Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e
Automação Industrial - DCA

CPTool:

Uma ferramenta para auxiliar o Planejamento de Capacidade de sistemas computacionais através de simulações

Autor: ANGELO FERECINI NETO

Orientador: José Raimundo de Oliveira

Co-Orientador: Rafael Santos Mendes

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação, da Universidade Estadual de Campinas, para obtenção do título de Mestre em Engenharia Elétrica, área de concentração: Engenharia de Computação.

CAMPINAS

2012

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Ferecini Neto, Angelo
F377c CPTool: uma ferramenta para auxiliar o planejamento de capacidade de sistemas computacionais através de simulações / Angelo Ferecini Neto. -- Campinas, SP: [s.n.], 2012.

Orientadores: José Raimundo de Oliveira, Rafael Santos Mendes.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Computadores - Capacidade - Planejamento. 2. Desempenho. 3. Modelagem e simulação. I. Oliveira, José Raimundo de. II. Mendes, Rafael Santos. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Título em Inglês: CPTool: a *Capacity Planning* tool for computer systems using simulation
Palavras-chave em Inglês: Computers - Capacity - Planning, Performance, Modeling and simulation
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Marcos Jose Santana, Alice Maria Bastos Hubinger Tokarnia
Data da defesa: 14-02-2012
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Angelo Ferecini Neto

Data da Defesa: 14 de fevereiro de 2012

Título da Tese: "CPTool: uma ferramenta para auxiliar o Planejamento de Capacidade de Sistemas Computacionais através de simulações"

Prof. Dr. José Raimundo de Oliveira (Presidente): _____

Prof. Dr. Marcos Jose Santana: _____

Profa. Dra. Alice Maria Bastos Hubinger Tokarnia: _____

Dedico este trabalho
aos meus pais Carlos e Rute,
a quem agradeço a vida e o amor que a mim dedicaram,
à minha irmã Geovana pela companhia durante minha existência.

Agradecimentos

Agradeço aos professores Dr. José Raimundo de Oliveira e Dr. Rafael Santos Mendes pela dedicação fundamental durante a minha orientação, aos meus companheiros de trabalho do CPqD que me deram todo o apoio e facilitaram minha presença nas aulas durante horário comercial, aos meus atuais amigos na empresa Inmetrics pela motivação e auxílio com idéias e a minha familia por todo amor e companherismo nos momentos mais difíceis. Agradeço também a todos aqueles que direta ou indiretamente contribuíram para que este trabalho fosse realizado.

Angelo Ferecini Neto

Resumo

O planejamento de capacidade de sistemas é um método muito útil para predizer qual hardware será necessário para processar um sistema de TI (Tecnologia da Informação) com alto desempenho. A ferramenta proposta foi desenvolvida para facilitar este difícil e importante trabalho, automatizando a análise do software em diferentes ambientes. Através da simulação discreta orientada a eventos, baseada em um modelo de teoria de filas, ela modela a interação entre hardware e softwares, simplificando a análise de qualquer sistema batch de TI. Neste tipo de sistema, normalmente existem muitas transações que necessitam ser processadas pelo sistema sem interação humana e frequentemente são utilizados os benefícios do processamento paralelo para um melhor desempenho. Podemos aplicar estas idéias para representar a carga de trabalho no modelo de simulação simplesmente como processos que entram no sistema para serem processados. Os resultados mostraram que é possível utilizar um modelo para representar o sistema real com uma precisão de até 30%.

Palavras-chave: Planejamento de capacidade, Desempenho e Simulação

Abstract

System *Capacity Planning* is a very useful method to predict which hardware will be necessary to run a high performance IT system. The proposed tool was developed to ease this hard and important job, automating the analysis of the software in different environments. It uses the discrete-event simulation using a queue theory model to represent the hardware and software interaction. The model presented here represents a simplification of any batch IT systems. In this type of system, commonly we have a lot of transactions that need to be processed by the system without human interaction and frequently use the benefits of parallel processing for better performance. We can apply these ideas to represent the workload in the simulation model as simple tasks that enter the system to be processed. Results show that it is possible to use a model to represent the real system with 30% fidelity.

Key-words: *Capacity Planning*, Performance and Simulation

Trabalho Relacionado com esta Dissertação.

A.F. Neto, J.R. de Oliveira and R.S.Mendes "A Capacity Planning Tool for Batch Parallel Processing Systems" In Proceedings of The 19th IASTED International Conference on Modelling and Simulation - *MS 2008,* Quebec City, Canada, May 26 – 28, 2008.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1
Introdução

A atividade de planejamento de capacidade (*Capacity Planning)* consiste em predizer o desempenho de sistemas computacionais considerando padrões distintos de cargas de trabalho (FERRARI, SERAZZI & ZEIGNER, 1983) em servidores com diferentes potenciais de processamento. De modo geral, é utilizada no campo da tecnologia da informação para predizer o comportamento de programas que ainda estão em fase de planejamento ou desenvolvimento e não foram implementados, usando como conhecimento a priori do sistema as provas de conceito e documentações gerais e fazendo uso da teoria de filas e de simulações para prever o comportamento da execução do que será produzido.

