. Essa avaliação é feita por meio da condição apresentada na eq.(13). Se essa condição for satisfeita, sabe-se que a capacidade média livre dos computadores do ambiente é maior que a capacidade livre do computador

sobrecarregado e, portanto, a transferência torna-se benéfica ao processo e ao desempenho global do ambiente.

$$!I_{comp-sobrecarregado} < L_{raiz} \tag{13}$$

Satisfazendo-se a condição apresentada na eq.(13), dá-se continuidade à política de transferência. Caso contrário, ela é cancelada. Em caso de continuidade, essa política ativa a política de localização para encontrar um receptor de processos (seção 7.3.2). O endereço desse receptor de processos é enviado para o computador sobrecarregado, também conhecido como emissor ou servidor de processos, que inicia a política de seleção (seção 7.3.4) para localizar seu processo de maior ocupação. O contexto desse processo é salvo, etapa conhecida como *checkpointing*, e transferido para o receptor que o reinicia.

Outra característica da política de transferência do TLBA é que um processo não pode ser transferido mais de κ vezes no ambiente. Para compreender melhor o uso deste valor κ, considere-se uma situação onde todos os computadores do ambiente apresentam a mesma capacidade computacional e encontram-se 100% ociosos. Um processo que ocupa quase a totalidade, ou seja, 100% dos recursos de CPU e memória de um computador, é submetido a esse ambiente. O computador que receber este processo deve detectar seu estado de sobrecarga e iniciar a política de transferência. Essa política irá comprovar que o ambiente apresenta capacidade média livre maior que a do computador sobrecarregado, dando continuidade e transferindo o processo. O computador que receber esse processo irá iniciar, novamente, a política de transferência e assim consecutivamente. O custo de transferir este processo não agrega benefícios de desempenho em sua execução. Essa solução do número máximo de transferências por processo é apresentada e discutida por Shivaratri, Krueger e Singhal [2], que comprovam as contribuições dessa técnica para a estabilidade de um algoritmo de balanceamento de carga.

7.3.4 Política de Seleção

A política de seleção de um algoritmo de balanceamento de carga é responsável por selecionar processos para transferência (seção 7.3.3). No algoritmo TLBA essa política busca pelo processo que ocupa maior quantidade dos recursos de CPU e

memória em um computador sobrecarregado α, conforme apresentado na eq.(14), em que μ é o tempo relativo de λ em que a CPU manteve-se utilizada pelo processo p; λ é o *quantum* total do processo p; σ é a quantidade de memória principal utilizada por um processo p; θ é a quantidade total de memória principal; e p é o identificador do processo, que varia do primeiro ao último processo que executam no computador α.

$$O_{processo-maior-ocupação-\alpha} = MAX\left(\frac{\frac{\mu_p}{\lambda_p} + \frac{\sigma_p}{\theta_\alpha}}{2}\right) \tag{14}$$

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 (seção 7.3.3).

A transferência de processos envolve as etapas de localização do computador mais ocioso, seleção do processo, *checkpointing* e a transferência propriamente dita. Estas etapas apresentam custo computacional considerável. Este custo somente é bem aproveitado através da transferência de processos de maior ocupação.

7.3.5 Outras Considerações sobre o Algoritmo TLBA

As seções anteriores apresentaram as políticas que compõem o algoritmo de balanceamento de carga TLBA. Além dessas políticas, existem outras considerações relevantes sobre esse algoritmo que devem ser apresentadas. Essas considerações compreendem: a alta disponibilidade, o balanceamento na montagem da árvore e a topologia física de interconexão dos computadores.

Quanto à alta disponibilidade, o algoritmo TLBA tem duas técnicas: a primeira, para tratar a falha de computadores participantes do sistema; a segunda, para tratar a falha de um Broker.

A falha dos computadores participantes do sistema pode ser detectada em duas circunstâncias: na primeira, quando um computador antecessor, utilizando a política de localização (seção 7.3.2), não consegue se comunicar com um de seus sucessores, pois este apresenta falha; na segunda, quando um computador, utilizando a política de

informação (seção 7.3.1), tenta comunicar-se com seu antecessor, o qual apresenta falha. Em ambos os casos o computador que detectou a falha submete uma mensagem ao Broker, que reorganiza a árvore do sistema.

Para evitar falhas, o Broker pode ser replicado, mantendo a consistência nas informações da árvore de conexões tanto em disco quanto memória principal. Para manter a consistência, faz-se uso de um algoritmo de eleição para acessar uma região crítica [13]. Essa região crítica contém o código correspondente às operações de inserção de participantes e reorganização da árvore. Quando diversos Brokers desejam acessar a mesma região, eles disputam entre si.

Qualquer Broker pode receber requisições para reorganizar ou inserir processos no sistema, mesmo se outro Broker estiver acessando a região crítica. Nesse caso, a requisição entra numa fila e aguarda a liberação da região crítica, que é novamente disputada. Os Brokers, periodicamente, realizam testes entre si para analisar se continuam ativos. Caso um Broker falhe, este é automaticamente retirado do sistema. Se voltar à atividade, ele se conecta aos demais e sincroniza suas informações da árvore.

A falha de um Broker que travou acessando a região crítica, é detectada pelos demais. Nesse caso, um Broker v que fez a detecção, realiza uma busca em profundidade, analisando e atualizando suas informações sobre a estrutura da árvore. Após concluir essa busca, as informações atualizadas são enviadas para todos os Brokers, e v libera a região crítica.

Ambientes compostos por um único Broker também contêm algum nível de contenção às falhas. As informações estruturais da árvore são mantidas na memória principal e no disco do Broker. Caso o Broker falhe, ele pode reiniciar sua participação no sistema e recuperar do disco físico o estado estrutural da árvore. Durante o período em que o sistema não tem um Broker disponível, não se pode submeter processos ao sistema, inserir novos participantes na árvore, nem reorganizar a estrutura da árvore quando na falha de um participante.

A próxima consideração relevante do TLBA é em relação ao balanceamento e alocação dos computadores na árvore montada pelos Brokers. Antes de inserir um novo computador na árvore, o Broker lhe requisita a execução de um *benchmark*. O resultado do *benchmark* reflete a capacidade de CPU e memória principal.

O resultado do *benchmark* é utilizado pelo Broker para inserir, de forma balanceada, um computador na árvore. Nesse balanceamento estrutural da árvore, o Broker realiza uma busca em profundidade passando pelos níveis que apresentam menor capacidade computacional. Após chegar nas folhas, o Broker analisa se esse nível *Nr* está preenchido com seu número máximo de computadores. Caso esteja, o Broker sobe um nível, ou seja, dirige-se para o *Nr-1* e faz o mesmo teste. Ao encontrar um nível disponível, o Broker insere o participante nesse local. Se o Broker chegar até o nível N0, onde fica a raiz da árvore, e todos os níveis estiverem preenchidos, ele volta até o nível *Nr*, seleciona randomicamente um computador desse nível e insere o novo participante como seu sucessor.

Alguns detalhes são relevantes na estrutura da árvore. O primeiro é que o número máximo de computadores por nível da árvore é configurável. O segundo é que após receber o resultado do *benchmark*, todas as operações realizadas pelo Broker para balancear a estrutura da árvore, são realizadas com informações locais e disponíveis na memória principal, sem criar conexões com os computadores da árvore.

Outra consideração relevante é que a topologia lógica criada pelo algoritmo TLBA independe da topologia física da rede dos computadores que participam do sistema. Contudo, há uma ressalva para redes geograficamente separadas que apresentam baixa largura de banda de comunicação. Essa ressalva se refere à definição de intervalos de tempo mais longos para atualização periódica de informações entre níveis da árvore (seção 7.3.1), evitando sobrecarga no meio de comunicação.

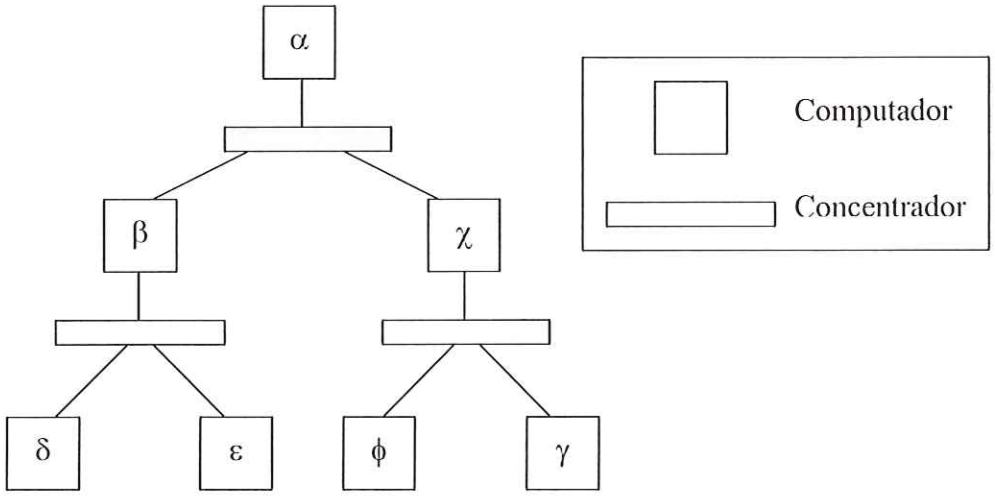

FIGURA 2 – Exemplo de Topologia Física de uma Rede

Apesar da topologia lógica ser independente da física, existe um benefício em projetar a parte física tal como a lógica [9, 32]. Considere-se, por exemplo, a figura 2, onde há três redes distintas: a primeira interconecta os computadores α, β e χ; a segunda interconecta β, δ e ε; a terceira interconecta χ, ϕ e γ. Nesse caso, recomenda-se a aplicação da organização lógica sobre a física, onde α representa o computador raiz, presente no nível 0; β e χ correspondem aos computadores do nível 1; δ, ε, ϕ e γ correspondem aos computadores do nível 3.

Em casos como o anterior, recomenda-se que a topologia lógica siga a física, pois gera menor sobrecarga no meio de comunicação. A menor sobrecarga se deve à distribuição de mensagens, originadas pelas políticas do algoritmo TLBA, entre 3 diferentes concentradores. Contudo, há uma limitação nessa abordagem, que não permite o uso de um Broker para reorganizar e balancear a estrutura da árvore de computadores participantes. Portanto, ela deve ser configurada manualmente.

7.4 Contribuições

Este projeto de doutorado contribuiu com a proposta e avaliação do algoritmo de balanceamento de carga TLBA (*Tree Load Balancing Algorithm*) [9]. Esse algoritmo apresentou aumento no desempenho das aplicações, que executam em ambientes distribuídos escaláveis. Esse ganho de desempenho foi observado por meio de simulações (capítulo 9) e confirmado nos testes realizados com um protótipo (capítulo 10).

7.5 Resumo do Capítulo

Este capítulo apresentou a proposta deste projeto de doutorado que consiste no desenvolvimento de um algoritmo de balanceamento de carga denominado TLBA (*Tree Load Balancing Algorithm*). Essa denominação deve-se ao fato do algoritmo criar uma topologia de interconexão lógica entre os computadores no formato de uma árvore.

O algoritmo TLBA é indicado para ambientes distribuídos homogêneos e heterogêneos altamente escaláveis. Simulações e testes realizados, usando-se um protótipo, permitiram comprovar as contribuições desse algoritmo. Essas contribuições

são referentes ao aumento de desempenho na execução de processos e queda no número de mensagens geradas pelas operações de balanceamento de carga.

8. CONSTRUÇÃO DE UM EMULADOR DO TLBA

8.1 Considerações Iniciais

Para compreender e visualizar de forma mais simples o funcionamento de cada etapa do algoritmo de balanceamento de carga TLBA, seria necessário construir um sistema real ou algo que emulasse seu comportamento. Contudo, isso seria inviável durante sua proposta. Nessa fase foi construído um emulador, que colaborou no projeto e em refinamentos do TLBA. Esse emulador é apresentado a seguir.

8.2 Objetivos do Emulador do TLBA

O emulador (código-fonte no Apêndice D) foi construído com o objetivo de estudar, compreender e refinar o funcionamento do algoritmo de balanceamento de carga TLBA. Esse emulador permitiu validar idéias que surgiram durante a definição do algoritmo, refinando, conseqüentemente, sua construção.

8.3 Implementação do Emulador

A implementação do emulador foi realizada obedecendo-se às políticas de informação, localização, transferência e seleção do algoritmo TLBA (capítulo 7). Esse emulador foi escrito na linguagem Java [84, 86] utilizando *threads*, permitindo representar o funcionamento real de cada parte do sistema com seus paralelismos inerentes. É composto pelas seguintes classes:

1) Broker - classe que inicia a execução do emulador e que implementa os serviços para realizar buscas pelo melhor computador do sistema. O melhor computador no sistema é sempre o mais ocioso (veja política de informação – seção 7.3.1).

2) Node - classe que representa um computador no sistema. A classe Node contém uma instância da classe Scheduler. Node faz as leituras de carga do computador que representa. Caso sua carga tenha variado significantemente ou tenha recebido atualizações de carga de seus sucessores, Node gera mensagens de atualização para seu antecessor na árvore. Se a carga estiver muito alta, essa classe gera mensagens para a migração de processos. Node é implementada como uma *thread* Java.
3) Scheduler – cada instância desta classe representa o escalonador de um computador do sistema. Nessa classe são implementados os chaveamentos de contexto, que assim como no Linux (sistema operacional escolhido como base para os estudos e aplicado nos testes do protótipo), ocorrem usando a técnica round-robin por prioridades [73]. O escalonador trata os processos que estão na fila para execução. As informações desses processos são utilizadas pela classe Node para calcular índices de carga (seção 7.3.1). A classe Scheduler é implementada como uma *thread*.
4) Process - classe que representa os processos submetidos ao sistema. Essa classe contém os parâmetros de consumo do processo, instante de chegada, início de atendimento e saída do sistema.
5) RandomPoissonDistribution - classe que gera uma função de distribuição de probabilidade Poisson para a chegada de processos ao sistema. Essa função é adotada pelos principais trabalhos na área, pois reflete a ocupação do sistema em situações aonde um evento de chegada não tem ligação com outro [2, 68].
6) RandomSample - classe que gera valores randomicamente distribuídos. Utilizada para definir o consumo de CPU e memória de cada processo.

A implementação desse emulador em Java foi feita para compreender as etapas do algoritmo TLBA e refinar seu funcionamento. As classes que representam os computadores e escalonadores são *threads*, assim como o gerador de processos para o ambiente. O uso de *threads* permitiu definir um sistema de balanceamento de carga muito próximo do real, onde ações são paralelas.

8.4 Resumo do Capítulo

O emulador permitiu acompanhar o funcionamento detalhado do algoritmo TLBA, conseqüentemente validando e colaborando na implementação e na inserção de pequenas alterações no algoritmo até que fosse totalmente projetado. O código fonte e modelagem desse emulador estão presentes no apêndice D.

Esse emulador pode ser utilizado por todos aqueles interessados em compreender o funcionamento do algoritmo TLBA sem a necessidade de implementá-lo como um sistema real.

