Um Novo Simulador para o Algoritmo de Balanceamento de Carga denominado *Tree Load Balancing Algorithm*

Evandro Raphaloski[1], Maria Stela Veludo de Paiva[2]

[1]Divisão de Desenvolvimento Eletrônico – SMAR Equipamentos Industriais Ltda
Rua Dr. Antônio Furlan Jr., 1028 – 14170-480 – Sertãozinho – SP – Brasil

[2]Escola de Engenharia de São Carlos – Universidade de São Paulo (EESC-USP)
Av. Trabalhador Sãocarlense, 400 – 13566-590 – São Carlos – SP – Brasil

[1]evandro@smar.com.br, [2]mstela@sel.eesc.usp.br

Resumo. Este trabalho apresenta a implementação de um novo simulador para um estudo mais abrangente dos parâmetros de balanceamento de carga do *Tree Load Balancing Algorithm* (TLBA), com base em amostras estatisticamente geradas e maior fidelidade em suas políticas de balanceamento. Suas novas características possibilitam simulação e depuração de algumas variáveis do programa, visualização da árvore computacional de acordo com as capacidades relativas, resultados em tabelas, gráficos e uma análise aplicada a diferentes tipos de escalonamento e sistemas.
Palavras-chave: Algoritmos de balanceamento de carga, TLBA, Modelos de carregamento, Alto desempenho e Simuladores.

A New Simulador for a Load Balancing Algorithm called Tree Load Balancing Algorithm

Abstract. This work shows the implementation of a new simulator for detailed studies of the Tree Load Balancing Algorithm (TLBA) parameters, based on samples statistically generated, and higher fidelity on the implementation of its load balancing policies. Its new features allow simulations and debug of some variable of the *software*, visualization of the computational tree according to their relatives capacities, results displayed on tables, graphics and an analysis applied to different kinds of scheduling and systems.
Keywords: Load Balancing Algorithms, TLBA, Workload models, High Performance and Simulators.

(Received November 29, 2005 / Accepted July 27, 2006)

1. Introdução

O desenvolvimento inicial desse simulador teve como base o emulador e o simulador apresentados por [15], implementado em Linguagem de Programação da [19], utilizada no desenvolvimento de sistemas orientados a objeto [3] e respeitado os termos especificados na [1].

O novo simulador (TLBASim) foi implementado em linguagem de programação C#, devido aos recursos disponíveis e de forma à atender às políticas de balanceamento de carga do *Tree Load Balancing Algorithm* (TLBA) (veja Seção 2) e incorporar novos recursos e características para a sua melhoria.

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

O objetivo da simulação é o de avaliar sistemas e seus comportamentos com um menor custo e maior flexibilidade de variação dos parâmetros para encontrar as condições que conduzam a um melhor desempenho. Isto se aplica ao TLBA através da variação de seus parâmetros e identificação de casos aos quais ele apresenta um melhor desempenho.

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

Este trabalho é composto pela Seção 2 com uma breve explicação do TLBA e suas políticas de balanceamento; na Seção 3 é apresentado o modelo de carregamento de carga utilizado no simulador; a Seção 4 traz uma breve descrição dos componentes

utilizados no simulador; a Seção 5 apresenta parte de seus recursos e interfaces gráficas; a Seção 6 apresenta a validação e algumas casos de estudos; Seção 7 contém as conclusões; a Seção 8 agradecimentos dos autores e a Seção 9 as referências.

2. *Tree Load Balancing Algorithm*

O TLBA foi projetado de forma a organizar computadores de um determinado sistema em uma topologia lógica na forma de árvore, independentemente da topologia física da rede de computadores, sobre a qual são executadas operações de balanceamento de carga.

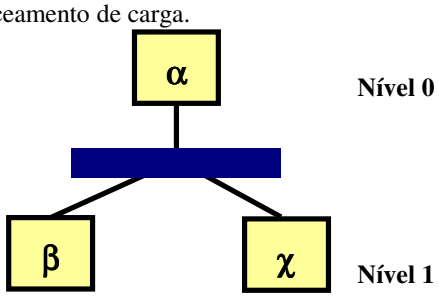

Figura 1: Sub-árvore de um computador raiz

Na Figura 1 é apresentada uma sub-árvore formada por um computador raiz (α) e dois sucessores (ß e χ). Cada computador de um nível *N* envia a seu índice de carga atualização para o seu antecessor no nível *N-1*. O computador raiz é localizado no nível zero da árvore [16].

A busca por um receptor ou pelo computador mais ocioso do sistema é dada da forma *top-down*, partindo do computador raiz até o último nível da árvore computacional.

O TLBA contém dois componentes fundamentais: o *Broker* e o *Peer* [15].

O *Broker* é o componente responsável pela montagem de toda a árvore de computadores interconectados. Ele define em que nível da árvore um computador será inserido, qual seu antecessor e quais seus sucessores.

O Peer é responsável por requisitar a adição de um computador na árvore e implementa as políticas de balanceamento de carga do algoritmo TLBA. O *Peer*, periodicamente, analisa a ocupação do recurso de CPU e memória principal e em seguida calcula o índice de carga. Esse evento periódico tem intervalos definidos pelo administrador do sistema.