Atualmente existem poucas ferramentas disponíveis no mercado que realizam este tipo de trabalho específico e as que já existem normalmente são muito caras e complexas, inviabilizando sua aplicação em empresas de menor porte com baixo capital financeiro. Podemos citar como exemplos de ferramentas disponíveis a TeamQuest e BEZ Vision, além de outras ferramentas proprietárias muitas vezes específicas para o hardware de determinado fabricante. Por outro lado, são disponíveis atualmente diversos aplicativos de simulação geral e bibliotecas constituindo ferramentas de uso muito genérico não focado totalmente à computação. Desta maneira o profissional que não conhece detalhadamente o modelo de desempenho de um sistema computacional, tem dificuldades para realizar o estudo de planejamento de capacidade.

Uma vantagem de se ter uma ferramenta específica para simulação do planejamento de capacidade é que podemos prever o comportamento de um software em um ambiente computacional com tecnologia muito recente que ainda não foi validada por mecanismos de *benchmark* como, por exemplo, os desenvolvidos pela SPEC (System Performance Evaluation Corporation), que é uma organização de fornecedores da indústria de informática que desenvolve testes de desempenho bem padronizados e publica o resultado de suas análises. Muitas vezes quando queremos fazer a comparação do desempenho de duas máquinas é necessário fazer aproximações, pois a definição da carga que a máquina suporta não é bem conhecida.

O objetivo da ferramenta proposta neste trabalho é facilitar toda a atividade do planejamento de capacidade, sendo que o usuário final dela não precisará conhecer técnicas de modelagem e

simulação nem realizar cálculos matemáticos para que o trabalho seja feito com precisão e bom desempenho.

É reconhecido que o planejamento de capacidade através de técnicas de simulação (BANK, 2000; LAW & KELTON, 1991) consegue representar com maior precisão (XU & CHUNG, 2001) um sistema computacional do que cálculos analíticos, mas exige muita potência computacional. Portanto para que se torne viável a utilização no processo de planejamento de capacidade, esta ferramenta deve apresentar um tempo de resposta consideravelmente baixo e utilizar pouco recursos computacionais, permitindo executá-la em um simples computador *desktop*. Pensando neste detalhe, após a implementação e validação funcional, foram realizados testes de desempenho (LILJA, 2000) nesta ferramenta com auxílio de ferramentas de *profiling*, para torná-la a mais rápida possível, viabilizando o seu uso em baixa plataforma.

1.1 Organização do texto

Esta dissertação está organizada da seguinte forma. O Capítulo 2 apresenta uma revisão detalhada dos conceitos fundamentais a serem utilizados. O Capítulo 3 apresenta a proposta de trabalho, o modelo conceitual criado baseado na proposta, conceitos sobre benchmarking que são utilizados para validação de modelo e o processo de validação e análise de resultados do modelo. O Capítulo 4 mostra a aplicação dos conceitos em um caso de estudo através de simulação. O Capítulo 5 apresenta como a ferramenta foi implementada e executada e também a comparação de resultados obtidos através da implementação do modelo na ferramenta de simulação Extend versus ferramenta gerada através de programação. O Capítulo 6 apresenta a conclusão final do trabalho e indica direções para trabalhos futuros.

Capítulo 2
Estado da arte

A demanda de *Capacity Planning* está cada vez maior em grandes empresas do mundo todo para direcionar decisões de compra de recursos computacionais. Diante da crise econômica cada vez mais se torna imprescindível definir bem os investimentos (DHANANJAY, SRINIVASA & KARTHIK, 2009).

Apesar da crise, algumas empresas continuam crescendo e, algumas vezes, é necessário prever crescimentos explosivos de demandas e utilizar arquiteturas que permitam o crescimento de capacidade rapidamente e para todos estes propósitos temos as técnicas e ferramentas do *Capacity Planning* (ALISPAW, 2008).

No Brasil, com o aumento crescente da demanda computacional principalmente nos setores de Telecom e Financeiro, as empresas estão mais preocupadas em garantir a compra adequada do hardware necessário conforme a demanda de negócio (FREIRE, 2011a), evitando assim perdas por clientes descontentes com lentidão e indisponibilidade e consequentemente a degradação de imagem. O custo gerado por esses fatores negativos, em apenas alguns minutos, pode superar o custo de se manter uma equipe especializada no assunto dentro da empresa.

Além disso, as empresas estão cada vez mais preparadas para planos de contingência de seus datacenters e o investimento tem que ser bem direcionado para que não haja gasto indevido com servidores desnecessários (GALVES, 2006). Diante deste cenário, as empresas destes setores estão definindo e criando suas próprias equipes de *Capacity Planning*, formadas por funcionários e consultores de empresas especializadas nesta área.

Existem algumas ferramentas que facilitam este trabalho, tais como a Teamquest e Compuware Gomez. Estas ferramentas se utilizam de técnicas de simulação e teoria de filas para os cálculos realizados neste tipo de atividade. Outro mecanismo utilizado nessas ferramentas é a correlação matemática de métricas. Normalmente são ferramentas que estão sendo desenvolvidas há muito tempo, mas somente agora tiveram sua devida atenção pelo mercado. São ainda ferramentas muito caras que não são acessíveis financeiramente a todas empresas.