9. CONSTRUÇÃO E AVALIAÇÃO DOS RESULTADOS OBTIDOS DE UM SIMULADOR DO ALGORITMO TLBA

9.1 Considerações Iniciais

O emulador, apresentado no capítulo 8, permitiu refinar e detalhar as políticas do algoritmo TLBA. Por outro lado, não permitiu avaliar seu desempenho. Com esse propósito, foi construído um simulador. Os resultados desse simulador são comparados com os do simulador do algoritmo Lowest modificado [9], ambos implementados neste projeto.

Segundo os estudos de Mello, Trevelin e Paiva [8], o algoritmo Lowest modificado apresentou maior desempenho que o algoritmo estável simetricamente iniciado, proposto por Shivaratri, Krueger e Singhal [2] como a melhor solução de balanceamento de carga. Por esta razão o algoritmo Lowest modificado foi adotado como base comparativa para o algoritmo TLBA.

As seções a seguir apresentam os resultados das simulações e análises comparativas dos algoritmos TLBA e Lowest modificado [9]. Nessas seções o Lowest modificado é muitas vezes referenciado apenas como Lowest.

9.2 Objetivos da Simulação

O objetivo da simulação do algoritmo TLBA é comprovar seu desempenho e viabilidade de implementação para o balanceamento de carga de ambiente real. Para isso, os resultados desse simulador (código-fonte no Apêndice D) são comparados aos

resultados obtidos pelo simulador do algoritmo Lowest modificado, proposto por Mello, Trevelin e Paiva [8].

9.3 Métricas de Análise

A simulação tem por objetivo avaliar um sistema ou modelo proposto. Contudo, é necessário definir os tipos de saídas que são geradas pela simulação. Essas saídas compreendem os resultados, que posteriormente são analisados. As análises permitem concluir sobre o comportamento do sistema ou modelo.

A definição dos tipos de resultados gerados por um simulador influencia em sua construção. Não é necessário que um simulador implemente todos os módulos de um sistema para que gere resultados consistentes. Contudo, o simulador deve implementar todos os módulos do sistema que influenciam na geração dos resultados esperados [61-66].

Para definir os resultados esperados de um simulador deve-se conhecer sobre seus eventos. Há métricas adequadas para cada evento [76]. O evento estudado neste projeto de doutorado é o balanceamento de carga. 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 [1, 2, 15-42, 76].

O tempo médio de resposta dos processos influencia no 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 [2, 16].

O número de mensagens trocadas nas operações de balanceamento de carga não pode ser alto, pois sobrecarrega o meio de comunicação, prejudicando, conseqüentemente, o desempenho dos processos que se comunicam. Há, por exemplo, 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 [2, 16].

Este projeto de doutorado, como também os principais trabalhos da área de balanceamento de carga [1, 2, 15-42], considera como resultado das simulações o tempo

médio de resposta dos processos e o número de mensagens geradas nas operações de balanceamento de carga.

9.4 Considerações sobre os Simuladores do Algoritmo TLBA e Lowest Modificado

Os simuladores do algoritmo TLBA e Lowest modificado foram construídos utilizando-se modelos de redes de filas [61-66, 70, 76]. Neste modelo, cada computador representa um servidor com sua fila, onde os processos aguardam atendimento.

9.5 Propostas de Avaliação usando Simuladores

Para analisar o algoritmo TLBA foram propostas diversas situações de simulação. Essas simulações buscam analisar o comportamento do algoritmo sob diferentes cargas de processos, número de computadores por nível da árvore, número de níveis, ambiente de hardware homogêneo e heterogêneo. As mesmas situações foram simuladas para o algoritmo Lowest modificado, permitindo compará-los.

As simulações propostas para os dois algoritmos são subdivididas nos seguintes grupos:

1) Simulações para analisar o desempenho do algoritmo TLBA em um pequeno sistema homogêneo.
2) Simulações para analisar o desempenho do algoritmo TLBA em um pequeno sistema heterogêneo.
3) Simulações para analisar o desempenho do algoritmo TLBA em um grande sistema homogêneo.
4) Simulações para analisar o desempenho do algoritmo TLBA em um grande sistema heterogêneo.
5) Simulações para analisar o número de mensagens trocadas nas operações de balanceamento de carga.

As seções a seguir apresentam estes grupos de simulações, seus resultados e suas respectivas análises.

9.6 Simulações para Analisar o Desempenho do TLBA em um Pequeno Sistema Homogêneo

As simulações que seguem visam analisar o desempenho do algoritmo de balanceamento de carga TLBA comparando-o ao Lowest modificado. Estes estudos foram realizados para pequenos ambientes compostos de hardware de capacidade de processamento homogênea. Todos os resultados apresentados compreendem a média de 100 simulações sucessivas.

9.6.1 Política de Localização do Algoritmo Lowest Modificado

O algoritmo Lowest tem sua política de localização (seção 7.3.2) baseada na seleção randômica de um subconjunto de computadores, requisição da carga destes computadores e, posteriormente, análise para detectar o computador mais ocioso, que recebe processos iniciados no ambiente. Essas operações ocorrem para cada processo iniciado no ambiente.

O tamanho dos subconjuntos de computadores analisados pelo Lowest é um parâmetro que influencia diretamente na qualidade da política de localização, ou seja, na escolha de um computador ocioso [8, 16]. Essa técnica utilizada pelo Lowest não garante que o computador mais ocioso do sistema seja escolhido.

O tamanho dos subconjuntos de computadores analisados pela política de localização do algoritmo Lowest é modificado de acordo com a escala do sistema, sendo proporcional à mesma. Sistemas formados por poucos computadores tendem a ter pequenos subconjuntos. Grandes sistemas apresentam subconjuntos maiores. Contudo, o tamanho desses subconjuntos não pode ser grande demais, pois gera muitas mensagens no meio de comunicação, durante as operações de balanceamento de carga.

O sistema analisado nas seções a seguir é formado por sete computadores. Para simular o Lowest em seu desempenho máximo, é necessário definir o tamanho de seu subconjunto de computadores. Esta seção apresenta um estudo que resulta no tamanho deste subconjunto.

Antes de definir o tamanho do subconjunto de computadores do Lowest, são descritas as características da simulação. Essa simulação utiliza sete computadores. A capacidade de processamento de cada computador do sistema é de 368 ciclos por milisegundo, número que reflete a capacidade de um computador com um processador AMD Duron de 950 Mhz com 256 Mb de memória RAM (maiores detalhes no apêndice A). Os processos chegam ao sistema através de uma função Poisson de distribuição de probabilidades, com média 3 segundos. Foram submetidos 320 processos ao sistema, cada um com ocupação de 10330000 ciclos.

Foi atribuído o valor de 368 ciclos por milisegundo para cada computador simulado, pois durante o projeto havia uma rede de computadores disponível para testes que apresentava a mesma capacidade de processamento por computador. A escolha desse valor foi baseada na execução exaustiva de um programa em cada computador. Esse mesmo programa foi utilizado para testar o protótipo do TLBA (seção 10.5). Após essas execuções, a média encontrada foi de 368 ciclos para computadores com tais características.

A função de distribuição de probabilidades Poisson para a chegada de processos, é adotada pelos principais trabalhos da área de balanceamento de carga [1, 2, 16, 34] e, portanto, foi adotada neste trabalho.

O termo variável, nas simulações apresentadas nesta seção, foi o tamanho do subconjunto de computadores analisados pela política de localização do algoritmo Lowest. O tamanho do subconjunto variou do número mínimo de um computador, até o máximo de seis computadores analisados. Os resultados dessas simulações são apresentados na tabela 2.

A tabela 2 comprova que há queda significativa nos tempos médios de resposta dos processos, até o número de 3 computadores por subconjunto. Subconjuntos maiores apresentam flutuação nos tempos médios de resposta e, como se pode observar na tabela 2, apresentam alto número de mensagens trocadas no meio.

TABELA 2 – Simulações para definir o número mais indicado de computadores analisados pela política de localização do algoritmo Lowest

Número de Processos	Ocupação por Processo (ciclos)	Poisson (segs)	Capacidade de Processamento por Computador (ciclos/ms)	Número de Computadores Analisados	Tempo Médio de Resposta por Processo (ms)	Número de Mensagens geradas
320	1,033E+07	3	368	1	1,0370895630E+06	960
320	1,033E+07	3	368	2	7,4511096880E+05	1600
320	1,033E+07	3	368	3	6,6255829220E+05	2240
320	1,033E+07	3	368	4	6,7799078270E+05	2880
320	1,033E+07	3	368	5	6,6186491220E+05	3520
320	1,033E+07	3	368	6	6,5694759970E+05	4160

Quanto maior o subconjunto de computadores analisados pelo Lowest, maior a probabilidade de encontrar o computador mais ocioso do sistema. Contudo, quanto maior o subconjunto, maior o número de mensagens por operação de balanceamento de carga, pois são selecionados mais computadores, obtidas suas cargas e estas analisadas para definir qual o computador que deve receber processos no ambiente.

A etapa de obtenção da carga dos computadores envolve a troca de duas mensagens por computador do subconjunto. Uma mensagem para requisitar a ocupação e outra de retorno que contém esta informação.

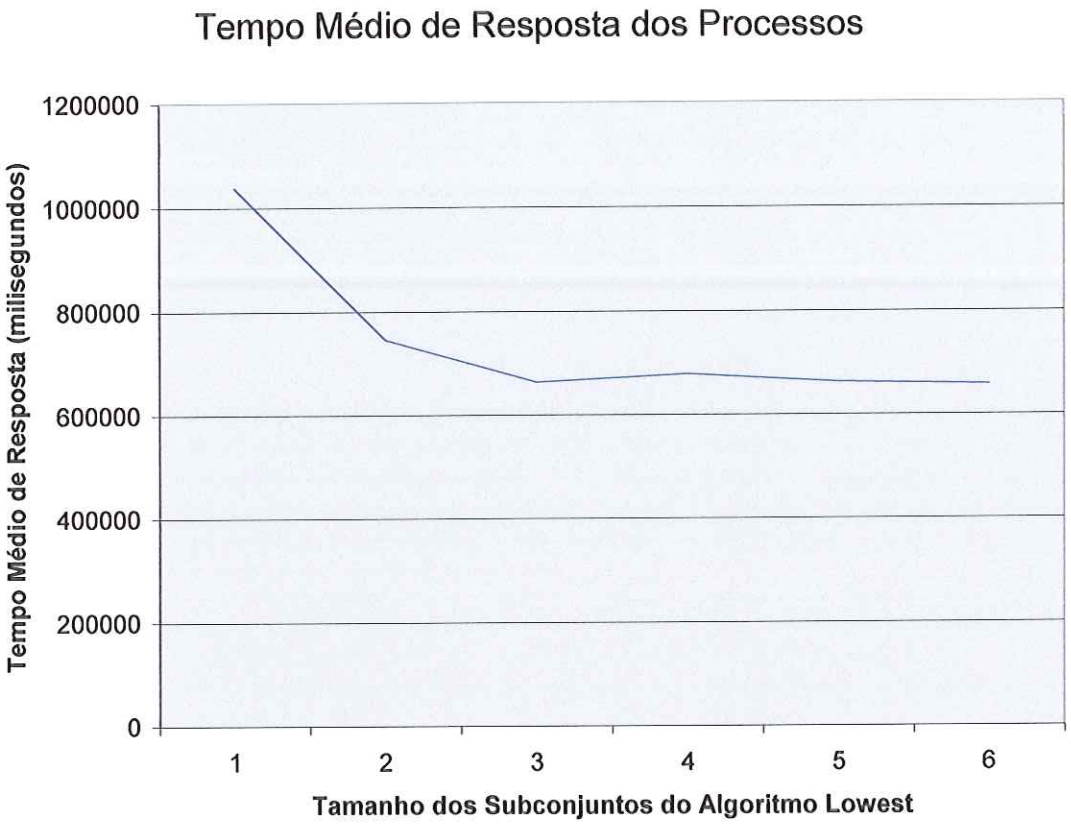

FIGURA 3 - Tempo médio de resposta dos processos em função do tamanho do subconjunto de computadores analisados pela política de localização do algoritmo Lowest

As figuras 3 e 4 apresentam gráficos que simplificam a visualização dos dados das simulações. Pelo gráfico da figura 3, pode-se notar a queda acentuada no tempo médio de resposta dos processos para subconjuntos compostos por 3 computadores. Acima desse número, a queda no tempo de resposta é muito pequena. No gráfico da figura 4, pode-se notar o aumento expressivo no número de mensagens trocadas no meio de comunicação para cada operação de balanceamento de carga.

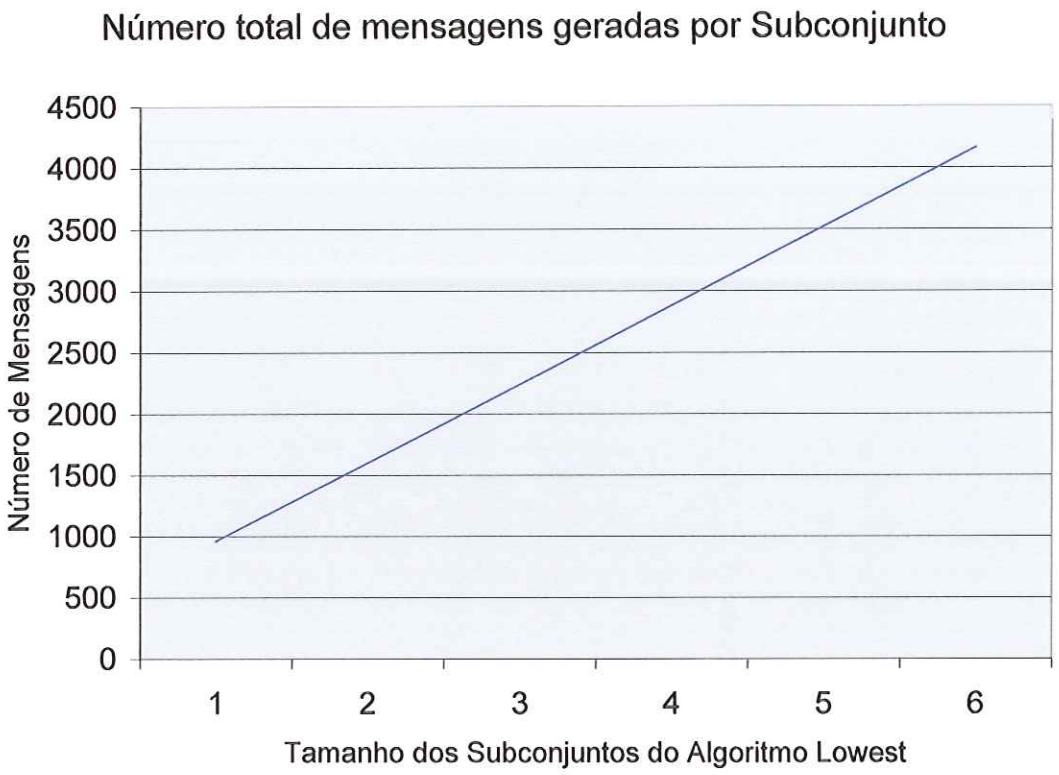

FIGURA 4 - Número total de mensagens geradas no meio de comunicação, para diferentes tamanhos de subconjunto de computadores

Levando-se em consideração os resultados apresentados, optou-se por adotar 3 computadores para os subconjuntos analisados pela política de localização do algoritmo Lowest em sistemas compostos por 7 computadores. Esse tamanho de subconjunto é adotado nas próximas seções, onde são apresentadas simulações do Lowest.