As políticas de balanceamento de carga do TLBA são descritas a seguir.

2.1. Política de Informação

Essa política 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 são geradas periodicamente ou sob demanda [15].

O índice de carga define a capacidade de cada computador ou processador no ambiente e são utilizados por essa política para localizar o receptor de processos, ou seja, o computador mais ocioso do sistema. No TLBA o índice de carga é baseado em análises da ocupação percentual de CPU e memória utilizada pelos processos.

Os índices de cargas calculados por um computador β localizado no último nível *N* da árvore são propagados para seu antecessor α localizado no nível *N-1* [16]. Se o computador α não fosse o computador raiz seu índice de carga e de seus sucessores seria armazenado em memória e haveria, ainda, um somatório de seu índice e dos sucessores, que seria submetido ao seu antecessor na árvore. Essa operação seria propagada até chegar à raiz da árvore, atualizando as informações do sistema [15].

Essas informações são propagadas somente se o novo índice de carga superar a um determinado valor percentual de *Threshold*. Essa definição do *Threshold* permite um menor número de mensagens relacionadas à atualização de carga no sistema e, conseqüentemente, uma menor sobrecarga nos meios de comunicação aumentando a estabilidade do sistema [15][18].

2.2. Política de Localização

A política de localização do algoritmo TLBA privilegia localizar computadores mais próximos das "folhas", ou seja, computadores presentes no último nível *N* da árvore [15]..

Para sistemas com baixas cargas, as mensagens de busca em profundidade tendem a percorrer todos os níveis e localizar computadores receptores próximos do nível *N*.

Já em altas cargas, os receptores estão mais próximo ao computador raiz e, portanto, há um menor consumo de recursos do ambiente.

Uma vez localizado o melhor receptor de processos, seu endereço é enviado ao componente responsável pela ativação da política de transferência, que pode ser o *broker* ou um computador sobrecarregado.

2.3. 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 [15].

No algoritmo TLBA a política de transferência como mencionado anteriormente pode ser ativada por dois componentes do sistema [15]:

- ***Broker*** se recebeu a requisição para executar um novo processo no ambiente e ativou a política de localização; e
- **Computador sobrecarregado** se ativou a política de localização para transferir parte de sua carga.

Outra característica da política de transferência do TLBA é que um processo não pode ser transferido mais de κ vezes no ambiente, evitando com que ocorra transferência cíclica de processos entre as máquinas [15]

Após o recebimento do endereço do receptor, o novo processo é transferido. Esse contexto de transferência de processos envolve a localização do computador mais ocioso, seleção de processo e uma etapa denominada *checkpointing* [20].

O *checkpointing* salva o contexto de um determinado processo, para que seja reiniciado em um outro computador [5]. A última etapa é a da transferência propriamente dita.

2.4. Política de Seleção

A política de seleção de um algoritmo de balanceamento de carga é responsável por selecionar processos para transferência.

No algoritmo TLBA essa política busca pelo processo que ocupa maior quantidade dos recursos de CPU e memória em um computador sobrecarregado α.

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 [15].

3. Modelo de Carregamento de Cargas

Uma das formas de se avaliar algoritmos de balanceamento de cargas sem a implementação total de um protótipo é através da simulação. Naturalmente, a qualidade de seus resultados está relacionada aos parâmetros de entrada utilizados na simulação. Assim sendo, um dos fatores fundamentos a serem considerados é a escolha do modelo de carregamento de cargas [9].

Alguns autores afirmam que não existe informação confiável sobre carregamento de cargas em máquinas paralelas [6][11][12][14]. No entanto, de acordo com [9] isso não é verdade. Assim como sistemas uniprocessados, a maioria dos sistemas paralelos mantém o rastreamento de todas as tarefas em execução em um sistema. A análise deste rastreio fornece informações significantes com relação a distribuição de cargas no sistema.

Nos estudos realizados por [9] são utilizados as informações obtidas de 6 diferentes tipos de sistemas apresentados a seguir e com distribuição real de cargas.

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

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

Esse estudo teve como finalidade produzir um modelo de carregamento de cargas que refletisse a de um sistema real e pudesse ser utilizado em simulações com escalonamento paralelo [9]

De forma a complementar esses estudos [10] observaram que execuções repetidas da mesma aplicação tendem a apresentar padrões similares de consumo de recursos e que informações relativas ao comportamento das aplicações, podem ser descobertas sem a cooperação explícita do usuário, através da observação do histórico de execuções passadas.

Os dados obtidos da simulação desse modelo de carregamento de cargas foram utilizados como parâmetros de entrada do simulador apresentado nesse trabalho e foram utilizados para determinar os tempos iniciais de chegada e execução dos processos no sistema.

4. Componentes do Simulador

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

4.1. Classes Relacionadas ao TLBA

Nesta seção são apresentadas as classes que fazem parte do desenvolvimento do TLBA propriamente dito. Elas auxiliam na elaboração de todo o sistema de balanceamento, atualização de carga, cálculo dos índices de carga etc. Existem quatro classes fundamentais para o funcionamento do algoritmos.