Algumas empresas oferecem consultorias na área de *Capacity Planning* e normalmente utilizam ferramentas próprias e do mercado ou simplificam estes cálculos através de correlação e

regressões matemáticas (ALMEIDA, 2011) de métricas de sistema com métricas de infraestrutura. Com isso é possível produzir uma fórmula que representa o consumo de um recurso ou o tempo de resposta, baseado em uma métrica de sistema ou negócio que representa a carga de trabalho no ambiente. (FREIRE, 2011b)

O principal desafio do trabalho de *Capacity Planning* é entender como funciona a carga de trabalho nos sistemas e instrumentar estes para que gerem um histórico de métricas. Além disso, existe uma dificuldade grande em entender como as métricas de sistema, que definem a carga de trabalho, são influenciadas pelo crescimento de negócio da empresa. Cálculos simplesmente matemáticos, não considerando detalhes do sistema, podem gerar resultados errados. Tudo isso é sanado com reuniões e questionários que devem ser respondidos pelos usuários e desenvolvedores do sistema, que são as pessoas que mais entendem do sistema sendo analisado (FREIRE, 2011c).

A ferramenta desenvolvida neste trabalho de mestrado facilita esta atividade de *Capacity Planning*, permitindo ao usuário definir parâmetros específicos de software e hardware e executar uma simulação para verificar o comportamento do sistema. Esta ferramenta pode ser evoluída para contemplar outras tecnologias e tipos de sistema e disseminar mais a cultura do *Capacity Planning* através de simulações para empresas de menor porte também.

O propósito é que esta ferramenta seja disponibilizada livremente, inclusive seu código fonte, para que haja um desenvolvimento colaborativo daqui em diante.

A tendência é que no futuro próximo as empresas conheçam bem cada uma de suas aplicações e o impacto do negócio em sua infraestrutura (PÉREZ, 2010), com isso poderão evitar mais as perdas financeiras e detrimento da imagem com as indisponibilidades e lentidões nos sistemas que afetam direta ou indiretamente os clientes.

Mesmo no caso da popularização do Cloud Computing o planejamento de capacidade será necessário, já que as empresas não possuirão capacidade infinita, principalmente pela limitação de rede, além de ter que estudar qual servidor virtual possui melhor custo-benefício, mesmo períodos que o processamento é mais barato. (TAURION, 2006).

Capítulo 3
Conceitos Fundamentais

Neste capítulo será apresentada de uma forma detalhada a base teórica utilizada nesta dissertação de mestrado.

3.1 Sistemas em lote (*Batch*)

A ferramenta proposta surgiu com uma demanda inicial por uma ferramenta precisa e de baixo custo para a análise do planejamento de capacidade de alguns sistemas do setor de telecomunicação, mais precisamente, sistemas em lote (*batch*) (GRACON, NOLBY & SANSOM, 1971). Neste tipo de sistema, as variáveis geradas pelas requisições de entrada passam por um ciclo de processamento definido de acordo com o algoritmo da funcionalidade implementada, até gerarem variáveis de saída mantidas por um fluxo de controle sem interrupções externas do usuário durante a execução (WALDBAUM, 1973).

Normalmente os sistemas em lote utilizam o recurso de processamento paralelo através do uso de *threads* em sistemas operacionais multitarefa. As *threads* representam tarefas ou sub-processos que são executados independentemente umas das outras, mas dentro do contexto de um mesmo processo pai. Uma *thread* compartilha seção de códigos e dados com o processo pai, mas cada uma tem sua própria pilha e contexto, onde está incluído o contador de programa. Desta forma, uma *thread* requer menos recursos do sistema do que um processo e, além disso, a comunicação entre elas é bem simples e rápida.

Um sistema operacional multitarefa é um sistema capaz de executar diversos processos simultaneamente. Neste tipo de sistema cada processo possui uma fatia de tempo. A criação de um processo envolve a geração de um espaço de endereçamento e de uma imagem do aplicativo na memória, envolvendo seção de códigos, dados e pilha.

Neste caso, o que acontece é que esse processador é compartilhado entre as diversas tarefas que estão em execução. Como esse compartilhamento é muito rápido, a sensação que se tem é que todas as tarefas estão sendo executadas ao mesmo tempo. É o que chamamos de compartilhamento temporal do processador, ou *time-slice*.

O sistema *batch* alvo da pesquisa faz utilização intensiva dos processadores dos servidores envolvidos e a configuração do número de threads ideal é essencial para seu bom desempenho. Se o número de threads estiver abaixo do número de processadores, deixaremos de utilizar algum processador (figura 1), desperdiçando assim recursos computacionais, e se estiver muito alto podemos gerar um grande número de trocas de contexto nos processadores envolvidos no processamento da tarefa e, consequentemente, uma degradação de desempenho no ambiente (figura 2). A definição do número correto de *threads* otimizando o desempenho de determinado processo envolve varios testes no ambiente em que este irá ser executado.