9.6.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado

Esta seção apresenta e analisa resultados de simulações do algoritmo TLBA e Lowest modificado. As simulações foram realizadas sobre um ambiente composto por 7 computadores de capacidade de processamento homogênea. Cada computador tem a capacidade de 368 ciclos por milisegundo. Os processos chegam ao sistema através de uma função Poisson de distribuição de probabilidades com média 100 segundos. Foram

submetidos 320 processos ao sistema. Essa ocupação sempre manteve o ambiente sob alta carga.

O algoritmo Lowest modificado utilizou subconjuntos compostos por 3 computadores para sua política de localização. O algoritmo TLBA criou uma topologia lógica de interconexões na forma de árvore, contendo dois computadores por nível, totalizando três níveis.

A tabela 3 apresenta os tempos médios de resposta resultantes das simulações do algoritmo TLBA e Lowest modificado.

TABELA 3 – Tempos médios de resposta do algoritmo TLBA e Lowest modificado em ambientes compostos por 7 computadores de capacidade de processamento homogênea

Número de Processos	Ocupação por Processo (ciclos)	Poisson média (segs)	Subconjunto de Computadores Analisados pelo Lowest	Capacidade de Processamento por Computador (ciclos/ms)	Lowest Modificado Tempo Médio de Resposta dos Processos (ms)	TLBA Tempo Médio de Resposta dos Processos (ms)
320	1,00E+07	100	3	368	6,1275533000E+05	6,0450143000E+05
320	1,00E+08	100	3	368	6,3940207300E+06	6,3176135300E+06
320	1,00E+09	100	3	368	6,4152160200E+07	6,3446532460E+07
320	1,00E+10	100	3	368	6,4098737963E+08	6,3473560175E+08
320	1,00E+11	100	3	368	6,4189585691E+09	6,3476263532E+09
320	1,00E+12	100	3	368	6,5020352364E+10	6,3476532508E+10
320	1,00E+13	100	3	368	6,4361410067E+11	6,3476559492E+11
320	1,00E+14	100	3	368	6,4261208947E+12	6,3476562200E+12
320	1,00E+15	100	3	368	6,4770720080E+13	6,3476562470E+13
320	1,00E+16	100	3	368	6,4230638584E+14	6,3476562497E+14
320	1,00E+17	100	3	368	6,5764266304E+15	6,3476562500E+15
320	1,00E+18	100	3	368	6,4104959239E+16	6,3476562500E+16

Analisando os dados apresentados na tabela 3, observa-se que os tempos médios de resposta obtidos na simulação do algoritmo TLBA são menores que do Lowest modificado. Pode-se concluir, portanto, que em pequenos sistemas homogêneos com alta carga, o algoritmo TLBA apresenta melhores resultados.

9.6.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado

Foram realizadas simulações para avaliar a precisão nos tempos de resposta dos processos sobre os algoritmos TLBA e Lowest modificado. Essas simulações se deram nas mesmas configurações de ambiente apresentadas na seção anterior, com exceção da ocupação dos processos, que foi fixa e igual a 10330000 ciclos. Essa simulação foi realizada 100 vezes e os resultados de média, mediana e desvio padrão, são apresentados na tabela 4. A ocupação por processo é fixa, permitindo avaliar a variação e, conseqüentemente, a precisão dos algoritmos de balanceamento de carga.

As simulações da tabela 4 comprovam a maior variação nos tempos médios de resposta do algoritmo Lowest modificado, o que é observado através do alto desvio padrão. Esse desvio deve-se ao fato deste algoritmo analisar um subconjunto randômico de computadores, para localizar o mais ocioso. Essa técnica é probabilística e não permite obter o computador mais ocioso do ambiente [34]. De outro lado, o baixo desvio padrão do algoritmo TLBA permite comprovar sua maior eficiência, principalmente nas políticas de informação e localização.

TABELA 4 - Médias, medianas e desvios padrão obtidos como resultado das simulações de precisão no tempo de resposta de processos, realizadas sobre os algoritmos TLBA e Lowest modificado

Algoritmo Simulado	Tempos Médios de Resposta dos Processos (ms)	Tempos Medianos de Resposta dos Processos (ms)	Desvio Padrão nos Tempos de Resposta dos Processos (ms)
TLBA	655.250,104	655.250,108	15,78408409
Lowest Modificado	902.153,4	872.479,6	149.771,5632

Além do baixo desvio padrão do algoritmo TLBA, esse algoritmo apresenta outra contribuição que compreende a geração de menores tempos médios e medianos de resposta dos processos. Portanto, os processos executam com maior desempenho.

O gráfico da figura 5 apresenta todos os dados que foram utilizados para gerar a tabela 4. Através deste gráfico pode-se observar a maior variação nos tempos de resposta do algoritmo Lowest modificado.

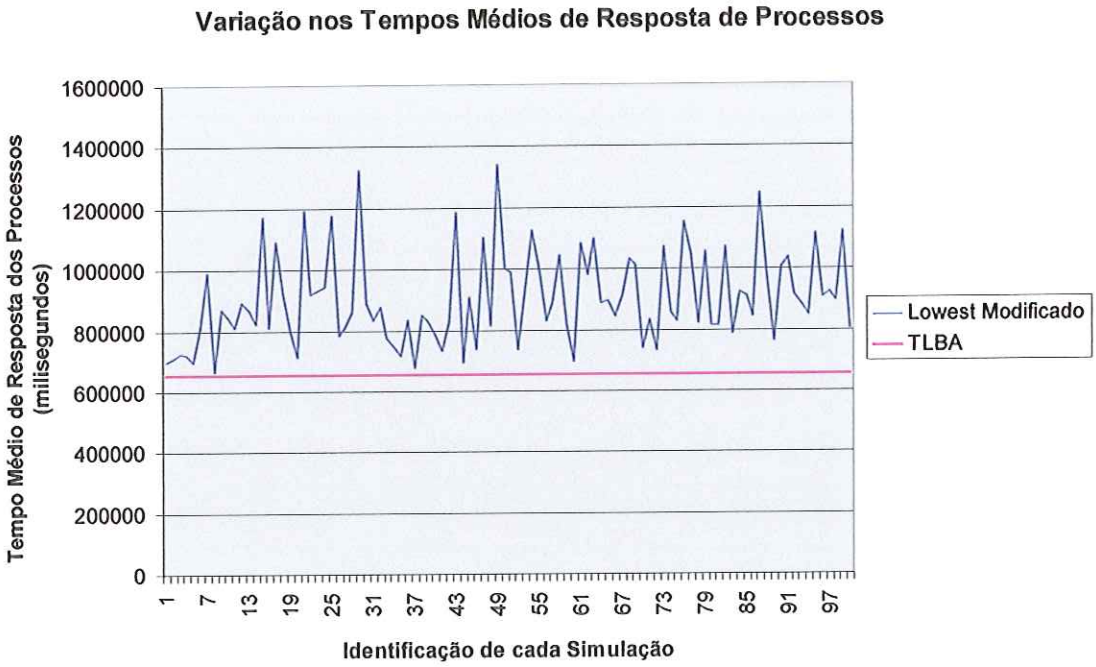

FIGURA 5 – Variação nos tempos de resposta de 100 simulações sucessivas do algoritmo TLBA e Lowest modificado, em ambiente composto por 7 computadores homogêneos

9.7 Simulações para analisar o desempenho do TLBA em um pequeno sistema heterogêneo

As simulações que seguem visam analisar o desempenho do algoritmo de balanceamento de carga TLBA, comparando-o ao Lowest modificado. Esses estudos foram realizados para pequenos ambientes compostos de hardware de capacidade de processamento heterogênea. Todos os resultados apresentados compreendem a média de 100 simulações sucessivas.

9.7.1 Política de Localização do Algoritmo Lowest Modificado

O algoritmo Lowest tem sua política de localização (seção 2.3) baseada na seleção randômica de um subconjunto de computadores, na requisição da carga desses computadores e, posteriormente, na análise para detectar o computador mais ocioso, que recebe processos iniciados no ambiente. O tamanho desses subconjuntos varia de acordo com o número de computadores do sistema, influenciando diretamente na qualidade da política de localização.

As simulações apresentadas nas seções seguintes analisam um sistema de capacidade de processamento heterogênea composto por 7 computadores. Nessas circunstâncias, são adotados 3 computadores por subconjunto, conforme avaliado na seção 9.6.1.

9.7.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado

Esta seção apresenta resultados de simulações do algoritmo TLBA e Lowest Modificado. As simulações foram realizadas para obter resultados em um ambiente composto por 7 computadores de capacidade de processamento heterogênea, sendo que 4 deles apresentam a capacidade de 368 ciclos por milisegundo e 3 deles de 139 ciclos por milisegundo. Os processos chegam ao sistema simulado através de uma função Poisson de distribuição de probabilidades, com média 100 segundos. Foram submetidos 320 processos ao sistema. Essa ocupação sempre manteve o ambiente sob alta carga.

Foi atribuído o valor de 368 e 139 ciclos por milisegundo para os computadores simulados, pois durante o projeto havia uma rede de computadores disponível para testes que apresentava recursos de tal capacidade (veja apêndice A e B). A escolha desse valor foi baseada na execução exaustiva de um programa em cada computador; este mesmo programa foi utilizado para testar o protótipo do TLBA (seção 10.5).

A função de distribuição de probabilidades Poisson para a chegada de processos é adotada pelos principais trabalhos da área de balanceamento de carga [1, 2, 16, 34] e, portanto, foi adotada neste trabalho.

O algoritmo Lowest modificado utilizou subconjuntos compostos por 3 computadores para sua política de localização. O algoritmo TLBA criou uma topologia lógica de interconexões na forma de árvore, contendo dois computadores por nível, totalizando três níveis.

A tabela 5 apresenta os tempos médios de resposta, resultantes das simulações do algoritmo TLBA e Lowest modificado.

TABELA 5 – Tempos médios de resposta do algoritmo TLBA e Lowest modificado em ambientes compostos por 7 computadores de capacidade de processamento heterogênea

Número de Processos	Ocupação por Processo (ciclos)	Poisson média (segs)	Subconjunto de Computadores Analisados pelo Lowest	Capacidade de Processamento por Computador (ciclos/ms)	Lowest Modificado Tempo Médio de Resposta dos Processos (ms)	TLBA Tempo Médio de Resposta dos Processos (ms)
320	1,00E+07	100	3	368 e 139	8,7436701000E+05	8,3431215000E+05
320	1,00E+08	100	3	368 e 139	8,9294303600E+06	8,6260096700E+06
320	1,00E+09	100	3	368 e 139	8,9918432930E+07	8,6541094870E+07
320	1,00E+10	100	3	368 e 139	8,9724402245E+08	8,6569206619E+08
320	1,00E+11	100	3	368 e 139	8,9697114641E+09	8,6572026521E+09
320	1,00E+12	100	3	368 e 139	9,0185968863E+10	8,6572308610E+10
320	1,00E+13	100	3	368 e 139	9,0240793834E+11	8,6572336769E+11
320	1,00E+14	100	3	368 e 139	8,9810014499E+12	8,6572339589E+12
320	1,00E+15	100	3	368 e 139	8,9777274689E+13	8,6572339869E+13
320	1,00E+16	100	3	368 e 139	8,9860080004E+14	8,6572339897E+14
320	1,00E+17	100	3	368 e 139	8,9554548820E+15	8,6572339900E+15
320	1,00E+18	100	3	368 e 139	8,9532098574E+16	8,6572339900E+16

Analisando os dados apresentados na tabela 5, pode-se concluir que os tempos médios de resposta obtidos na simulação do algoritmo TLBA são menores que do Lowest modificado. Pode-se concluir, portanto, que em pequenos sistemas heterogêneos com alta carga, o algoritmo TLBA apresenta melhores resultados.

9.7.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado

Foram realizadas simulações para avaliar a precisão nos tempos de resposta dos processos sobre os algoritmos TLBA e Lowest modificado. Essas simulações foram realizadas nas mesmas configurações de ambiente apresentadas na seção anterior, com

exceção da ocupação dos processos que foi linear e igual a 10330000 ciclos. Esta simulação foi realizada 100 vezes. Os resultados de média, mediana e desvio padrão são apresentados na tabela 6.

As simulações da tabela 6 comprovam a maior variação nos tempos médios de resposta do algoritmo Lowest modificado, o que é observado através do alto desvio padrão. Esse desvio deve-se ao fato deste algoritmo analisar um subconjunto randômico de computadores para localizar o mais ocioso. Essa técnica é probabilística e não permite obter o computador mais ocioso do ambiente [34]. De outro lado, o baixo desvio padrão do algoritmo TLBA permite comprovar sua maior eficiência, principalmente, nas políticas de informação e localização.

Além do baixo desvio padrão do algoritmo TLBA, este algoritmo apresenta outro benefício, que compreende a geração de menores tempos médios e medianos de resposta dos processos. Portanto, os processos executam com maior desempenho.

TABELA 6 – Médias, medianas e desvios padrão obtidos como resultado das simulações de precisão no tempo de resposta de processos, realizadas sobre os algoritmos TLBA e Lowest modificado

Algoritmo Simulado	Tempos Médios de Resposta dos Processos (ms)	Tempos Medianos de Resposta dos Processos (ms)	Desvio Padrão nos Tempos de Resposta dos Processos (ms)
TLBA	878154,5215	878153,7914	17,64180909
Lowest Modificado	1096288,999	1063515,619	206927,8546

O gráfico da figura 6 apresenta todos os dados que foram utilizados para gerar a tabela 6. Por meio deste gráfico pode-se observar a maior variação nos tempos de resposta do algoritmo Lowest modificado.