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

Peer: é a classe responsável pela aplicação das politicas do TLBA e contém métodos para migração e adição de novos nós nos escalonadores de cada computador, cálculos da ocupação da CPU, Memória e Índice de Carga. O escalonador nada mais é do que uma instanciação desse componente para classe *Computer*.

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

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

4.2. Projetos Auxiliares Utilizados no Simulador

O novo simulador contempla recursos adicionais ao simulador apresentado por [15], sendo de caráter didático para introduzir conceitos básicos relacionados a ambientes distribuídos e algoritmos de balanceamento de cargas.

Para o desenvolvimento de suas interfaces, foram utilizados alguns projetos auxiliares que são descritos a seguir. Esses projetos ofereceram recursos tais como: tabela com *grids* e funções de copiar, colar, ordenar linhas etc; gráficos de barras, colunas e linhas; montagem gráfica da árvore de computadores entre outros recursos.. A maioria dos projetos são encontrados em [7][8].

Os principais projetos auxiliares utilizados são descritos a seguir.

SourceGrid: um Projeto *Open Source* desenvolvido por [2]. Esse projeto é responsável pela tabela com *grids*, onde os dados finais das simulações são apresentados..

Lithium Tree Control: também conhecido como *Netron Project*. É responsável pela diagramação da árvore de computadores de acordo com suas capacidades computacionais. Foi a princípio desenvolvido para edição e visualização de dados em XML ou qualquer árvore de dados estruturados [17].

***TSWizard*:** um *framework* para .NET utilizado para a implementação do *Wizard* para a montagem final dos gráficos e foi desenvolvido por [4].

***ZedGraph*:** esse projeto foi utilizado para a elaboração dos gráficos apresentados no simulador em função dos resultados obtidos das simulações e disponíveis em tabelas [7].

***WinFormsUI*:** Essa biblioteca é responsável pela elaboração de todo o ambiente de trabalho. Possui recursos de movimento e posicionamento da janelas principais e secundários do simulador entre outras propriedades [13].

4.3. Diferenças entre os Simuladores

Nesta Seção são apresentadas as principais diferenças entre o simulador proposto neste trabalho e o implementado por [15].

1) **Simulação em ambientes diversificados**, dando um enfoque maior às características do ambiente, desde ambientes totalmente homogêneos até ambientes totalmente heterogêneos e, também, levando-se em conta o tamanho do sistema. O simulador implementado por [15] permite a análise de apenas três casos de estudos (com relação aos níveis da árvore computacional do TLBA) e pouca diversificação do ambiente de *hardware*.
2) **Interface gráfica amigável** com o usuário, facilitando a alteração dos parâmetros de entrada e a visualização dos resultados na própria janela onde se encontram os parâmetros, além de externar em arquivos resultados que demandam um maior processamento. O simulador implementado por [15] anterior executava no modo *Console* e para conhecer os parâmetros de entrada era necessário um conhecimento prévio do funcionamento do simulador.

3) **Abordagem de um maior número de parâmetros**. O implementado por [15] leva em consideração os seguintes parâmetros: capacidade de processamento por computador, ocupação da CPU por processos, números de processos no sistema e geração aleatória do tempo de chegada de cada processo através da distribuição de Poisson. Além disto, utiliza como índice de carga o número de processos na fila de espera do escalonador e não leva em consideração o percentual de ocupação de memória pelos processos. Já o simulador aqui proposto leva em consideração não só os parâmetros citados anteriormente, mas também os apresentados a seguir.
   a) Cálculo do índice de carga, baseado em percentuais da ocupação da CPU e memória, para os quais diferentes taxas de ocupação são atribuídas a cada um desses elementos (CPU e memória), verificando-se, assim, qual deles causaria uma maior influência no desempenho do sistema;
   b) Intervalos de confiança e desvios padrão;
   c) *Thresholds* de viabilidade de transferência de processos e de atualização dos índices de cargas;
   d) Número de mensagens condizentes com as políticas de balanceamento de carga do algoritmo;
   e) Cálculo das capacidades ociosas e seus complementos tanto em sistema com ou sem sucessores;
   f) Cálculo das variações superiores e inferiores que permitem verificar mudanças significativas nos índices de carga dos computadores; e
   g) Cálculo da média das capacidades dos computadores sucessores ao computador raiz, possibilitando a verificação da viabilidade de migração de processos.
4) **As Bases estatísticas fundamentadas em estudos prévios** e os dados estocásticos foram obtidos da monitoração e coleta de informação de sistemas reais. Nesse último caso, os tempos de execução e chegada de processos são baseados no trabalho de [9] e o número de interações é dependente do intervalo de confiança e erro percentual sobre a média estabelecida. O simulador anterior baseia-se na geração de dados randômicos e aleatórios.
5) **Sistema de escalonamento** no simulador proposto tende a uma maior similaridade com os sistemas reais, exceto pelas *threads*, porém com bases conceituais muito similares, em que um processo, ao chegar no sistema, entra em execução caso não haja tarefas pendentes e seguindo a política de escalonamento de cada sistema. No novo simulador foram implementadas quatro políticas de escalonamento: *Round-Robin, Shortest Job First, First-come First-served* e por prioridades.