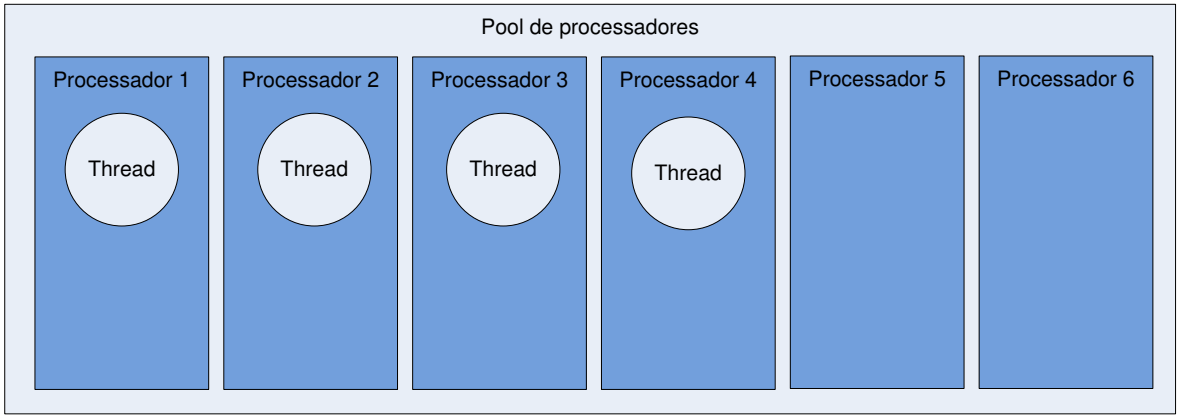

Figura 1 Número de threads menor do que o número de processadores

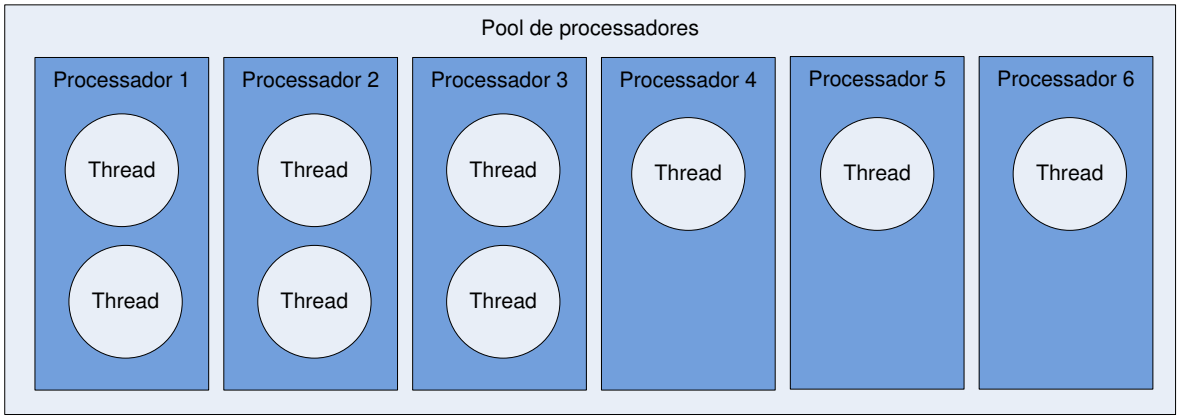

Figura 2: Número de threads maior do que o número de processadores

Apesar deste trabalho considerar somente o comportamento de sistemas *batch*, nada impede que em um trabalho futuro possamos fazer uma adaptação no modelo para torná-lo mais genérico, tratando outros tipos de sistemas, tais como, sistemas *online* e *web* (MENASCÉ & GOMAA, 2000).

3.2 Planejamento de Capacidade

O planejamento de capacidade (*Capacity Planning)* é o processo usado para prever quando os níveis de carga de trabalho saturarão o desempenho do sistema alvo e determinar o modo mais econômico de fazer com que esta saturação ocorra com a maior carga possível (MENASCÉ, 2000).

Algumas vezes o problema de desempenho do sistema não está na falta de um *hardware* mais rápido e sim em arquitetura, que não permite uma escalabilidade maior, mesmo adicionando novos recursos computacionais. Um exemplo de gargalo neste caso seria um único objeto Java sendo acessado por diversas threads/processos ao mesmo tempo. Portanto o planejamento de capacidade de sistemas é importante mesmo para empresas que possuam capital financeiro suficiente para a compra de um novo recurso computacional com maior desempenho (LO, 1980).

O planejamento de capacidade é muito importante para se evitar perdas financeiras, garantir a satisfação do cliente com um baixo tempo de resposta e preservar a imagem externa da empresa, principalmente quando o sistema for de uso público. Muitas vezes um problema de capacidade e desempenho não pode ser resolvido instantaneamente e exige um estudo detalhado de todas as camadas da aplicação envolvida. Por isso é recomentado que ele seja um processo contínuo dentro de uma fábrica de *software*, desde o planejamento do sistema até o momento em que este sistema já estiver sendo executado na empresa cliente (BAGCHI, *et al.*, 2006).

O processo de planejamento de capacidade é ilustrado pelo diagrama da figura 3. É necessário inicialmente entender o ambiente e seguir algumas etapas cíclicas para gerar modelos de custo, carga e desempenho do sistema. Com estes modelos é possível ter uma análise completa de custo/desempenho do sistema e gerar planos de configuração e investimento futuro de acordo com o desempenho esperado (CARPER, HARVEY & WETHERBE, 1983).