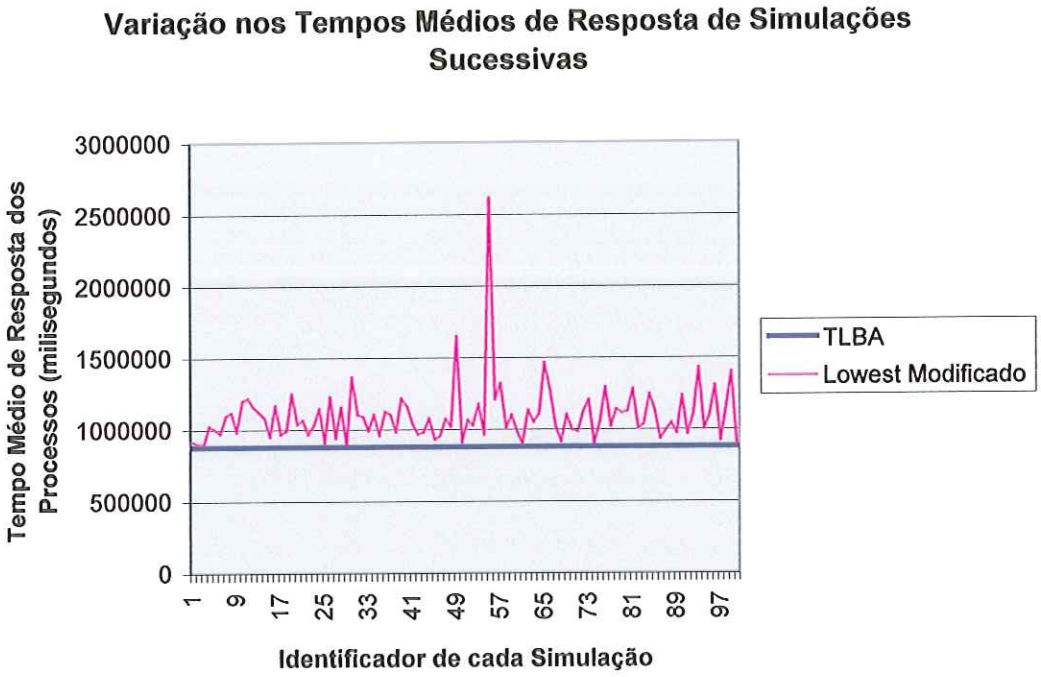

FIGURA 6 – Variação nos tempos de resposta de 100 simulações sucessivas do algoritmo TLBA e Lowest modificado, em ambiente composto por 7 computadores heterogêneos

9.8 Simulações para analisar o desempenho do TLBA em um grande sistema homogêneo

As simulações a seguir visam analisar o desempenho do algoritmo de balanceamento de carga TLBA, comparando-o ao Lowest modificado. Esses estudos foram realizados para grandes ambientes compostos de hardware de capacidade de processamento homogênea. Todos os resultados apresentados compreendem a média de 100 simulações sucessivas.

9.8.1 Política de Localização do Algoritmo Lowest Modificado

O algoritmo Lowest tem sua política de localização (seção 2.3) baseada na seleção randômica de um subconjunto de computadores, requisição da carga destes computadores e, posteriormente, análise para detectar o computador mais ocioso, que

recebe processos iniciados no ambiente. Essas operações ocorrem para cada processo iniciado no ambiente.

O tamanho dos subconjuntos de computadores analisados pelo Lowest é um parâmetro que influencia diretamente na qualidade da política de localização, ou seja, na escolha de um computador ocioso [8, 34]. Essa técnica utilizada pelo Lowest não garante que o computador mais ocioso do sistema seja escolhido.

O tamanho dos subconjuntos de computadores analisados pela política de localização do algoritmo Lowest é modificado de acordo com a escala do sistema, sendo proporcional à mesma. Sistemas formados por poucos computadores tendem a ter pequenos subconjuntos. Grandes sistemas apresentam subconjuntos maiores. Contudo, o tamanho destes subconjuntos não pode ser grande demais, pois gera muitas mensagens no meio de comunicação durante as operações de balanceamento de carga [8, 16].

O sistema analisado nas seções a seguir é formado por 781 computadores. Para simular o Lowest em seu desempenho máximo, é necessário definir o tamanho de seu subconjunto de computadores. Esta seção apresenta um estudo que resulta no tamanho deste subconjunto.

Antes de definir o tamanho do subconjunto de computadores do Lowest, são descritas as características da simulação, que utiliza 781 computadores. A capacidade de processamento de cada computador do sistema é de 368 ciclos por milisegundo, número que reflete a capacidade de um computador com um processador AMD Duron de 950 Mhz com 256 Mb de memória RAM (maiores detalhes no apêndice A). Os processos chegam ao sistema através de uma função Poisson de distribuição de probabilidades, com média 3 segundos. Foram submetidos 7810 processos ao sistema, com ocupação unitária de 10330000 ciclos.

A função de distribuição de probabilidades Poisson para a chegada de processos é adotada pelos principais trabalhos da área de balanceamento de carga [1, 2, 16, 34] e, por isso, foi também adotada neste trabalho.

O termo variável nas simulações apresentadas nesta seção foi o tamanho do subconjunto de computadores analisados pela política de localização do algoritmo Lowest. O tamanho do subconjunto variou do número mínimo de um computador até o

máximo de 780 computadores analisados. Os resultados destas simulações são apresentados no gráfico da figura 7.

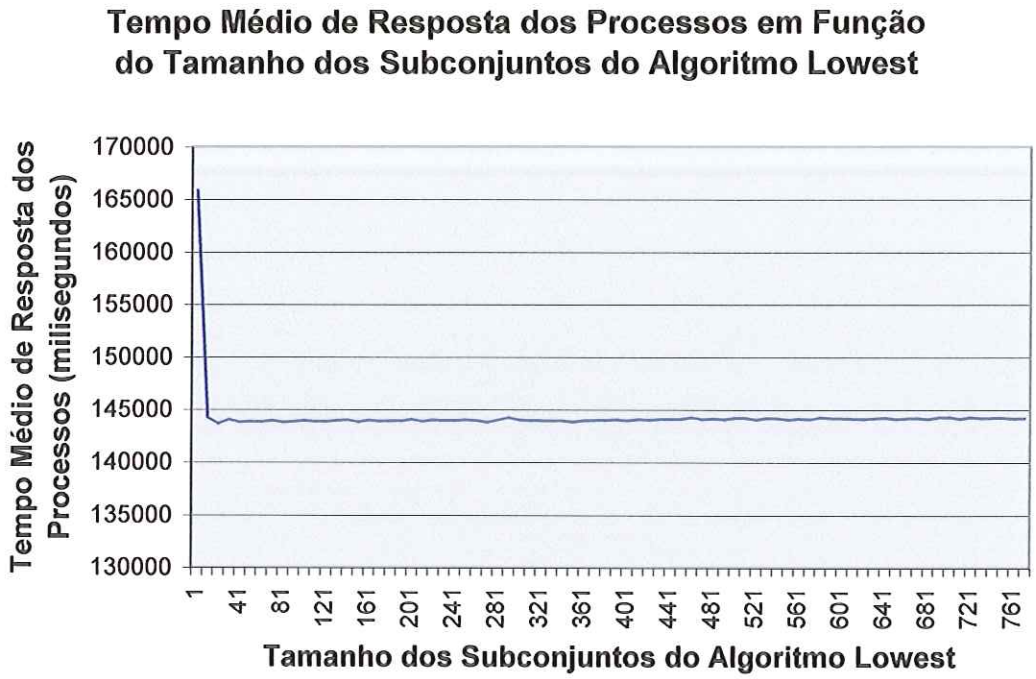

FIGURA 7 – Tempo médio de resposta dos processos em função do tamanho do subconjunto de computadores analisados pela política de localização do algoritmo Lowest

O gráfico da figura 7 e os dados apresentados na tabela 7 mostram a queda expressiva no tempo médio de resposta dos processos, quando o subconjunto de computadores é de tamanho igual a 11. Subconjuntos com mais de 11 elementos apresentam tempos médios de resposta estáveis para os processos. De outro lado, como se pode notar na tabela 7 e no gráfico da figura 8, o aumento no tamanho dos subconjuntos gera aumento no número de mensagens que trafegam no meio de comunicação durante as operações de balanceamento de carga.

TABELA 7 – Resumo dos resultados obtidos nas simulações para definir o número mais indicado de computadores analisados pela política de localização do algoritmo Lowest

Número de Processos	Ocupação por Processo (ciclos)	Poisson média (segs)	Capacidade de Processamento por Computador (ciclos/ms)	Número Computadores Analisados	Tempo Médio de Resposta por Processos (ms)	Número de Mensagens
7810	10330000	3	368	1	165860,5	23430
7810	10330000	3	368	11	144237,3	179630
7810	10330000	3	368	21	143700,4	335830

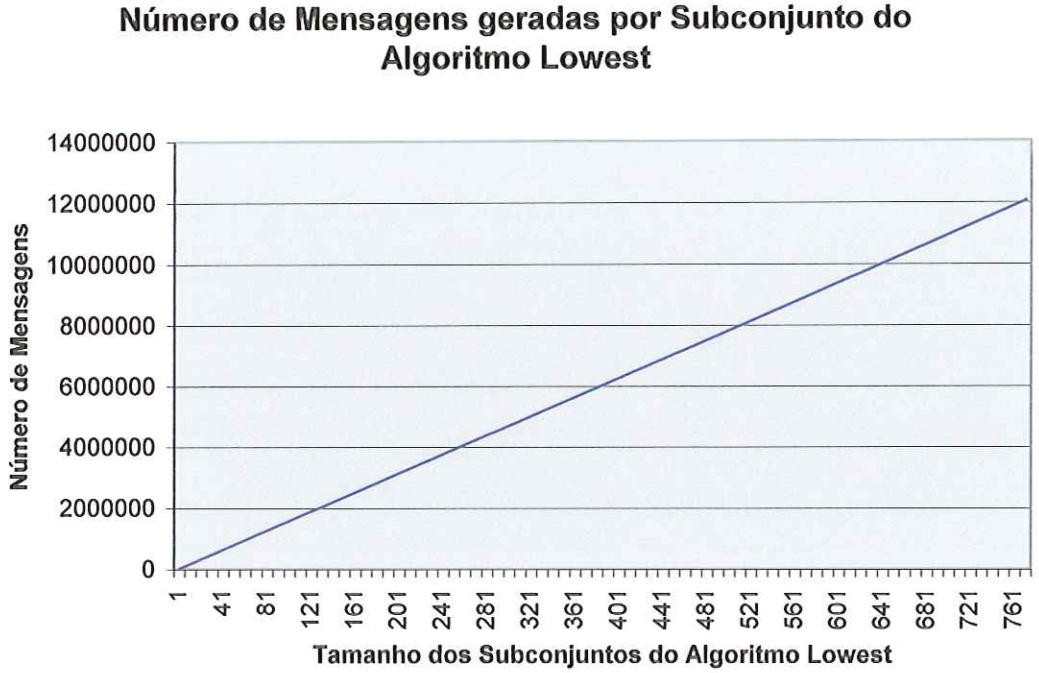

FIGURA 8 – Número de mensagens geradas no ambiente para diferentes tamanhos de subconjunto de computadores em um ambiente composto por 781 computadores homogêneos

Levando-se em consideração os resultados apresentados, optou-se por adotar subconjuntos compostos por 11 computadores. Esse tamanho de subconjunto é adotado nas próximas seções, onde são apresentadas simulações do Lowest.

9.8.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado

Esta seção apresenta e analisa resultados de simulações do algoritmo TLBA e Lowest modificado. As simulações foram realizadas sobre um ambiente composto por 781 computadores de capacidade de processamento homogênea. Cada computador tem a capacidade de 368 ciclos por milisegundo. Os processos chegam ao sistema através de uma função Poisson de distribuição de probabilidades com média 100 segundos. Foram submetidos 35145 processos ao sistema. Essa ocupação sempre manteve o ambiente sob alta carga.

O algoritmo Lowest modificado utilizou subconjuntos compostos por 11 computadores para sua política de localização. O algoritmo TLBA criou uma topologia lógica de interconexões na forma de árvore, contendo cinco computadores por nível, totalizando cinco níveis.

A tabela 8 apresenta os tempos médios de resposta resultantes das simulações do algoritmo TLBA e Lowest modificado.

TABELA 8 – Tempos médios de resposta do algoritmo TLBA e Lowest modificado em ambientes compostos por 781 computadores de capacidade de processamento homogênea

Número de Processos	Ocupação por Processo (ciclos)	Poisson média (segs)	Subconjunto de Computadores Analisados pelo Lowest	Capacidade de Processamento por Computador (ciclos/ms)	Lowest Modificado Tempo Médio de Resposta dos Processos (ms)	TLBA Tempo Médio de Resposta dos Processos (ms)
35145	1,00E+10	100	11	368	6,2159995608E+08	6,2156474184E+08
35145	1,00E+11	100	11	368	6,2469085561E+09	6,2465734548E+09
35145	1,00E+12	100	11	368	6,2500016636E+10	6,2496667285E+10
35145	1,00E+13	100	11	368	6,2503193101E+11	6,2499760509E+11
35145	1,00E+14	100	11	368	6,2503454303E+12	6,2500069804E+12
35145	1,00E+15	100	11	368	6,2503491403E+13	6,2500100731E+13
35145	1,00E+16	100	11	368	6,2503482897E+14	6,2500103823E+14
35145	1,00E+17	100	11	368	6,2503475474E+15	6,2500104132E+15
35145	1,00E+18	100	11	368	6,2503488649E+16	6,2500104163E+16

Analisando os dados apresentados na tabela 8, pode-se concluir que os tempos médios de resposta obtidos na simulação do algoritmo TLBA são menores que do Lowest modificado. Pode-se concluir, portanto, que em grandes sistemas homogêneos com alta carga, o algoritmo TLBA apresenta melhores resultados.

9.8.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado

Foram realizadas simulações para avaliar a precisão nos tempos de resposta dos processos sobre os algoritmos TLBA e Lowest modificado. Essas simulações aconteceram nas mesmas configurações de ambiente apresentadas na seção anterior, com exceção da ocupação dos processos que foi linear e igual a 10330000 ciclos. Esta simulação foi realizada 100 vezes. Os resultados de média, mediana e desvio padrão são apresentados na tabela 9.

As simulações da tabela 9 comprovam a maior variação nos tempos médios de resposta do algoritmo Lowest modificado, o que é observado através do alto desvio padrão. Esse desvio deve-se ao fato do algoritmo em questão analisar um subconjunto randômico de computadores para localizar o mais ocioso. Essa técnica é probabilística e não permite obter o computador mais ocioso do ambiente [34]. De outro lado, o baixo desvio padrão do algoritmo TLBA permite comprovar sua maior eficiência, principalmente nas políticas de informação e localização.

Além do baixo desvio padrão do algoritmo TLBA, esse algoritmo apresenta outro benéfico, que compreende a geração de menores tempos médios e medianos de resposta dos processos. Portanto, os processos executam com maior desempenho.

TABELA 9 – Médias, medianas e desvios padrão obtidos como resultado das simulações de precisão no tempo de resposta dos processos, realizadas sobre os algoritmos TLBA e Lowest modificado

Algoritmo Simulado	Tempos Médios de Resposta dos Processos (ms)	Tempos Medianos de Resposta dos Processos (ms)	Desvio Padrão nos Tempos de Resposta dos Processos (ms)
TLBA	14583693,91	14583688,91	86,81921307
Lowest Modificado	14599591,5	14599549,8	1480,660404

O gráfico da figura 9 apresenta todos os dados que foram utilizados para gerar a tabela 9. Observando-se este gráfico, pode-se notar a maior variação nos tempos de resposta do algoritmo Lowest modificado.