5. Interfaces do Novo Simulador (TLBASim)

As principais interfaces gráficas do TLBASim são apresentadas nas Figuras 1 a 3. A princípio consiste de um janela principal e várias janelas filhas. O projeto *WinformsUI* [13], utilizado para a elaboração dessas interfaces permite a movimentação dessas janela filhas em diversas parte da janela principal. Isso possibilita uma maior flexibilidade para o usuário de trabalhar com o ambiente e posicionar a telas relevantes em determinados momentos da simulação.

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

A parte da janela principal e suas principais janelas filhas - janelas onde são apresentados as tabelas com resultados das simulações (Figura 1, parte superior-lado esquerdo), gráficos que podem ser gerados a partir da tabela com os resultados (Figura 2), a árvore com a diagramação dos computadores distribuídos durante a simulação (Figura 3) e uma a janela para edição de texto - tem-se também as janelas com as propriedades do simulador.

As propriedades do simulador são constituídas de parâmetros configuráveis para cada simulação em específico. Possuem parâmetros de balanceamento de carga, ambientes de redes, dados estatísticos etc.

Uma outra janela filha que merece destaque e a de *Logging* (Figura 1, parte inferior). Essa janela traz os *logs* de alguns dos principais pontos do simulador, que depuram o código e os resultados durante cada simulação, podendo-se encontrar falhas ou não, no simulador.

Existem ainda outras janelas filhas com menor importância ao TLBASim, como a *Task List* e *Toolbox*, que estão presentes na janela principal, mas não têm nenhuma funcionalidade. Como parte de outros estudos elas podem ser implementadas.

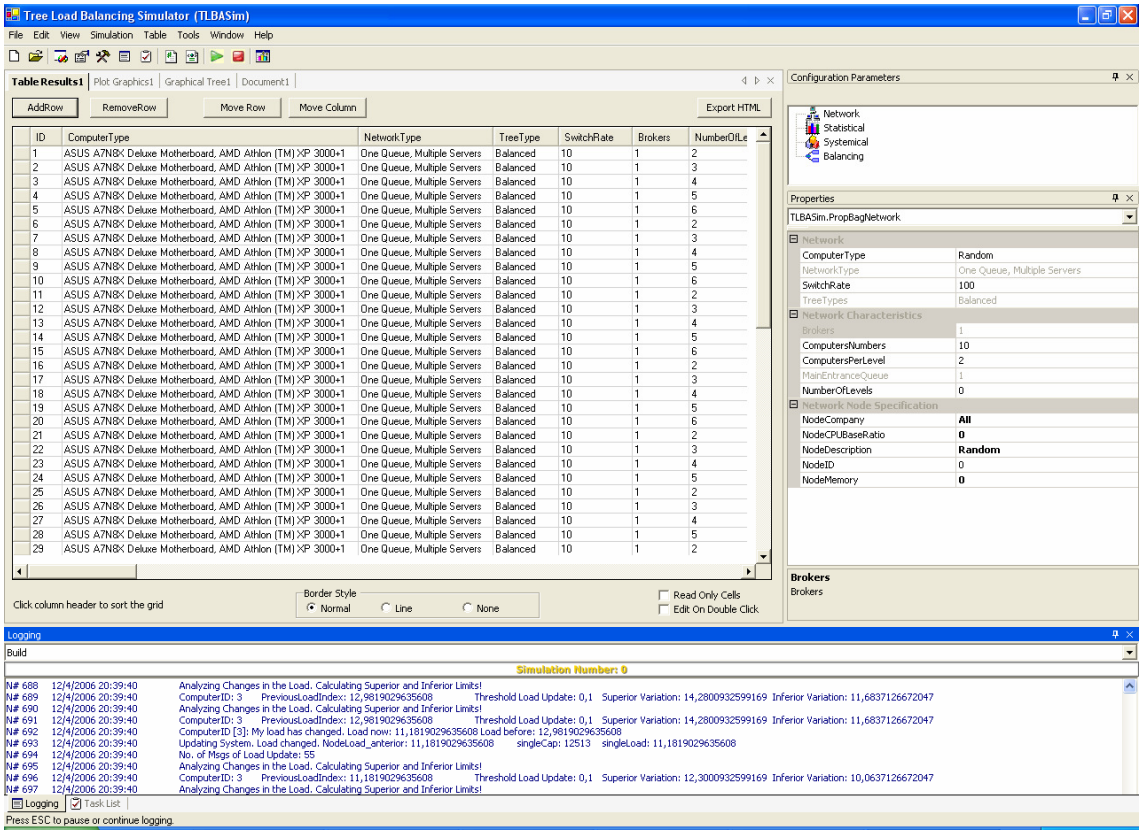

Figura 1: Interface inicial do simulador com a tabela de resultados (parte superior - lado direito), janela de classes de parâmetros (parte superior - lado esquerdo) e janela de mensagens (parte inferior).