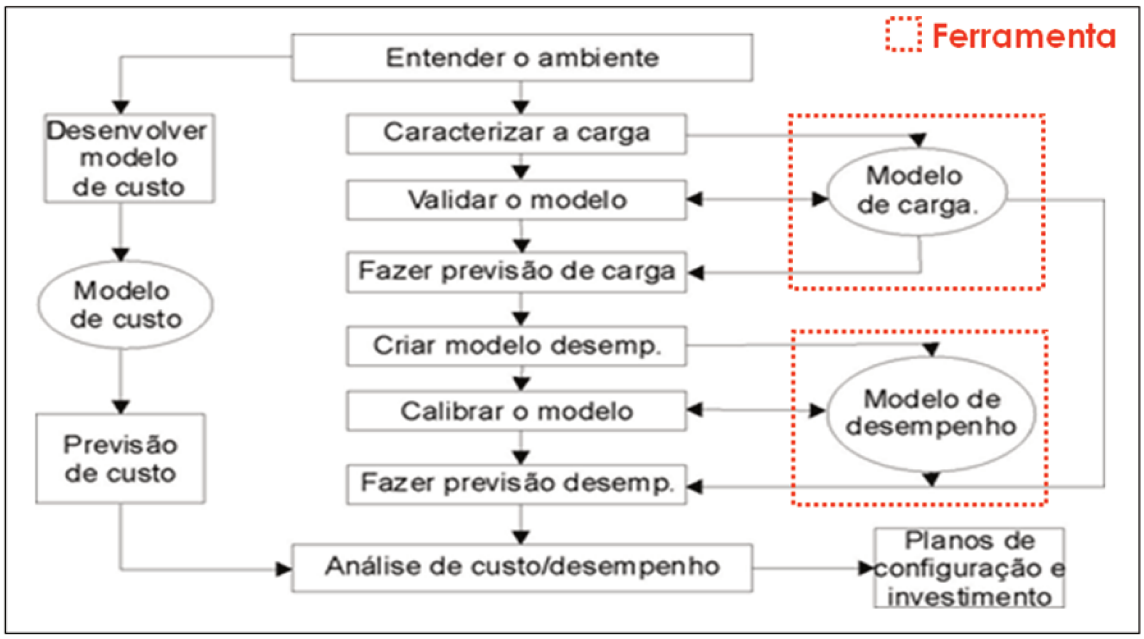

Figura 3: Representação do processo de Planejamento de Capacidade

Como visto na figura 3, existem basicamente 3 modelos que podem ser construídos através do trabalho de planejamento de capacidade, detalhados a seguir:

- Modelo de carga: define a utilização de recursos computacionais pelo sistema e considera a quantidade de recursos disponíveis de *hardware*. Normalmente são modelos apoiados em cálculos matemáticos que definem qual é a carga do sistema, de acordo com a unidade de processamento específica do sistema, em qualidade e quantidade (qual o tipo de carga e qual o impacto desta carga no ambiente respectivamente). Este modelo é muito importante para que o sistema real seja bem representado, pois caso contrário a utilização de recursos computacionais não será conhecida. Distribuições estatísticas são utilizadas para representar a carga que chega no sistema, através de variáveis aleatórias e processos estocásticos (KLEINROCK, 1975).
- Modelo de desempenho: avalia a situação atual do desempenho do sistema, incluindo tempo de resposta, vazão e utilização de recursos computacionais. Busca identificar perfis de desempenho da aplicação em determinado *hardware*, conforme a funcionalidade sendo executada e a carga definida no modelo de carga.
- Modelo de custo: é uma visão monetária do que é necessário para que o sistema seja executado com determinado desempenho. É importante para decidir se a compra de um

tipo de máquina compensa financeiramente, considerando seus custos versus retorno de desempenho para o sistema.

A ferramenta proposta neste documento tem foco no modelo de carga e no modelo de desempenho. O modelo de custo se baseia em uma pesquisa de custos atualizados que muda de acordo com o ritmo de inovações tecnológicas. Sua implementação se torna fácil e simples através do uso de planilhas, a partir do momento em que estejam estabelecidos os modelos de carga e desempenho.

3.3 Métricas de desempenho

Na figura 4, para um dado sistema computacional são obtidos alguns resultados possíveis quando é verificado um pedido (requisição) de serviço ao sistema. Desta forma são consideradas as respectivas métricas de desempenho para cada um dos resultados (SURUAGY, 2004):