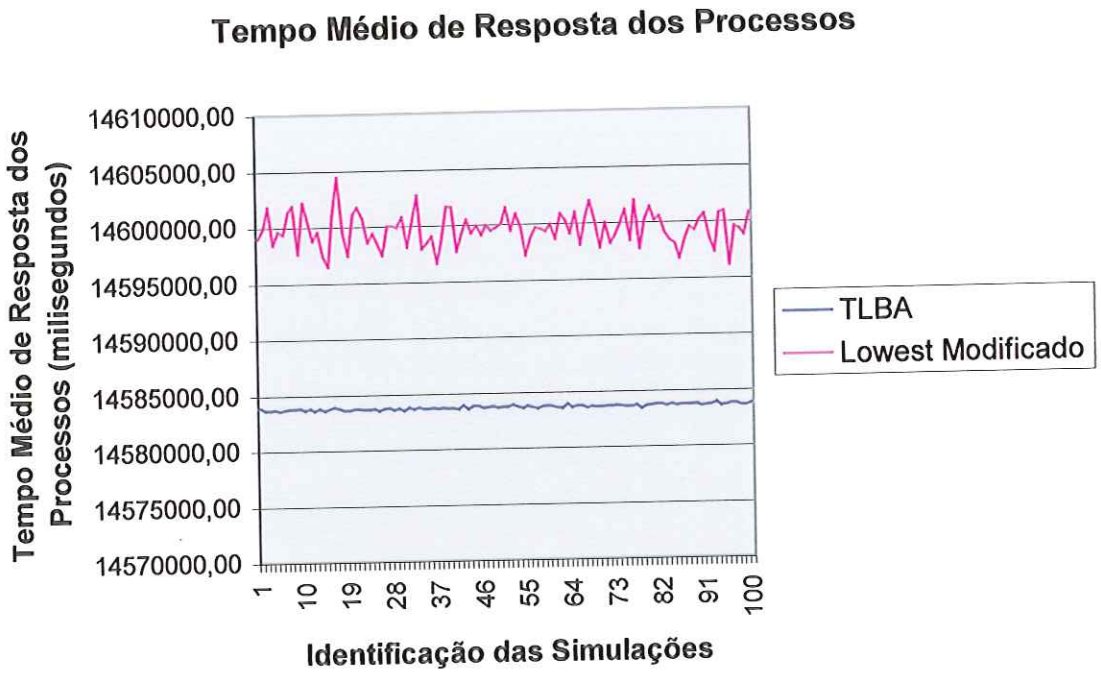

FIGURA 9 – Tempos de resposta obtidos em 100 simulações sucessivas

9.9 Simulações para analisar o desempenho do TLBA em um grande sistema heterogêneo

As simulações que se seguem, visam analisar o desempenho do algoritmo de balanceamento de carga TLBA, comparando-o ao Lowest modificado. Esses estudos foram realizados para grandes ambientes compostos de hardware de capacidade de processamento heterogênea. Todos os resultados apresentados compreendem a média de 100 simulações sucessivas.

9.9.1 Política de Localização do Algoritmo Lowest Modificado

O algoritmo Lowest tem sua política de localização (seção 2.3) baseada na seleção randômica de um subconjunto de computadores, requisição da carga destes computadores e, posteriormente, análise para detectar o computador mais ocioso, que

recebe processos iniciados no ambiente. O tamanho destes subconjuntos varia de acordo com o número de computadores do sistema, influenciando diretamente na qualidade da política de localização.

As simulações apresentadas nas seções seguintes analisam um sistema de capacidade de processamento heterogênea, composto por 781 computadores. Nessas circunstâncias, são adotados 11 computadores por subconjunto, conforme avaliado na seção 9.8.1.

9.9.2 Analisando os Tempos Médios de Resposta do Algoritmo TLBA e Lowest Modificado

Esta seção apresenta e analisa resultados de simulações do algoritmo TLBA e Lowest modificado. As simulações realizaram-se sobre um ambiente composto por 781 computadores de capacidade de processamento heterogênea, onde 391 computadores têm a capacidade de 368 ciclos por milisegundo e os 390 restantes, a capacidade de 139 ciclos por milisegundo. Os processos chegam ao sistema através de uma função Poisson de distribuição de probabilidades com média 100 segundos. Foram submetidos 35145 processos ao sistema. Essa ocupação sempre manteve o ambiente sob alta carga.

O algoritmo Lowest modificado utilizou subconjuntos compostos por 11 computadores para sua política de localização. O algoritmo TLBA criou uma topologia lógica de interconexões na forma de árvore, contendo cinco computadores por nível, totalizando cinco níveis.

A tabela 10 apresenta os tempos médios de resposta resultantes das simulações do algoritmo TLBA e Lowest modificado.

TABELA 10 – Tempos médios de resposta do algoritmo TLBA e Lowest modificado em ambientes compostos por 781 computadores de capacidade de processamento heterogênea

Número de Processos	Ocupação por Processo (ciclos)	Poisson média (segs)	Subconjunto de Computadores Analisados pelo Lowest	Capacidade de Processamento por Computador (ciclos/ms)	Lowest Modificado Tempo Médio de Resposta dos Processos (ms)	TLBA Tempo Médio de Resposta dos Processos (ms)
35145	1,00E+10	100	11	368 e 139	9,0352098932E+08	9,0349469055E+08
35145	1,00E+11	100	11	368 e 139	9,0661207290E+09	9,0658727685E+09
35145	1,00E+12	100	11	368 e 139	9,0692197539E+10	9,0689656981E+10
35145	1,00E+13	100	11	368 e 139	9,0695255607E+11	9,0692749095E+11
35145	1,00E+14	100	11	368 e 139	9,0695560153E+12	9,0693058444E+12
35145	1,00E+15	100	11	368 e 139	9,0695569517E+13	9,0693089372E+13
35145	1,00E+16	100	11	368 e 139	9,0695613143E+14	9,0693092466E+14
35145	1,00E+17	100	11	368 e 139	9,0695587575E+15	9,0693092776E+15
35145	1,00E+18	100	11	368 e 139	9,0695602819E+16	9,0693092806E+16

Analisando os dados apresentados na tabela 10, vê-se que os tempos médios de resposta obtidos na simulação do algoritmo TLBA são menores que do Lowest modificado. Pode-se concluir, portanto, que em grandes sistemas heterogêneos com alta carga, o algoritmo TLBA apresenta melhores resultados.

9.9.3 Análise da Precisão nos Tempos de Resposta dos Algoritmos TLBA e Lowest Modificado

Esta seção apresenta e analisa resultados de simulações do algoritmo TLBA e Lowest modificado. As simulações foram realizadas sobre um ambiente composto por 781 computadores de capacidade de processamento heterogênea, onde 391 computadores têm a capacidade de 368 ciclos por milisegundo e os 390 restantes, a capacidade de 139 ciclos por milisegundo. Os processos chegam ao sistema através de uma função Poisson de distribuição de probabilidades com média 3 segundos. Foram submetidos 7810 processos ao sistema, cada um com a ocupação de 10330000 ciclos/ms. Essa ocupação sempre manteve o ambiente sob baixa carga.

O algoritmo Lowest modificado utilizou subconjuntos compostos por 11 computadores para sua política de localização. O algoritmo TLBA criou uma topologia lógica de interconexões na forma de árvore, contendo cinco computadores por nível, totalizando cinco níveis.

A tabela 11 apresenta os tempos médios de resposta resultantes das simulações do algoritmo TLBA e Lowest modificado.

TABELA 11 – Médias, medianas e desvios padrão obtidos como resultado das simulações de precisão no tempo de resposta dos processos, realizadas sobre os algoritmos TLBA e Lowest modificado

Algoritmo Simulado	Tempos Médios de Resposta dos Processos (ms)	Tempos Medianos de Resposta dos Processos (ms)	Desvio Padrão nos Tempos de Resposta dos Processos (ms)
TLBA	21444793,61	21444786,57	75,34808115
Lowest Modificado	21432565,59	21432442,69	1281,798169

A tabela 11 permite concluir que, no caso de baixa carga, o tempo de resposta médio e mediano para o algoritmo Lowest é um pouco menor que o do algoritmo TLBA. Essa situação proposital foi criada com o objetivo de avaliar os dois algoritmos em situações de baixa carga. Pode-se observar que o desvio padrão nos tempos de resposta do algoritmo Lowest é expressivamente maior. Esse desvio padrão degrada o desempenho do algoritmo Lowest, conforme aumenta a carga do ambiente. Esse desvio, que neste caso é em torno de 17 vezes maior, caracteriza a maior instabilidade do Lowest, causada por sua política de localização e informação.

Avaliando de forma detalhada a tabela 11, pode-se concluir que mesmo em baixa carga, os tempos médios e medianos de resposta gerados pelo algoritmo TLBA apresentam pouca diferença dos gerados pelo Lowest modificado. Por exemplo, neste caso, a diferença no tempo médio de resposta foi de 12228,02 milisegundos, ou seja, 12,22802 segundos para uma execução que demora 21444,79361 segundos, ou seja, 5,95689 horas.

9.10 Simulações para Analisar o Número de Mensagens Trocadas nas Operações de Balanceamento de Carga

Esta seção apresenta simulações que visam analisar o número de mensagens trocadas em uma rede de computadores que utiliza os algoritmos TLBA e Lowest modificado. A situação simulada foi composta por 7 computadores de capacidade de

processamento homogênea e igual a 368 ciclos/ms. Foram submetidos diferentes números de processos ao sistema. Esses números variaram de 10 a 350. Foi utilizada a função Poisson de distribuição de probabilidades de chegada de processos ao ambiente com média 3 segundos.

O algoritmo Lowest modificado utilizou subconjuntos compostos por 3 computadores, para sua política de localização. O algoritmo TLBA criou uma topologia lógica de interconexões na forma de árvore contendo cinco computadores por nível, totalizando cinco níveis.

Os resultados das simulações, em termos de número de mensagens geradas, são apresentados no gráfico da figura 10.

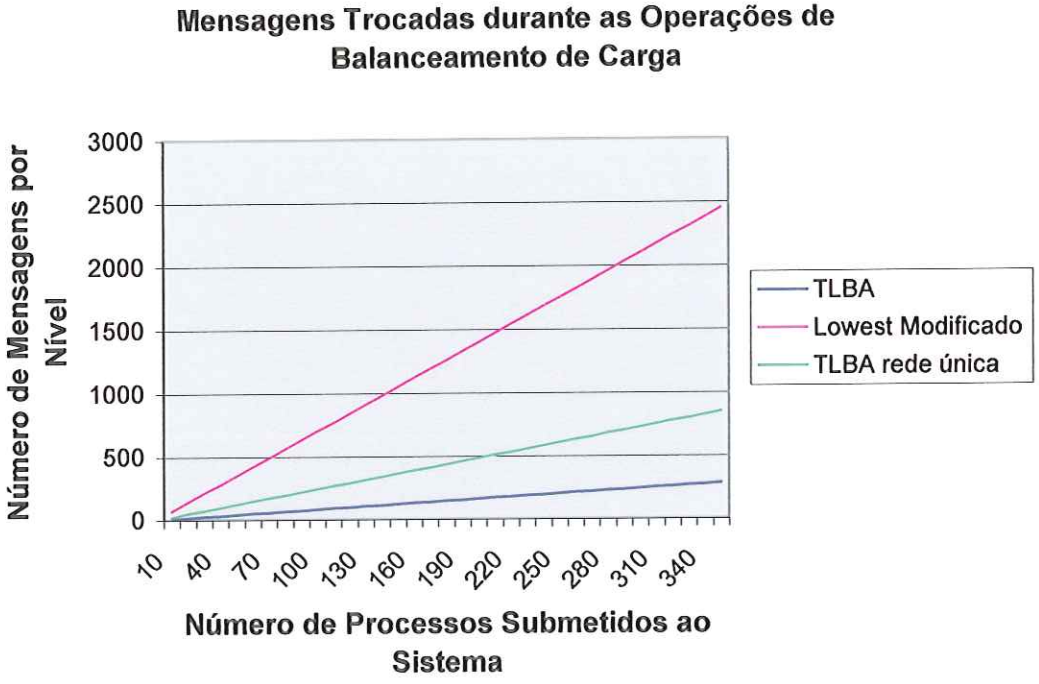

FIGURA 10 – Número de mensagens trocadas nas operações de balanceamento de carga

O gráfico da figura 10 apresenta três simulações: uma do TLBA em uma rede única, outra do Lowest, na mesma configuração e uma terceira simulação que apresenta o TLBA criando a árvore lógica sobre uma estrutura de topologia física, tal como analisado na seção 7.3.5. Pelo gráfico, pode-se concluir que o TLBA apresenta um

menor número de mensagens do que o Lowest modificado, tanto para topologia de árvore lógica quanto para topologia física.

Os algoritmos TLBA e Lowest modificados variam o número de mensagens trocadas no meio, conforme variam o número de processos. No Lowest, as mensagens são geradas para capturar, sob demanda, informações de carga dos computadores de um subconjunto. No TLBA, as mensagens são geradas pela política de localização e informação.

As simulações permitiram concluir que as políticas de informação e localização do algoritmo TLBA geram menor número de mensagens no meio de comunicação. Conseqüentemente, há menor sobrecarga na rede. Em topologia de rede física, o TLBA é o maior favorecido, pois as mensagens são subdivididas entre concentradores distintos que interconectam níveis da rede.

9.11 Simulações para Analisar o Número de Mensagens Trocadas nas Operações de Balanceamento de Carga em Função do Número de Computadores por Nível da Árvore do Algoritmo TLBA

As simulações desta seção permitem analisar os impactos impostos pela variação no número de computadores por nível de uma árvore criada pelo algoritmo TLBA. Esse número indica o número de computadores sucessores de outro computador α, localizado em um nível *Nk* da árvore.

As simulações a seguir foram realizadas para um ambiente composto por 781 computadores de capacidade de processamento homogênea e igual a 368 ciclos/ms. Foram submetidos 3000 processos ao ambiente, segundo a função Poisson de distribuição de probabilidades com média de 3 segundos. O TLBA foi simulado e analisado em três situações:

1) Computadores divididos em uma árvore de dois níveis – neste caso há um computador no primeiro nível da árvore e 780 no segundo. Esta situação simulada é semelhante ao algoritmo Central [16].

2) Computadores divididos em cinco níveis – neste caso, há um computador no primeiro nível, cinco níveis, e cada computador α de um nível *Nk* apresenta 5 sucessores (Esta topologia lógica em níveis, criada pelo TLBA, é apresentada no capítulo 7).
3) Computadores divididos em 781 níveis – neste caso há um computador por nível da árvore.