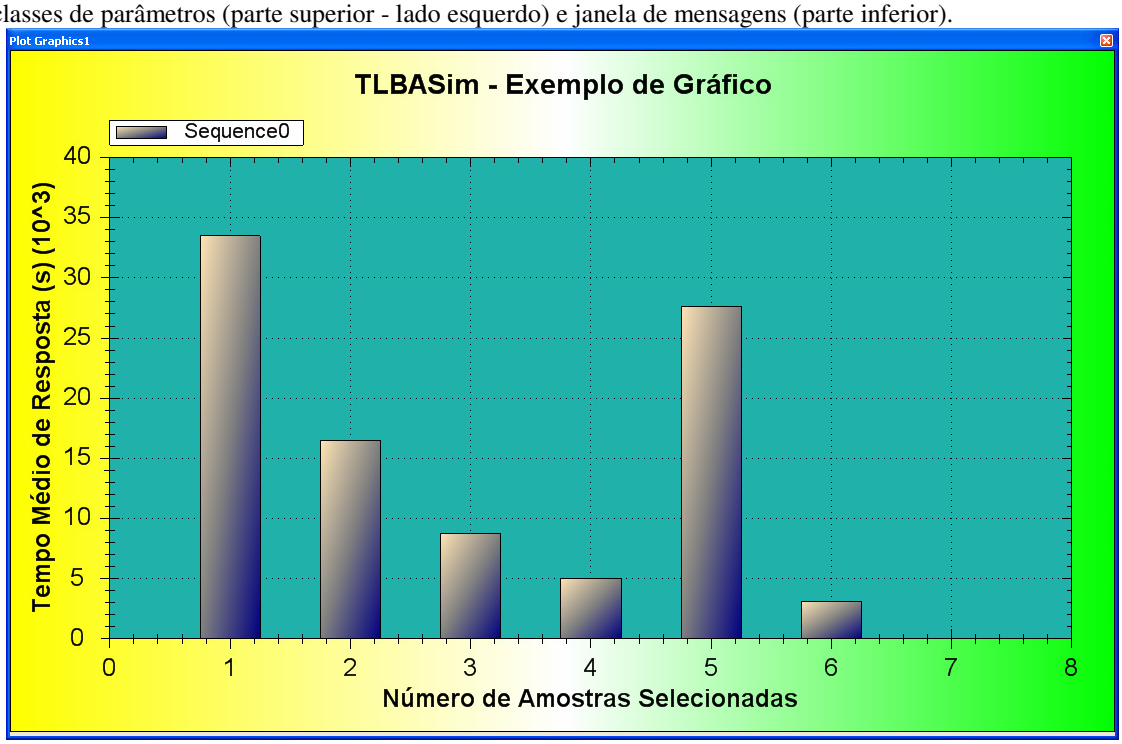

Figura 2: Exemplo de um gráfico de colunas elaborado pelo TLBASim.

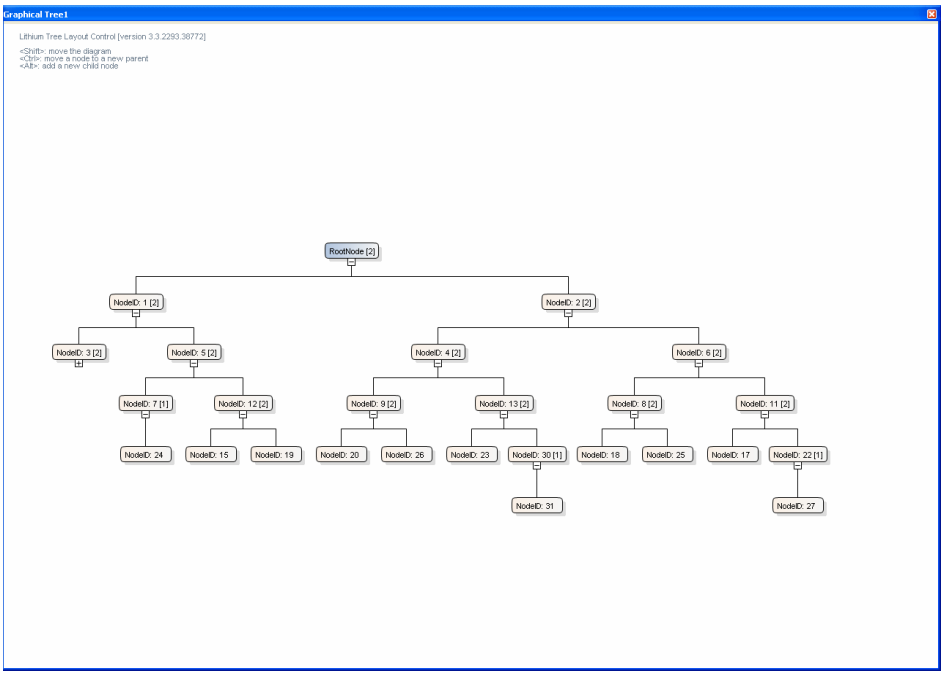

Figura 3: Montagem da árvore computacional durante processo de simulação.

6. Validação e Casos Estudados

O novo simulador (TLBASim) será validação com a utilização da comparação de seus resultados com o do antigo simulador (TLBA V1.0), cujos resultados foram validados através de um protótipo construído por [15].