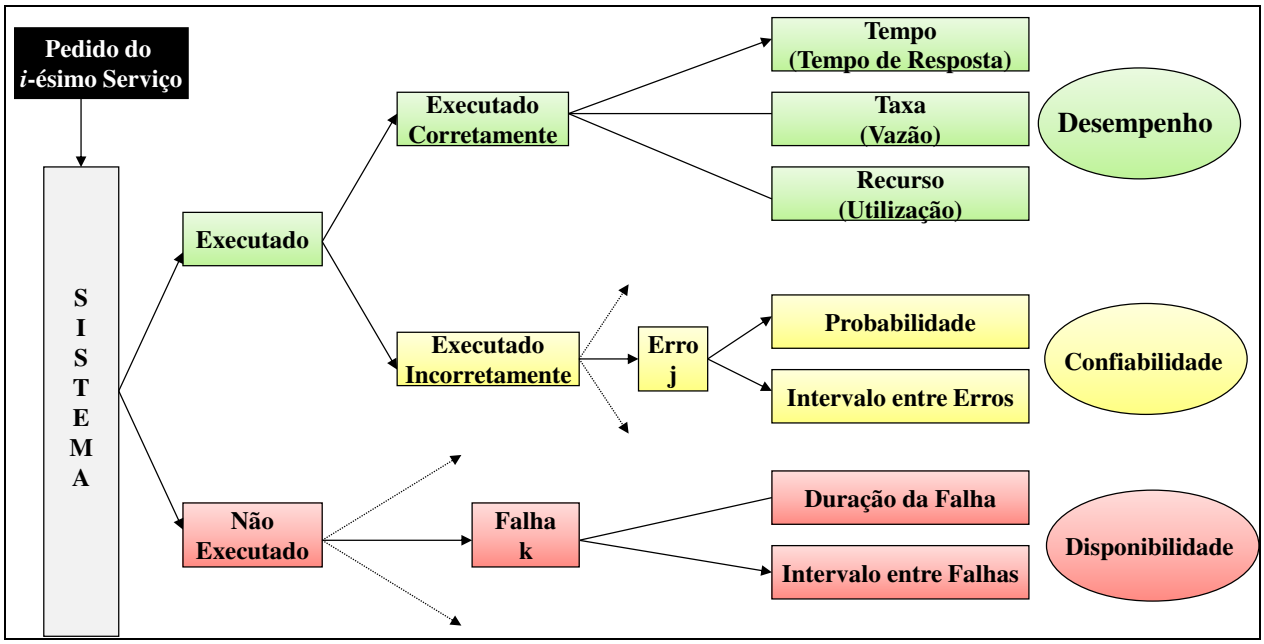

Figura 4: Resultados de execução e métricas de desempenho de um sistema

Os possíveis resultados para a execução de um pedido ao sistema são:

- Execução correta
- Execução Incorreta devido à ocorrência de um erro. Um erro ocorre quando o resultado esperado de determinada funcionalidade não é obtido devido à uma falha funcional do programa.
- Não execução devido à ocorrência de uma falha. Uma falha ocorre quando algo inesperado acontece no ambiente como um todo impedindo a execução de determinada funcionalidade ou sistema.

Na figura 4 consideramos as seguintes métricas de desempenho com seus respectivos parâmetros, conforme o resultado da execução do pedido ao sistema (SURUAGY, 2004):

- Desempenho:
    - Vazão (*Throughput*): é a métrica que define o número de requisições executadas por unidade de tempo. A unidade utilizada depende do tipo da requisição que está sendo enviada ao servidor. Por exemplo, para um sistema *batch*, uma unidade utilizada pode ser a quantidade de transações servidas por segundo.
    - Utilização: fração do tempo total de execução que um determinado recurso ou conjunto de recursos computacionais estão ocupados. É medido em um percentual que diz quanto do tempo total do recurso foi utilizado.
    - Tempo de resposta: tempo desde o início até o final da realização de um determinado serviço disponível na aplicação alvo. O tempo de resposta pode ser decomposto no tempo de serviço e tempo de fila. O tempo de serviço corresponde ao tempo em que cada um dos recursos (CPU, Discos e Rede) foram utilizados e o tempo de fila representa o tempo de espera pela utilização de cada um desses recursos.
- Confiabilidade
    - Probabilidade da ocorrência de erros e intervalo de tempo esperado entre estas ocorrências.
- Disponibilidade
    - Probabilidade da ocorrência da falhas e intervalo de tempo esperado entre estas ocorrências.

A ferramenta proposta baseia-se nas métricas de velocidade, foco principal de todo o trabalho de Planejamento de Capacidade. As métricas de confiabilidade e de disponibilidade não são foco desta dissertação e exigem um estudo diferente do que foi realizado aqui, com geração de novos modelos estatísticos específicos para isto.

Outro conceito fundamental no desempenho de sistemas são os gargalos de desempenho. Conforme o número de processos cliente no servidor aumenta, o desempenho para o usuário final tende a ser restrito pelo desempenho de alguns componentes do sistema e do *hardware*. Estes componentes que limitam o desempenho do sistema são chamados de gargalos. A identificação desse gargalo é muito importante pois deverá ser o primeiro componente a receber um *upgrade* para melhorarmos o desempenho do sistema. Sempre que eliminamos um gargalo, o desempenho do sistema melhora, porém surge um outro gargalo em outro ponto que gera outro limite de desempenho ao sistema. Gargalos sempre existem, mas o objetivo da análise de desempenho é melhorar o sistema o máximo possível para que tenha um desempenho aceitável para o usuário final.