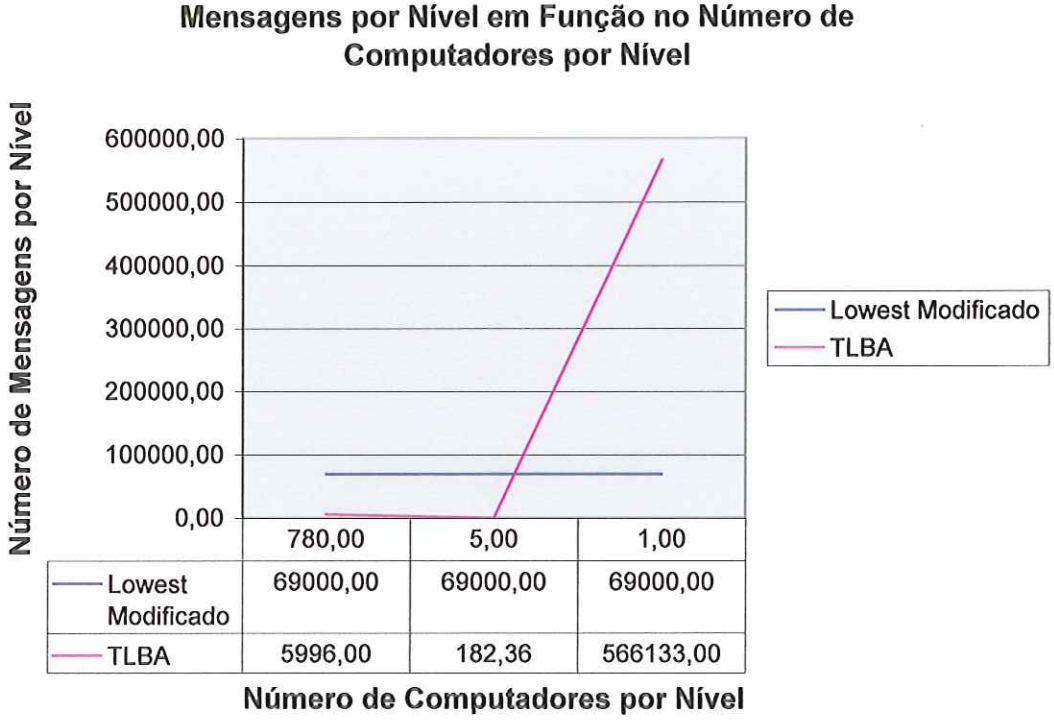

FIGURA 11 – Número de mensagens trocadas nas operações de balanceamento de carga, utilizando o algoritmo TLBA e Lowest modificado

O gráfico da figura 11 apresenta os resultados obtidos nas simulações do algoritmo TLBA e Lowest modificado. Esses resultados permitem concluir que o algoritmo Lowest modificado gera um número de mensagens constante e dependente do tamanho do subconjunto de computadores analisados por sua política de localização [8, 16, 34]. O algoritmo TLBA varia o número de mensagens de acordo com o número de computadores por nível da árvore. Árvores com muitos níveis e, conseqüentemente,

poucos computadores por nível, geram muita sobrecarga nas buscas em profundidade utilizadas para localizar os computadores ociosos (veja política de localização – seção 7.3.2). Árvores com poucos níveis geram maior número de mensagens entre níveis e diminuem a escalabilidade do ambiente.

Um número intermediário de sucessores para cada computador α de um nível *Nk* cria subconjuntos de comunicação bem dimensionados, que minimizam o número de mensagens geradas no meio de comunicação e, portanto, otimizam o balanceamento de carga e aumentam a escalabilidade [2, 9, 16, 28-31].

Nas simulações do TLBA em dois níveis, apresentadas na figura 11, o algoritmo tem estrutura semelhante ao Central [16]. Esta estrutura gera menos mensagens se comparada a uma árvore com um computador por nível. Contudo, apresenta baixa escalabilidade para o sistema, pois não há suportes físicos ilimitados para interconectar computadores em uma mesma rede fisicamente única. Outra limitação dessa técnica é que, quanto maior o número de computadores, maior a carga do computador central [2, 16].

Por meio desses resultados apresentados, pode-se concluir que o algoritmo TLBA gera um menor número de mensagens no meio de comunicação, quando o tamanho do subconjunto de sucessores estiver bem dimensionado. Outra característica do TLBA é sua escalabilidade, conseqüência da hierarquia em forma de árvore, que cria grupos de comunicação entre um antecessor e seus sucessores.

9.12 Resumo do Capítulo

Este capítulo apresentou resultados de simulações dos algoritmos TLBA e Lowest modificado. O TLBA apresenta maior desempenho que o Lowest modificado, em pequenos e grandes ambientes, os quais podem ser homogêneos ou heterogêneos. As principais características do TLBA são o baixo desvio padrão na distribuição da carga no ambiente, o menor número de mensagens geradas no ambiente e a alta escalabilidade.

10. IMPLEMENTAÇÃO, AVALIAÇÃO E ANÁLISE DE UM PROTÓTIPO DO TLBA

10.1 Considerações Iniciais

As simulações do TLBA permitiram, através de resultados, comprovar suas contribuições [9]. Contudo, para validar estes resultados, foi necessário construir um protótipo do algoritmo TLBA. As seções a seguir apresentam o protótipo e seus resultados.

10.2 Objetivo da Implementação

O principal objetivo da implementação de um protótipo (código-fonte no Apêndice D) do algoritmo TLBA é constatar os resultados de simulação (capítulo 9) e, conseqüentemente, comprovar sua eficiência na geração de tempos médios de resposta mais baixos para os processos e menor número de mensagens no meio de comunicação.

10.3 Implementação

As seções a seguir apresentam os módulos que compõem o protótipo do algoritmo TLBA e demais aspectos sobre seu funcionamento.

10.3.1 Componentes do Protótipo

Para simplificar o desenvolvimento do protótipo do algoritmo TLBA, ele foi subdividido nos seguintes módulos:

1) Server – Este módulo representa o componente de software *Peer* (seção 7.3.1). Ele implementa as políticas do algoritmo de balanceamento de carga TLBA. O Server é implementado como um *daemon* do sistema operacional Linux, ficando ativo em *background*, e trocando mensagens com os demais Servers, que executam em outros computadores do sistema. São funções deste módulo:
    a. Trocar mensagens com informações de carga dos computadores (ver política de informação – seção 7.3.1).
    b. Realizar buscas em profundidade na árvore criada pelo TLBA para localizar o computador mais ocioso no sistema (ver política de localização – seção 7.3.2).
    c. Gerir o *checkpointing* e migração de processos dos computadores sobrecarregados (ver política de transferência - seção 7.3.3). O *checkpointing* é realizado através de um pacote sob licença GNU/GPL denominado CKPT [77], que salva o contexto de um processo em um arquivo no disco.
    d. Gerir filas de processos que executam nos computadores do sistema.
    e. Reiniciar processos migrados. Os contextos de todos os processos migrados ficam salvos em um volume NFS (*Network File System*) [13, 78], comum a todos os computadores do ambiente. Ao migrar um processo, o Server receptor reinicia o contexto, salvo em disco, utilizando o pacote CKPT.
2) Client – Este módulo conecta-se no módulo Server do computador raiz da árvore e submete um processo para execução. Essa conexão é feita através de *sockets*. O arquivo executável do processo submetido deve estar no volume NFS, comum a todos os computadores do sistema. Originalmente, conforme proposto pelo algoritmo TLBA, o Client deveria conectar-se ao Broker. Contudo, o módulo Broker não foi construído como parte integrante deste protótipo.
3) Cluster.conf – Este é um arquivo de configurações do módulo Server, que armazena definições de portas TCP (*Transmission Control Protocol*) [13, 79], utilizadas para a troca de mensagens geradas pela política de informação. Este arquivo também contém configurações do volume NFS.

4) Cluster-client.conf – Este é o arquivo de configurações do módulo Client. Ele contém o endereço do computador raiz da árvore e as portas de comunicação para a submissão de processos para a árvore.
5) Script.sh – É um arquivo de Shell Script do sistema operacional Linux, que tem definições das variáveis de ambiente utilizadas pelo pacote CKPT para realizar *checkpointing*, e o caminho do programa a ser executado.
6) CKPT – Pacote GNU/GPL [80] que gera o *checkpointing* de processos em disco a partir de seus identificadores. Este pacote foi adotado pelo protótipo, mas pode ser alterado.

10.3.2 Aspectos de Configuração

Para utilizar o protótipo do algoritmo TLBA, é necessário configurar os computadores que compõem o ambiente. Essa configuração é subdividida nas seguintes etapas:

1) Configurar o servidor SSH [81] – esse servidor habilita conexões remotas seguras nos computadores. Não é um servidor diretamente utilizado pelo protótipo, mas é recomendado para simplificar a administração do sistema.
2) Configurar o servidor e os clientes NFS – Deve-se definir um dos computadores do sistema como servidor do volume NFS. Esse volume armazena os programas, que podem ser executados no ambiente, bem como os *checkpointings*.
3) Instalar o módulo Server em cada computador do sistema.
4) Definir a árvore lógica de conexão entre os computadores – a organização dos computadores em uma estrutura lógica de árvore é feita através do arquivo de configurações Cluster.conf.
5) Criar um arquivo Script.sh contendo as variáveis de ambiente utilizadas pelo CKPT e o caminho para o programa a ser executado.
6) Iniciar a execução do módulo Server no computador raiz e depois nos níveis subseqüentes, até chegar nas folhas. Essa ordem deve ser seguida, pois os computadores de um determinado nível conectam-se a seus antecessores na árvore durante sua inicialização.
7) Concluindo os passos descritos anteriormente, pode-se iniciar a submissão de processos utilizando o módulo Client.

10.3.3 Considerações

Esta seção apresenta algumas características e direções tomadas durante o desenvolvimento do protótipo do algoritmo TLBA. No início, foi necessário optar por um sistema operacional, uma linguagem de programação e pela forma que esse protótipo seria construído. O sistema operacional escolhido foi o Linux. A linguagem de programação escolhida foi C, utilizando-se o compilador GNU/GCC, um software livre, integrante das distribuições Linux [73, 82]. As formas planejadas para desenvolver esse protótipo foram: a criação de um pacote que alterasse as características do escalonador do sistema, ou a criação de um serviço (*daemon*) do sistema.

A criação de um pacote para alterar o escalonador do sistema foi analisada e testada no início do desenvolvimento desse protótipo. Para isso, foi implementada uma chamada de função *inline* dentro do arquivo *sched.c*, que implementa o escalonador do *kernel* do sistema operacional Linux. Essa função era invocada todas as vezes que o escalonador era chamado. Ela contava um período e iniciava a análise de carga segundo a política de informação. Esse tipo de implementação não teve sucesso, pois as tarefas executadas pela função *inline* consumiam muito tempo para associá-la ao escalonador, causando sobreposição de execuções [73].

Após analisar as alterações no escalonador e não obter resultados satisfatórios, o protótipo foi criado como um serviço do sistema operacional Linux. No Linux, um serviço funciona como um *daemon*, que é representado por um processo executando em *background* e que suporta chamadas especiais de sistema para manipular sua execução.

São características de implementação do protótipo: a comunicação, que foi implementada através de *sockets* [79, 83]; a simultaneidade no atendimento de múltiplas requisições, que foi implementada através de *threads* [79]; o suporte transparente na execução de aplicações, que esconde do usuário onde os processos estão sendo executados; o suporte à execução de aplicações legadas.

10.4 Características e Limitações do Protótipo do Algoritmo TLBA

Antes de optar pelo uso do protótipo do TLBA, deve-se conhecer suas características e limitações. As características estão relacionadas ao acesso do volume

NFS, ao acesso de base de dados e à submissão de processos ao ambiente. A limitação está associada à perda de referência dos canais de comunicação durante a reinicialização de processos transferidos.

Para configurar um ambiente com o protótipo do TLBA, é necessário criar um volume NFS compartilhado. Somente aplicações presentes nesse volume podem ser executadas pelo protótipo. Os *checkpointings* de processos são salvos em arquivos deste volume. Todo *checkpointing* salvo nesse volume pode ser reiniciado por outro computador do ambiente.

Aplicações que utilizam base de dados podem apresentar problemas, executando sobre o protótipo. Para esse tipo de aplicação é recomendada a criação de um *alias* para um servidor de banco de dados centralizado. Essa técnica faz-se necessária dado que a aplicação perde sua conexão com o banco de dados durante suas migrações. Contudo, mesmo adotando tal técnica, é necessário reiniciar a conexão com o servidor de banco de dados após uma migração.

A submissão de processos ao ambiente é feita através do módulo Client (seção 10.3.1) que se conecta ao computador raiz do sistema e submete o caminho de um arquivo *Shell script*. Esse arquivo contém variáveis de ambiente utilizadas pelo CKPT para realizar *checkpointing*. Além disto, o arquivo também contém uma variável de ambiente que contém o caminho para o programa a ser executado. Esse caminho se refere a um programa localizado no volume NFS, comum a todos os computadores.

O protótipo apresenta uma limitação para recuperar canais de comunicação de processos transferidos. Quando um processo migra, suas referências de ponteiros, tais como arquivos e *sockets*, são perdidas, não permitindo a reinicialização da aplicação. Essa limitação pode ser resolvida através de rotinas que detectem a perda de canais e os reinicie.

10.5 Avaliação dos resultados de execução do protótipo TLBA

Esta seção apresenta resultados obtidos em execuções do protótipo do algoritmo TLBA, descrito anteriormente. Esses resultados são comparados aos resultados de simulação (capítulo 9). Essas comparações permitiram validar o simulador, para

predizer o comportamento de ambientes que utilizam o algoritmo de balanceamento de carga TLBA.

Antes de iniciar as execuções do protótipo, foi necessário avaliar a capacidade de processamento cada um dos computadores do ambiente. Com esse objetivo, foi criado um programa na linguagem C para analisar o total de ciclos que um computador suporta. Foi utilizada uma chamada do sistema Linux, que conta o número de ciclos consumidos. Essa chamada é denominada *clock()*.

Após medir o número de ciclos suportados pelos computadores, foi escrito e submetido um outro programa ao ambiente. Esse programa também foi executado em cada computador do sistema e seus tempos de resposta capturados. Esses tempos foram divididos pelo número de ciclos calculados pelo programa em linguagem C, gerando a capacidade de processamento do sistema em executar esse aplicativo específico, em ciclos por milisegundo.

Para os testes, duas configurações com 7 computadores foram montadas. A primeira, contendo 7 computadores de capacidade de processamento homogênea e igual a 368 ciclos/ms (configuração apresentada no apêndice A). A segunda, contendo 5 computadores de capacidade igual a 368 ciclos/ms e 2 com capacidade de 139 ciclos/ms (configuração apresentada no apêndice B).

A tabela 12 e o gráfico da figura 12 apresentam os resultados obtidos nas execuções do protótipo e do simulador para um ambiente homogêneo composto por 7 computadores de capacidade de processamento igual a 368 ciclos/ms. Essas execuções foram feitas com processos que ocupam 10330000 ciclos. Foram submetidos 350 processos ao sistema. O número de ciclos por processo e processos no sistema foi definido com o intuito de gerar alta carga no ambiente (*load average* [73] do Linux em torno de 78), pois testando o protótipo em altas cargas, pode-se avaliá-lo em seu limite.