As simulações apresentadas neste trabalho foram geradas com intervalos de confiança de 95%, excetuando-se as simulações para comparação com o simulador TLBA V1.0 de forma a manter o código original desse simulador e que não continha tal recurso.

6.1. Análise do TLBA V1.0 e do TLBASim em Função da Variação do Número de Processos no Sistema (Validação)

No gráfico da Figura 4 pode-se observar a sobrecarga do sistema em função do número de mensagens geradas por nível à medida que se têm um aumento no número de processos submetidos ao sistema. Assim sendo, um sistema com um baixo número de computadores e um elevado número de processos estará sobrecarregado em virtude do número de mensagens (Figura 4) e terá um maior tempo médio de resposta para cada processo (Figura 5).

Nestes gráficos, TLBA V1.0 e TLBASim(1) apresentam os resultados do antigo e novo simulador, respectivamente, e levam em consideração apenas as mensagens de balanceamento do algoritmo para cada nível dos processos submetidos ao sistema. Já o TLBASim(2) considera o número total de mensagens geradas no sistema.

Essa última análise só é possível porque o novo simulador faz a análise de mensagens que são geradas quando um computador sobrecarregado solicita ao computador raiz, informações do sistema para transferência de seus processos, mensagens de atualização de carga, mensagens de balanceamento de carga, mensagem requisitando execução de processos no sistema e mensagens para inserir um novo computador na árvore.

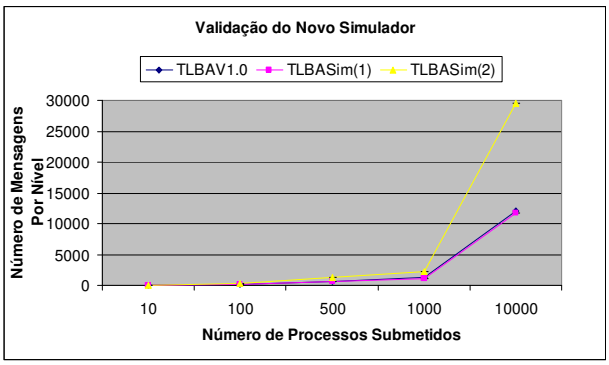

Figura 4: Variação do número de mensagens em função do número de processos

A divergência entre os valores apresentados do novo simulador e do simulador de [15] deve-se ao fato de que esse último considera fixa a ocupação percentual de cada

processo no sistema (que é da ordem de no mínimo 10^7) o que nem sempre representa o comportamento dos sistemas reais e eleva, então, o valor final do tempo de resposta de cada ciclo de execução.

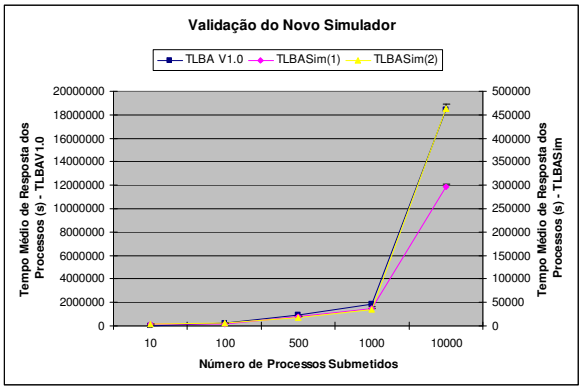

Figura 5: Tempo médio de resposta (s) dos processos em função do número de processos enviados.

Já o novo simulador assume valores de tempos de chegadas de processos de acordo com o modelo de [7], capacidades computacionais obtidas de especificações de sistemas reais e sua ocupação percentual varia para cada computador.

6.2. Análise da Variação dos *Thresholds* de Viabilidade de Transferência e Atualização de Carga

No gráfico das Figuras 6 e 7 é apresentado o número de mensagens contidas em função da variação dos *Thresholds*. Esses resultados mostram que para um sistema em processo de balanceamento de cargas é necessária uma maior preocupação com relação ao número de mensagens geradas pela política de localização que busca pelo computador mais ocioso do sistema do que demais mensagens.

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

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

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

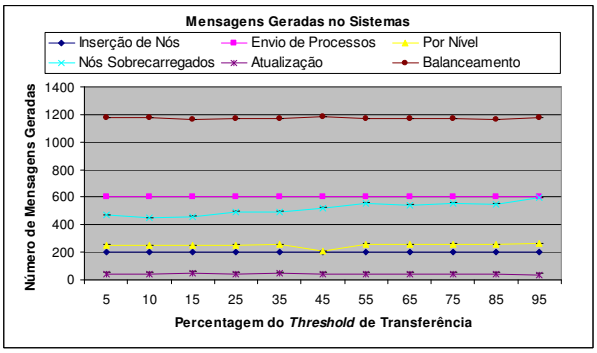

Figura 6: Número de mensagens geradas em função da variação do *Threshold* de transferência