3.4 Técnicas de Avaliação de desempenho

As técnicas de avaliação de desempenho de um sistema são diversas e têm o objetivo de traçar o real comportamento de um sistema. Para validação deste modelo, consideramos o resultado de mais de uma técnica, confrontando seus resultados. Nesta dissertação foi criado um modelo utilizando técnicas de simulação que foram confrontadas através de medições do sistema real implementado.

A figura 5 mostra as possíveis técnicas de avaliação de desempenho:

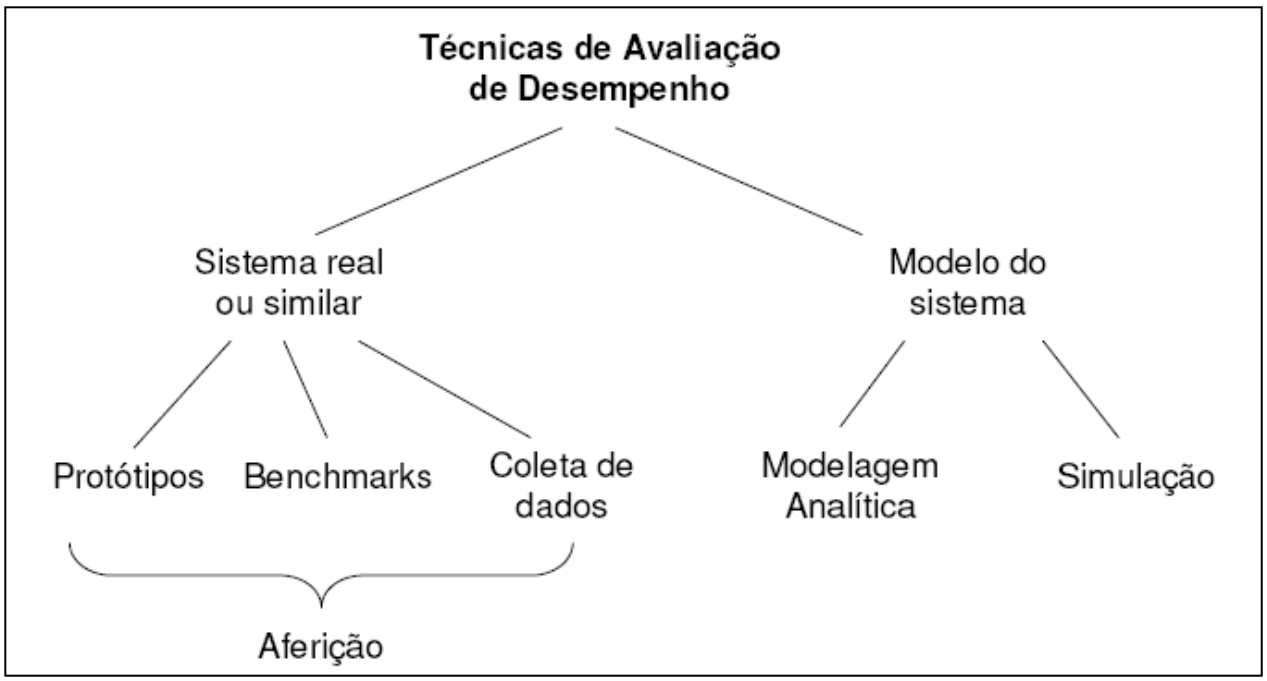

Figura 5: Técnicas de avaliação de desempenho

A seguir temos descritas as principais técnicas e suas respectivas características (ROSE, 1978):

- Medição direta do sistema real ou similar (aferição): consiste em comparar alternativas de *hardware* e configurações, confrontando resultados de desempenho do sistema em cada um deles. Esta técnica é inviável quando não temos o *software* pronto e todas as alternativas possíveis de *hardware* disponível para testes. Quando não existe o sistema real, baseamos-nos em resultados de um sistema similar, uma parte pequena do sistema (protótipo) e *benchmarks* (sistemas modelos), o que muitas vezes torna a análise um pouco imprecisa. Caso exista o sistema, utilizamos a coleta de dados como utilização de recursos e vazão.
- Modelagem Analítica: Uma das formas é a utilização da teoria de filas e faz a aproximação do sistema real através de um modelo matemático bem definido. A precisão do resultado utilizando esta técnica de forma simplificada é inferior ao da simulação, pois muitas vezes é necessário simplificar o modelo quando é impossível representá-lo totalmente devido sua complexidade de cálculo (BUZEN, 1978).
- Simulação: é baseada na simulação de sistemas a eventos discretos, onde o comportamento do sistema real é simulado através de ferramentas com esta finalidade. Desta forma é

possível tratar modelos mais complexos cujos cálculos seriam inviáveis analiticamente. Alguns autores consideram esse tipo de técnica ineficiente pois o desempenho da execução de uma simulação tende a ser custoso computacionamente. Com um trabalho de ajuste de desempenho de uma ferramenta de simulação e simplificação do modelo não deixando de lado a precisão, conseguimos um bom desempenho desta técnica e uma precisão totalmente aceitável. Na seção 2.6 detalhamos esta técnica e como foi utilizada.