As execuções do protótipo foram subdivididas em 3 tipos de carga. Na primeira execução, o programa foi executado de forma exaustiva sem realizar nenhuma operação de entrada e saída. Na segunda execução, o código do programa foi alterado para consumir 500 ms de entrada e saída após um bloco de comandos. Foram definidos 1025 blocos de comandos. Na terceira execução, o código do programa foi alterado para

consumir 500 ms de entrada e saída após um bloco de comandos. Neste caso, foram definidos 2048 blocos de comandos.

Os três tipos de execução apresentados anteriormente foram propostos com a intenção de analisar o protótipo executando aplicações com características distintas: um sistema CPU-bound [85], um sistema que realiza um número intermediário de operações de entrada e saída e CPU, e um terceiro sistema IO-bound [85].

TABELA 12 – Tempos médios de resposta obtidos como resultados das execuções do protótipo do TLBA e do simulador em um ambiente homogêneo

Execuções	Simulador	Protótipo	Variação
10330000	698125,6005	697645	0,00068889
10330000 com 1025 blocos de comandos	1210625,6	1194509	0,013492239
10330000 com 2048 blocos de comandos	1722125,6	1760153	0,022081664

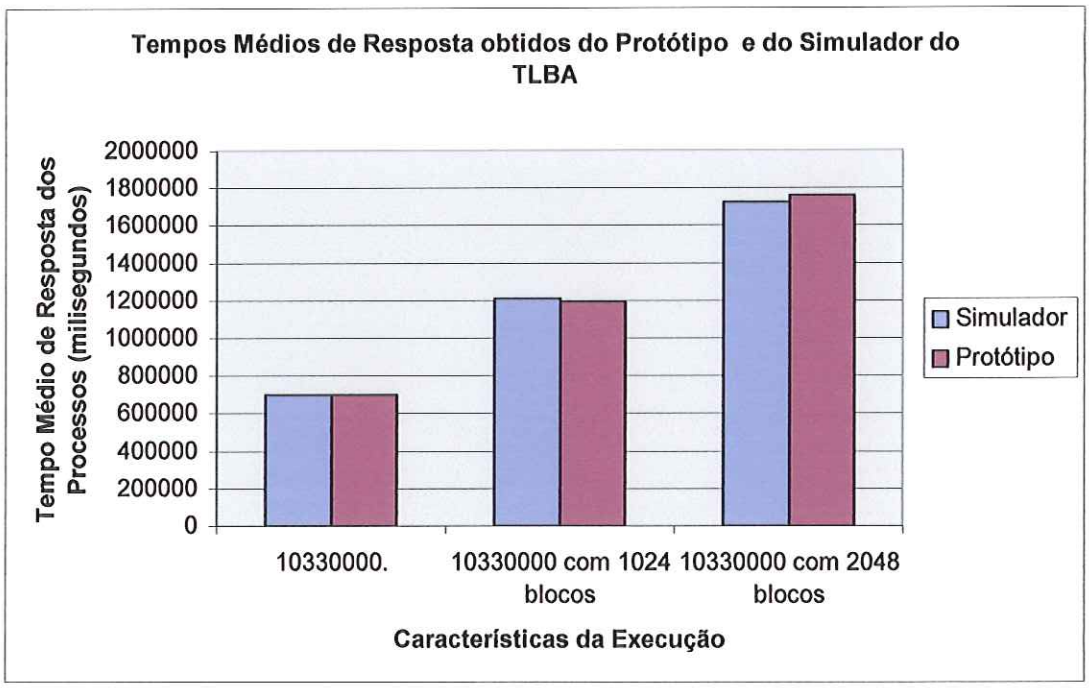

FIGURA 12 – Tempos médios de resposta obtidos como resultados das execuções do protótipo do algoritmo TLBA e do simulador em um ambiente homogêneo

Observando os resultados da tabela 12, pode-se constatar uma pequena variação entre os valores obtidos na execução do protótipo e os resultados do simulador. Essa baixa variação está entre 0,068889% e 2,2081664%, comprovando a viabilidade do simulador em analisar a construção de ambientes de balanceamento de carga que utilizam o algoritmo TLBA.

A tabela 13 apresenta o número médio de mensagens de balanceamento de carga trocadas durante a execução do protótipo e as simuladas. A variação nesse número de mensagens é de 5,8823529%, o que permite, também, concluir que o simulador avalia, com alto grau de precisão, o tráfego de mensagens no ambiente.

A tabela 13 apresenta duas colunas, uma contendo o número de mensagens trocadas entre níveis das árvores e outra com o número total de mensagens trocadas no ambiente. Se durante o projeto do sistema for adotada a construção de uma rede única, o número de mensagens geradas é o da coluna "Total". Se durante o projeto for adotada a construção de redes fisicamente distintas (seção 7.3.5), o número médio de mensagens é o da coluna "Por nível".

TABELA 13 – Número de mensagens geradas na rede de computadores

Informações	Por nível	Total
Simulador	283,3333333	850
Protótipo	300	900
Variação	5,8823529%	5,8823529%

A seguir, são apresentadas as execuções do protótipo e simulações sobre um ambiente heterogêneo composto de 5 computadores com capacidade de processamento igual a 368 ciclos/ms e 2 outros, com capacidade de 139 ciclos/ms. Essas simulações seguiram o mesmo padrão das simulações realizadas no ambiente homogêneo, onde havia 2 computadores por nível e 3 níveis compunham a topologia lógica de árvore criada pelo TLBA.

A tabela 14 e o gráfico da figura 13 apresentam os resultados da execução do protótipo e simulador no ambiente heterogêneo. Esses resultados demonstram a baixa variação nos tempos médios de resposta que ficaram em torno de 1 a 4%.

TABELA 14 - Tempos médios de resposta obtidos como resultados das execuções do protótipo do TLBA e do simulador em um ambiente heterogêneo

Execuções	Simulador	Protótipo	Variação
10330000	781348,8903	750352,93	4%
10330000 com 1025 blocos de comandos	1293848,89	1279203,00	1%
10330000 com 2048 blocos de comandos	1805348,89	1742498,00	4%

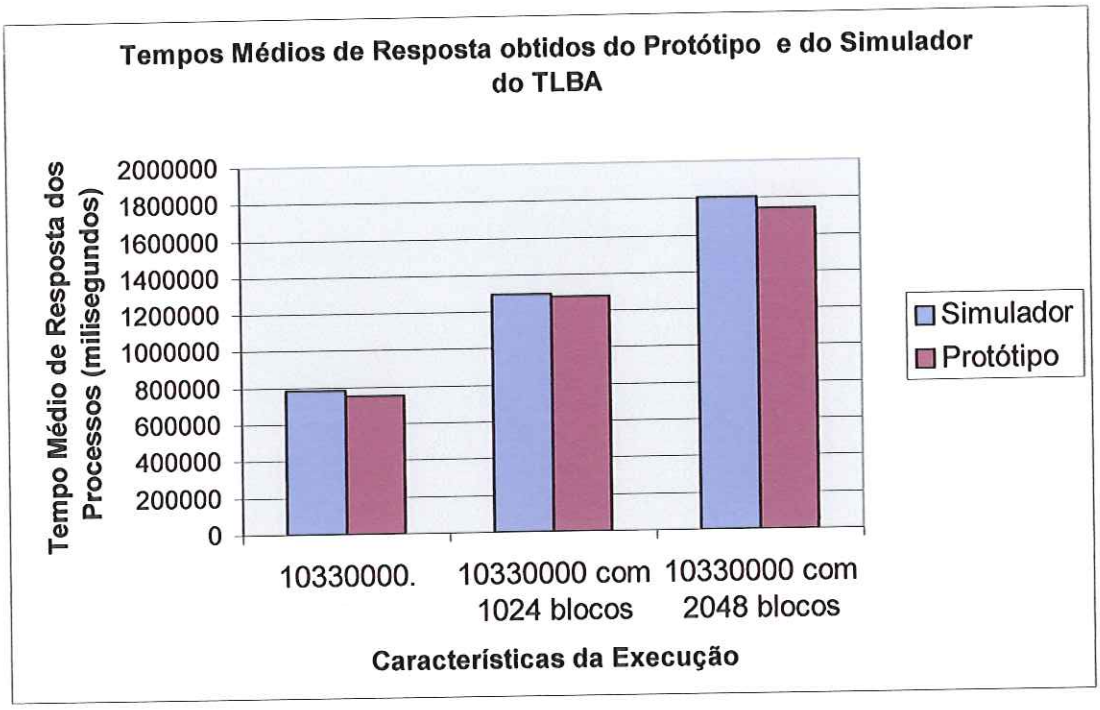

FIGURA 13 – Tempos médios de resposta obtidos como resultados das execuções do protótipo do TLBA e do simulador em um ambiente heterogêneo

A tabela 15 apresenta o número de mensagens geradas durante a execução do protótipo e do simulador. A variação do número de mensagens foi de 7%.

TABELA 15 – Mensagens trafegando na rede para o ambiente real e simulado

Informações	Por nível	Total
Simulador	257	771
Protótipo	275	825
Variação	7,0038911%	7,0038911%

O protótipo permitiu comprovar os valores obtidos nas simulações e, portanto, constatar as contribuições do algoritmo de balanceamento de carga TLBA. Os resultados apresentados nesta seção também atestam a utilidade do simulador na predição do comportamento e no projeto de ambientes que utilizam o TLBA.

10.6 Resumo do Capítulo

Este capítulo descreveu a construção do protótipo do algoritmo de balanceamento de carga TLBA, suas características e limitações. Esse protótipo foi submetido a testes e seus resultados comparados a simulações. Essas comparações permitiram comprovar as contribuições do algoritmo TLBA e atestaram a viabilidade do simulador no projeto de ambientes distribuídos, que utilizam esse algoritmo.

11. APLICABILIDADE DO ALGORITMO TLBA

O algoritmo de balanceamento de carga TLBA, proposto e avaliado neste projeto de doutorado, pode ser aplicado em ambientes distribuídos homogêneos e heterogêneos, que visam atingir alto desempenho e escalabilidade. Clusters, *Network of Workstations* e *Grid Computing* são os principais tipos de sistemas distribuídos que podem se beneficiar desses estudos. As contribuições do TLBA estão relacionadas ao aumento de desempenho na execução de aplicações e a menor sobrecarga no meio de comunicação.

As simulações e o protótipo do algoritmo TLBA permitem comprovar suas contribuições. O simulador construído (capítulo 9) possibilita projetar e analisar um sistema que utiliza o TLBA. O protótipo possibilita aplicar o TLBA em um ambiente composto por vários computadores interconectados em rede. Uma das características relevantes do protótipo é sua implementação no compilador GNU/GCC [74], suportado por mais de 20 sistemas operacionais, tais como Linux, Windows, FreeBSD, Solaris e outros [75]. Esse compilador gera binários para mais de 30 arquiteturas distintas de hardware, incluindo Motorola, IBM, MIPS, Intel, Texas, Alpha e outras [75].

12. CONCLUSÕES E TRABALHOS FUTUROS

Os algoritmos de balanceamento de carga visam homogeneizar a ocupação de recursos de um ambiente distribuído. Essa homogeneização é realizada para maximizar a ocupação dos recursos computacionais e, conseqüentemente, aumentar o desempenho das aplicações de um ambiente distribuído. Com esse objetivo, foi proposto um novo algoritmo de balanceamento de carga, denominado TLBA (*Tree Load Balancing Algorithm*) [9].

O TLBA foi proposto após estudos e análises de algoritmos existentes [2, 15-34, 38-42], que permitiram, também, obter melhorias nas técnicas de cálculo de índice de carga dos algoritmos de balanceamento de carga existentes [8]. Esse algoritmo foi avaliado através de resultados de um simulador e de um protótipo. Ambos puderam comprovar a queda nos tempos médios de resposta dos processos e o baixo número de mensagens geradas no meio de comunicação. Essas são as principais contribuições resultantes deste projeto de doutorado [9].

As contribuições do TLBA motivaram estudos e aplicações deste algoritmo. Atualmente, existem três mestrados relacionados ao tema. O primeiro mestrado é responsável por aplicar o TLBA em ambientes de *Grid Computing*, trabalho que apresenta um protótipo multiplataforma do TLBA escrito em Java. O segundo é responsável por realizar melhorias contínuas na implementação do protótipo do TLBA. E, finalmente, o terceiro, visa construir um sistema de monitoramento de carga e execução de aplicações.

As contribuições apresentadas neste projeto também motivam, como trabalho futuro, a construção de um modelo analítico do algoritmo TLBA. Esse modelo resultaria em equações matemáticas para predizer o comportamento do TLBA, eliminando a necessidade de realizar simulações.

REFERÊNCIAS

[1] KRUEGER, P. and LIVNY, M. (1987). The Diverse Objectives of Distributed Scheduling Policies. In: SEVENTH INT'L CONF. DISTRIBUTED COMPUTING SYSTEMS, IEEE CS Press, Los Alamitos, Califórnia. *Anais*... Order N.o 801 (microfiche only), p.242-249.

[2] SHIVARATRI, N. G., KRUEGER, P. and SINGHAL, M. (1992). Load Distributing for Locally Distributed Systems. *IEEE Computer*, p. 33-44, dez.

[3] SINHA, P. K. (1997). Distributed Operating Systems: Concepts and Design. John Wiley & Sons.

[4] MELLO, R. F. Et al. (2002). Model for Resources Load Evaluation in Clusters by Using a Peer-to-Peer Protocol, In: THE 2002 INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, Las Vegas, Estados Unidos. *Anais*... jun.

[5] MELLO, R. F., PAIVA, M. S., TREVELIN, L. C. (2002). OpenTella: A Peer-to-Peer protocol for the load balancing in a system formed by a cluster from clusters, In: 4TH INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTING ISHPC-IV, Japão. *Anais*... mai.

[6] MELLO, R. F., PAIVA, M. S., TREVELIN, L. C. (2002). Analysis on the Significant Information to Update the Tables on Occupation of Resources by Using a Peer-to-Peer Protocol, In: 16TH ANNUAL INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE COMPUTING SYSTEMS AND APPLICATIONS, Moncton, New-Brunewick, Canadá. *Anais*... jun.

[7] MELLO, R. F., PAIVA, M. S., TREVELIN, L. C. (2002). A Java Cluster Management Service, In: IEEE INTERNATIONAL SYMPOSIUM ON NETWORK COMPUTING AND APPLICATIONS, Cambridge, Massachussets, Estados Unidos. *Anais*... fev.

[8] MELLO, R. F., TREVELIN, L. C., PAIVA, M. S. (2003). 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*, ISSN 1386-7857, Kluwer.

[9] MELLO, R. F., TREVELIN, L. C., PAIVA, M. S. (2003). 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 (IJHPCN)*, ISSN 1740-0562.

[10] BUYYA, R. (1999). High Performance Cluster Computing – Architecture and Systems. Volume 1. Prentice Hall.

[11] MULLENDER, S. J. (1993). Distributed Systems. Addison-Wesley.