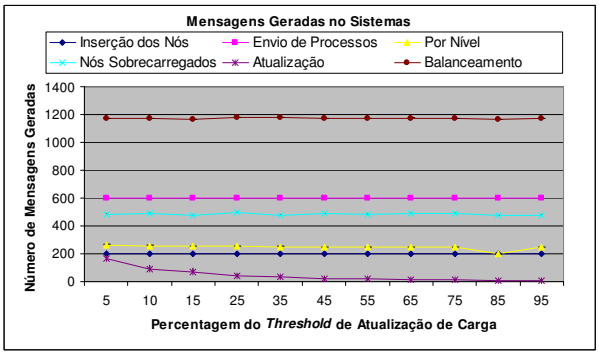

Figura 7: Número de mensagens por nível geradas em função da variação do *Threshold* de atualização de carga

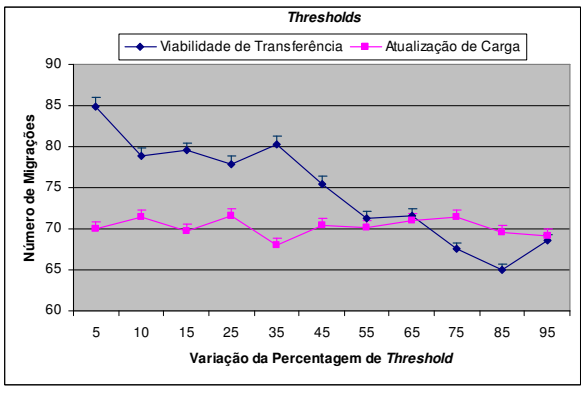

Figura 8: Número de migrações realizadas durante a variação dos *Thresholds* de transferência e atualização de carga.

6.3. Análise de Mensagens em Sistemas Homogêneos e Heterogêneos

As simulações dessa seção permitem analisar o TLBA para um ambiente composto por computadores homogêneos e heterogêneos (Tabela 1).

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

Já para os sistemas heterogêneos a capacidade computacional desses computadores variou aleatoriamente em função do número de computadores configurados.

Nº Nós	Mens. Por Nível	Mens. Comp. Sobrecar-regado	Mens. Atuali-zação	Mens. Balan-ceamento
SISTEMAS HOMOGÊNEOS				
10	1263,78 ± 3,56	552,42 ± 4,70	4,61 ± 2,03	718,65 1,0085
100	278,22 ± 5,30	664,05 ± 8,61	125,79 ± 50,93	1164,57 1,0101
500	101,05 ± 2,47	452,51 ± 8,84	761,62 ± 102,75	1388,00 ± 0
1000	63,72 ± 1,24	555,04 ± 8,67	761,76 ± 102,81	1388,00 ± 0
5000	55,39 ± 3,63	696,13 ± 24,44	7092,63 ± 1294,83	1388,00 ± 0
SISTEMAS HETEROGÊNEOS				
10	1056,49 ± 2,49	272,78 ± 3,13	2,96 ± 1,51	689,00 ± 0,94
100	274,26 ± 3,07	635,16 ± 5,91	105,30 ± 29,10	1174,75 ± 0,76
500	101,37 ± 2,50	455,23 ± 8,54	770,05 ± 104,07	1390,00 ± 0
1000	63,86 ± 1,26	552,43 ± 8,44	773,93 ± 105,15	1390,00 ± 0
5000	64,39 ± 4,41	732,97 ± 22,10	7386,40 ± 369,32	1399,00 ± 0

Tabela 1: Número de Mensagens geradas para sistemas homogêneos e heterogêneos

A Tabela 1 apresenta os resultados do número de mensagens geradas para simulações em ambientes homogêneos e heterogêneos. Através desta tabela, é possível concluir que o TLBA é recomendado para sistemas heterogêneos por apresentar em média um menor número de mensagens geradas para esse sistema, quando comparados aos ambientes homogêneos.

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

7. Conclusões

Este trabalho teve como objetivo a elaboração de um novo simulador para o TLBA para o estudo do efeito da variação de seus parâmetros de balanceamento de carga e da diversificação de aplicação em ambientes homogêneos e heterogêneos.

O TLBASim apresentou uma interface gráfica, coerente com os atuais práticas de desenvolvimento seja na área de sistemas distribuídos, balanceamento de cargas, áreas didática e até mesmo comercial.

Com os recursos presentes no simulador desenvolvidos neste trabalho, será possível a apresentação de conceitos básicos na área de balanceamento de cargas e sistemas distribuídos, tornando-o assim uma forte ferramenta didática.

Seus recursos de tabela e gráfica facilitam ao usuário a interpretação de suas simulações e análise de seus resultados. O recurso de diagramação da árvore lógica auxilia ao usuário analisar o algoritmo e verificar como está se dando a montagem da árvore computacional em tempo real de simulação.

Os resultados comprovam suas funcionalidades baseadas na comparação com o simulador implementado por [15].

Um dos casos estudos foi o da variação dos *Thresholds* de Viabilidade de Transferência e de Atualização onde se pode notar que não interferem significativamente no número de mensagens geradas na rede e no tempo médio de resposta.