[12] COULORIS, G.; DOLLIMORE, J.; KINDBERG, T. (2000). Distributed Systems: Concepts and Design. Addison-Wesley.

[13] TANEMBAUM, A. (1994). Distributed Operating Systems. Prentice Hall.

[14] CASAVANT, T. L.; KUHL, J.G. (1988). Taxonomy of Scheduling in Distributed Computing Systems. *IEEE Transactions on Software Engineering*, v. 14, n.2, p. 141-154, fev.

[15] KOSTIN, A.E.; AYBAY, I.; OZ, G. (2000). A Randomized Contention-Based Load-Balancing Protocol for a Distributed Multiserver Queuing System. *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n.12, p. 1252-1273, dez.

[16] ZHOU, S.; FERRARI, D. (1987). An Experimental Study of Load Balancing Performance, Report N.o UCB/CSD 87/336, *PROGRES Report N.o 86.8, Computer Science Division (EECS)*, Universidade da Califórnia, Berkeley, California 94720, jan.

[17] HUI, C.; CHANSON, S.T. (1999). Hydrodynamic Load Balancing. *IEEE Transactions on Parallel and Distributed Systems*, v. 10, n.11, p. 1118-1137, nov.

[18] MITZENMACHER, M. (2000). How Useful Is Old Information? *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n.1, p. 6-20, jan.

[19] AMIR, Y. et al. (2000). An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster. *IEEE Transactions on Parallel and Distributed Systems*, v. 11, n.7, p. 760-768, jul.

[20] HENDERSON, R.L. (1995). Job Scheduling under the Portable Batch System, In: JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING: IPPS'95 WORKSHOP. *Anais*... p.280-294, abr.

[21] FIGUEIRA, S.M.; BERMAN, F. (2001). A Slowdown Model for Applications Executing on Time-Shared Clusters of Workstations. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n.6, p. 653-670, jun.

[22] ZOMAYA, A.Y.; TEH, Y. (2001). Observations on Using Genetic Algorithms for Dynamic Load-Balancing. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n.9, p. 899-911, set.

[23] ZHANG, Y. (2001). Impact of Workload and System Parameters on Next Generation Cluster Scheduling Mechanisms. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n.9, p. 967-985, set.

[24] MITZENMACHER, M. (2001). The Power of Two Choices in Randomized Load Balancing. *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n.10, p. 1094-1104, out.

[25] GOSWAMI, K.K.; DEVARAKONDA, M.; IYER, R.K. (1993). Prediction-Based Dynamic Load-Sharing Heuristics. *IEEE Transactions on Parallel and Distributed Systems*, v.4, n.6, p.638-648, jun.

[26] WILLEBEEK-LEMAIR, M.H.; REEVES, A.P. (1993). Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, v.4, n.9, p.979-993, set.

[27] GONZALEZ, R.C.; WOODS, R.E. (2000). Processamento de Imagens Digitais. Edgard Blücher.

[28] NI, L.M.; XU, C.; GENDREAU, T.B. (1985). A distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engineering*, Edição Especial, p.1153-1161.

[29] RYOU, J.; WONG, J.S.K. (1989). A task migration algorithm for load balancing in a distributed system, In: XXII ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES. *Anais*... p.1041-1048, jan.

[30] SUEN, T.T.Y.; WONG, J.S.K. (1992). Efficient task migration algorithm for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, v.3, n.4, p.488-499.

[31] TIEMEYER, M.; WONG, J.S.K. (1998). A task migration algorithm for heterogeneous distributed computing systems. *The Journal of Systems and Software*, Elsevier Science, v. 41, n. 3, p.175-188.

[32] 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.

[33] FERGUSON, D.; YEMINI, Y.; NIKOLAU, C. (1988). Microeconomic Algorithms for Load Balancing in Distributed Computer Systems, *In Proc. VII Int. Conf. on Distributed Computer Systems,* p. 491-499, 1988.

[34] 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.

[35] STEENKISTE, P. (1996). Network-Based Multicomputers: A Pratical Supercomputer Architecture. *IEEE Transactions on Parallel and Distributed Systems*, v.7, n.8, p.861-875.

[36] LIVNY, M.; MELMANN, M. (1982). Load Balancing in Homogeneous Broadcast Distributed Systems, In: ACM COMPUTER NETWORK PERFORMANCE SYMP., p.47-55, 1982, Proceedings impressos em uma edição especial da ACM Performance Evaluation Rev., *Anais...* v.11, n.1, p.47-55.

[37] EAGER, D.L.; LAZOWSKA, E.D.; ZAHORJAN, J. (1986). A Comparison of Receiver-Initiated and Sender-Initiated Adaptative Load Sharing. *Performance Evaluation*, v.6, n.1, p.53-68, mar.

[38] DEVARAKONDA, M.V.; RAVISHANKAR, K.I. (1989). Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Transactions on Software Engineering*, v.15, n.12, p.1579-1586, dez.

[39] WU, M. (1997). On Runtime Parallel Scheduling for Processor Load Balancing. *IEEE Transactions on Parallel and Distributed Systems*, v.8, n.2, p.173-186, fev.

[40] HSU, T. et al. (2000). Task Allocation on a Network of Processors. *IEEE Transactions on Computers*, v.49, n.12, p.1339-1353.

[41] PALIS, M.A.; LIOU, J.; WEI, D.S.L. (1996). Task Clustering and Scheduling for Distributed Memory Parallel Architectures. *IEEE Transactions on Parallel and Distributed Systems*, v.7, n.1, p.46-55, jan.

[42] NEERACHER, M. (1998). *Scheduling for Heterogeneous Opportunistic Workstation Clusters*. Tese (Doutorado) - Swiss Federal Institute of Technology, Zurich. 1998.

[43] LIVNY, M. et al. (1997). Mechanisms for High Throughput Computing. In: XXI WORKSHOP ON VECTOR AND PARALLEL COMPUTING. *Anais*... Speedup, p.36-40, mar.

[44] LITZKOW, M.J.; LIVNY, M.; MUTKA, M.W. (1988). Condor – a Hunter of Idle Workstations. In: IEEE VII INT. CONF. ON DISTRIBUTED COMPUTING SYSTEMS. *Anais*... p.104-111.

[45] KRUEGER, P.; CHAWLA, R. (1991). The Stealth Distributed Scheduler. In: IEEE XI INT. CONF. ON DISTRIBUTED COMPUTING SYSTEMS. *Anais*... p.336-343.

[46] ACCETTA, M. et al. (1986). Mach: A New Kernel Foundation for UNÍS Development. In: SUMMER 1986 USENIX CONF. *Anais*... p.93-112.

[47] HENDERSON, R.L. (1995). Job Scheduling under the Portable Batch System. In: JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING: IPPS'95 WORKSHOP. *Anais*... p.280-294, abr.

[48] CHAPIN, S.J. (1993). *Scheduling Support Mechanisms for Autonomous, Heterogeneous, Distributed Systems*. Tese (Doutorado) - Universidade de Purdue. 1993.

[49] STERLING, T. (2000). *An Introduction to PC Clusters for High Performance Computing*. Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[50] APON, A.; BAKER, M. (2000). *Network Technologies*. Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[51] CHAPIN, S.; WORRINGEN, J. (2000). *Operating Systems.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[52] BUYYA, R.; CORTES, T.; JIN H. (2000). *Single System Image.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[53] BAKER, M.; APON, A. (2000). *Middleware.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[54] SKJELLUM, A. et al. (2000). *System Administration.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[55] SCHIKUTA, E.; WANEK, H. (2000). *Parallel I/O.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[56] PRAMANICK, I. (2000). *High Availability.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[57] BADER, D.A.; PENNINGTON, R. (2000). *Applications.* Disponível em: <http://www.csm.port.ac.uk/~mab/tfcc/WhitePaper/>. Acesso em: 10 out. 2001.

[58] BERMAN, F.; FOX, G.; HEY, T. (2003). Grid Computing: Making The Global Infrastructure a Reality. John Wiley & Sons.

[59] FOSTER, I.; KESSELMAN, C. (1998). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann.

[60] ABBAS, A. (2003). Grid Computing: A Practical Guide to Technology and Applications. Charles River Media.

[61] CECHIN, S.L.; NETTO, J.C. (2001). Ferramentas de Avaliação de Desempenho de Sistemas Computacionais, In: I ESCOLA REGIONAL DE ALTO DESEMPENHO – ERAD 2001, Gramado, Rio Grande do Sul. *Anais*... SBC, p.127-150, jan.

[62] ALLISON, D. (1995). About Benchmarks and Workloads. Disponível em: <http://umunhum.stanford.edu/embedded/about/about.html>. Acesso em: 15 nov. 2001.

[63] FERRARI, D. (1978). Computer Systems Performance Evaluation. Prentice Hall.

[64] GALENBE, E.; MITRANI, I. (1980). Analysis and Synthesis of Computer Systems. Academic Press.

[65] LAZOWSKA, E. et al. (1984). Quantitative System Performance: Computer System Analysis Using Queueing Networks Models. Prentice Hall.

[66] BRATLEY, P.; FOX, B., SCHRAGEL, L. (1987). A Guide to Simulation. Spring-Verlag.

[67] BANKS, J. (1998). Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice. Interscience.

[68] KLEINROCK, L. (1976). Queueing Systems – Volume II: Computer Applications. John Wiley & Sons.

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

[70] LAVENBERG, S.S. (1983). Computer Performance Modeling Handbook. Academic Press.

[71] SOARES, L.F.G.; LEMOS, G.; COLCHER, S. (1995). Redes de Computadores: das LANs, MANs e WANs às Redes ATM. Editora Campus.

[72] NETTO, P.O.B. (1979). Teoria e Modelos de Grafos. Edgard Blücher.

[73] MAXWELL, S. (2000). Kernel do Linux. Makron Books.

[74] Web Site do GNU/GCC (2003). Disponível em: <http://gcc.gnu.org/>. Acesso em: 25 jun.

[75] Plataformas Suportadas pelo GNU/GCC (2003). Disponível em: <http://gcc.gnu.org/install/specific.html>. Acesso em: 25 jun.

[76] JAIN, R. (1991). The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons.

[77] Web Site do CKPT (2003). Disponível em: <http://www.cs.wisc.edu/~zandy/ckpt>. Acesso em: 15 jun.

[78] STERN, H; EISLER, M.; LABIAGA, R. (2001). Managing NFS and NIS. O'Reilly & Associates.

[79] TANEMBAUM, A.S. (1997). Redes de Computadores. Editora Campus.

[80] Licença GNU/GPL (2003). Disponível em: <http://www.gnu.org/licenses/gpl.html>. Acesso em: 25 jun.

[81] BARRETT, D.J.; SILVERMAN, R. (2001). SSH, The Secure Shell: The Definitive Guide. O'Reilly & Associates.

[82] FERRERIRA, R. E. (2003). Linux: Guia do Administrador do Sistema. Novatec Editora.

[83] COMER, D. E. (2001). Computer Networks and Internets. Prentice Hall.

[84] MELLO, R.F.; CHIARA, R.; VILLELA, R.T.N. (2002). Aprendendo Java 2. Novatec Editora.

[85] HWANG, K. (1993). Advanced Computer Architecture. McGraw-Hill.

[86] MELLO, R.F. et al. (em fase de publicação). Desenvolvimento de Aplicações na Arquitetura J2EE. Novatec Editora.

[87] DANDAMUDI, S. P.; PIOTROWSKI, A. (1997). A Comparative Study of Load Sharing on Network of Workstations, In: THE INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS, New Orleans. *Anais*... out.

[88] DANDAMUDI, S. P. (1998). Sensitivity Evaluation of Dynamic Load Sharing in Distributed Systems, *IEEE Concurrency*, v.6, n.3, p.62-72, jul-set.

[89] BARAK, A.; LA'ADAN, O. (1998). The Mosix Multicomputer Operating Systems for High Performance Cluster Computing, *Journal of Future Generation Computer Systems*, v.13, n.4-5, p. 361-372, mar.

APÊNDICE A – Configuração do Ambiente Homogêneo Usado para Executar o Protótipo

Para interconexão dos computadores, foi utilizado um Hub Encore 100Mbits. O ambiente heterogêneo é composto por sete computadores, com as características que seguem.

TABELA 16 – Sistema Operacional

Linux version 2.4.18 2cl (root@mapi2.distro.conectiva) (gcc version 2.95.3 20010315 (release)) #1 Mon Mar 11 01:53:17 BRT 2002

TABELA 17 – Características do Processador

Processor	: 0
vendor_id	: AuthenticAMD
cpu family	: 6
model	: 3
model name	: AMD Duron(tm) Processor
stepping	: 1
cpu MHz	: 945.791
cache size	: 64 KB
fdiv_bug	: no
hlt_bug	: no
f00f_bug	: no
coma_bug	: no
fpu	: yes
fpu_exception	: yes
cpuid level	: 1
wp	: yes
flags	: fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx fxsr syscall mmxext 3dnowext 3dnow
bogomips	: 1887.43

TABELA 18 – Características de Memória

MemTotal:	62456 kB
SwapTotal:	514040 kB

TABELA 19 – Demais Características de Hardware

Silicon Integrated Systems [SiS] SiS630 GUI Accelerator+3D
Silicon Integrated Systems [SiS] SiS900 10/100 Ethernet
Silicon Integrated Systems [SiS] SiS PCI Audio Accelerator
Silicon Integrated Systems [SiS] 5513 [IDE]

APÊNDICE B – Configuração do Ambiente Heterogêneo Usado para Executar o Protótipo

Para interconexão dos computadores, foi utilizado um Hub Encore 100Mbits. O ambiente heterogêneo é composto por quatro computadores com as mesmas características apresentadas no apêndice A e três computadores com as características que seguem.

TABELA 20 – Sistema operacional

Linux version 2.4.18-3U8_4cl (root@buildmaster.distro.conectiva) (gcc version 2.95.3 20010315 (release)) #1 Qua Mai 29 20:09:33 BRT 2002

TABELA 21 – Características do Processador

processor	: 0
vendor_id	: AuthenticAMD
cpu family	: 5
model	: 8
model name	: AMD-K6(tm) 3D processor
stepping	: 12
cpu MHz	: 427.758
cache size	: 64 KB
fdiv_bug	: no
hlt_bug	: no
f00f_bug	: no
coma_bug	: no
fpu	: yes
fpu_exception	: yes
cpuid level	: 1
wp	: yes
flags	: fpu vme de pse tsc msr mce cx8 pge mmx syscall 3dnow k6_mtrr
bogomips	: 851.96

TABELA 22 – Características de Memória

MemTotal:	28176 kB
SwapTotal:	64256 kB

TABELA 23 – Demais Características de Hardware

Silicon Integrated Systems [SiS] 6306 3D-AGP
Davicom Semiconductor, Inc. Ethernet 100/10 MBit
Silicon Integrated Systems [SiS] 5513 [IDE]

APÊNDICE C – Artigos Publicados e Aceitos para Publicação

APÊNDICE D – Códigos-Fonte do Emulador, Simulador e do Protótipo do Algoritmo TLBA