Os resultados das simulações em ambientes homogêneos e heterogêneos confirmam os apresentados por [15] através de seu simulador e protótipo, onde se comprova a eficiência do TLBA para ambientes heterogêneos e altamente escaláveis.

8. Agradecimentos

Os autores agradecem à Libânio Carlos de Souza e Délcio Prizon da empresa SMAR Equipamentos Industriais Ltda., pela oportunidade e suporte na realização do Mestrado na Escola de Engenharia de São Carlos (EESC-USP) que resultou no desenvolvimento deste trabalho; ao Prof.Dr. Rodrigo Fernandes de Mello do Instituto de Ciências da Computação e Matemática da Universidade de São Paulo (ICMC-USP), pelas explicações sobre o TLBA, compartilhamento de código fonte, reuniões e comentários que contribuíram para a evolução do simulador; e ao Prof.Dr. Luis Carlos Trevelin do Departamento de Ciências da Computação da Universidade Federal de São Carlos (UFSCAR), pelos valiosos comentários e sugestões sobre o trabalho.

9. Referências

[1] GNU General Public License. Disponível em: <http://www.gnu.org/licenses/gpl.html>. Acesso em: 8 feb. 2004

[2] Icardi, D. *Projeto sourcegrid para apresentação de dados em tabela.* Disponível em: <www.codeproject.com>.Acesso em: 12 sept. 2005.

[3] *Java Technology*. Disponível em: <http://java.sun.com>. Acesso em: 9 oct. 2004.

[4] Johnson, J.T.. *TSWizard – a wizard framework for .NET*. Disponível em: <http://www.codeproject.com/cs/miscctrl/tswizard.asp>. Acesso em: 28 oct. 2005.

[5] Chanchio, K.; Sun, X.H. *A protocol design of comunication state transfer for distributed computing*. 2001, New York: IEEE Computer Society Press, p.715-718.

[6] Chiang, S.H.; Mansharamani, R.K.; Vernon, M.K.. Use of application characteristics and limited preemption for run-to-completion parallel processor scheduling policies. In: SIGMETRICS CONFERENCE OF MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 1994, New York, p.33-44.

[7] *Code Project:the - free source code and tutorials*. Disponível em: <http://www.codeproject.com>. Acesso em: 29 oct. 2005.

[8] *Codeguru:the – number one developer site*. Disponível em: <http://www.codeguru.com>. Acesso em: 18 oct. 2005.

[9] Feitelson, D.G. *Packing schemes for gang scheduling*. Disponível em: <http://www.cs.huji.ac.il/labs/parallel/workload/models.html# feitelson96>. Acesso em: 20 mar. 2006.

[10] Feitelson, D.G.; Jette, M.A.. Improved utilization and responsiveness with gang scheduling. In: WORKSHOP JOB SCHEDULING FOR PARALLEL PROCESSING, 1997, Geneva. *Proceedings...* Berlin: Springer. p.238-261. (Lectures notes in computer science, 1291).

[11] Krueger, P.; Lai, T.H.; Radiya, V.A.. Processor allocation vs. job scheduling on hypercube computers. In: 11th INTERNATIONAL CONFERENCE OF DISTRIBUTED COMPUTING SYSTEMS, 1991, *Proceedings...*, IEEE. p.394-401.

[12] Leutenegger, S.T.; Vernon, M.K.. The performance of multiprogrammed multiprocessor scheduling policies. In: SIGMETRICS CONF. MEASUREMENT & MODELING OF COMPUT. SYST., 1990, p.226-236.

[13] Luo, W. (2004). *Ambiente de Trabalho e Gerenciamento das Telas Gráficas*. Disponível em: <http://www.myxaml.com/wiki/>. Acesso em: 15 sept. 2005.

[14] Majumdar, S.; Eager, D.L.; Bunt, R.B. (1988). *Scheduling in multiprogrammed parallel systems*. IN: SIGMETRICS CONF. MEASUREMENT & MODELING OF COMPUT. SYST., 1988, *Proceedings...* p. 104-113.

[15] Mello, R. F. *Proposta e Avaliação de Desempenho de um Algoritmo de Balanceamento de Carga para Ambientes Distribuídos Heterogêneos Escaláveis*. 130p. Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2003.

[16] Mello, R.F.; Mattos, E.C.T; Trevelin, L.C.; Paiva, M.S.V.; Yang, L.T. Proposal of a Tree Load Balancing Algorithm to Grid Computing Environments. *IEICE Transactions on Information and Systems*, v. E87-D (7), July, 2004.

[17] Netron Project (2005). *Diagramação da Árvore Computacional*. Disponível em: <http://netron.sourceforge.net/wp/>. Acesso em: 28 out. 2005.

[18] Shivaratri, N.G., Krueger, P.; Singhal, M. Load distributing for locally distributed systems. *IEEE Computer*, 1992, New York, v.25, n.12, p.33-44.

[19] Sun Microsystems (2004). Disponível em: <http://www.sun.com>. Acesso em: 9 oct. 2004.

[20] Zandy, V.C. (2004). *CKPT – A process checkpoint library*. Disponível em: <http://www.cs.wisc.edu/~zandy/ckpt>. Acesso em: 17 apr. 2004.