3.5 Rede de filas

A teoria de filas provê modelos para demonstrar o comportamento de um sistema que ofereça serviços cuja demanda cresce aleatoriamente, tornando possível dimensioná-lo de forma a satisfazer os clientes e ser viável economicamente para o provedor do serviço, evitando desperdícios e gargalos (MENASCÉ & ALMEIDA; 2001).

A seguir, na figura 6, temos a representação básica de uma fila para um serviço, na qual existe a chegada de requisições, uma fila e um serviço a ser executado.

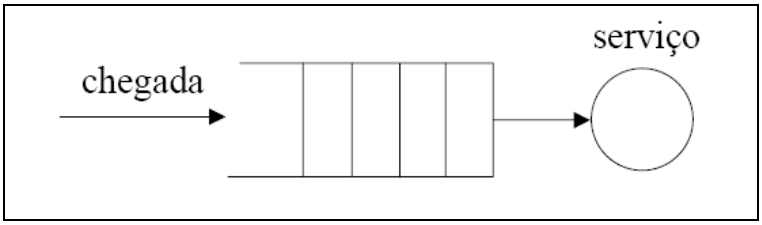

Figura 6: Fila para um serviço

Um parâmetro importante em uma fila é a função *S(n),* que caracteriza o tempo médio de serviço por uma requisição a determinado recurso, quando existem *n* clientes na fila. O número *n* é o tamanho atual da fila.

Existem 3 categorias de recursos em uma rede de fila e eles variam, dependendo da existência de enfileiramento ou não e se o tempo médio de serviço *S(n)*, depende do tamanho da fila *n* ou não (MENASCÉ & ALMEIDA; 2001):

- Recursos independentes de carga: o tempo médio de serviço não depende da carga, mas pode existir enfileiramento, ou seja *S(n)=S* para todos os valores de *n*.

- Recursos dependentes de carga: recursos onde existe enfileiramento e o tempo médio de serviço depende de carga, ou seja *S(n)* é uma função arbitrária de *n*.
- Recursos de atraso, indicam situações onde não existe enfileiramento, assim o tempo total gasto por um pedido em um recurso de atraso é o tempo de serviço do pedido. A função de tempo médio de serviço não depende do número de pedidos presentes no recurso, ou seja, *S(n)=S* para todos os valores de *n.*

Os modelos de filas se baseiam em processos aleatórios e dependem dos tipos de distribuição de probabilidade para as taxas de chegada e de serviço do sistema (DENNIN & BUZEN, 1978).
O modelo proposto nesta dissertação se baseia na Teoria de Filas. Para se especificar um modelo devemos obedecer os seguintes passos:

- Especificar modelos estocásticos que representam os processos de chegada e de serviço no sistema
- Especificar parâmetros que definem a capacidade e número dos servidores
- Especificar as políticas de operação do serviço, tais como priorização e negação de atendimento

Quando um sistema de filas é inicializado, ele passa por um período inicial de operação que geralmente não reflete seu comportamento típico. Este período inicial de operação (período transiente) não é representativo de como o sistema vai se comportar no restante do tempo, pois neste período inicial o consumo de recursos pode ser maior. Normalmente nos preocupamos com a operação de sistemas de filas cujo funcionamento está estabilizado. Neste momento dizemos que o sistema atingiu o equilíbrio (*steady state*) (DENNIN & BUZEN, 1978; BUZEN, 1978). A partir deste instante diz-se que o sistema entrou em regime permanente, de funcionamento estável e previsível. A solução e o estudo de sistemas de filas é muito mais simples quando sabemos que o sistema está em equilíbrio. Neste contexto, a maioria dos parâmetros importantes se mantêm fixa, o que torna a análise do sistema bem mais simples.
O conceito de utilização representa a relação entre o tempo em que determinado recurso foi utilizado e o tempo total de execução do sistema. De acordo com a fórmula a seguir, U_i é a utilização de um recurso i, B_i é o tempo que o recurso ficou ocupado e T é o tempo total do período de observação do sistema.

$$U_i = \frac{B_i}{T}$$

Existem 2 tipos de modelos baseados nesta teoria:

- Modelo de rede aberta: neste modelo, os clientes passam através do sistema uma única vez. Por exemplo, um dispositivo de rede processando um fluxo de pacotes pode ser modelado com um sistema aberto. O número de clientes do sistema é calculado pela taxa de chegada de clientes e as características do serviço, tais como tamanho máximo da fila e tempo de serviço. Se existirem tipos diferentes de clientes, um modelo aberto de múltiplas classes de serviços poderá ser utilizado.
- Modelo de rede fechada: neste caso, o número de clientes é fixo. O exemplo clássico é o sistema computacional de tempo compartilhado com um número fixo de terminais de usuário. Neste modelo, o número de clientes é conhecido e a vazão é calculada. Um método interativo conhecido como Análise do Valor Médio (*Mean Value Analysis* ou MVA) é usado para calcular vazão e latência. A exemplo dos modelos abertos, os modelos fechados podem ser de classe única ou de múltiplas classes.

A escolha do modelo utilizado na análise das filas depende do sistema que estamos considerando. No sistema onde temos o número de clientes conhecido e constante utilizamos o modelo de rede fechada, caso contrário, o modelo de rede aberta.

3.6 Simulação

A simulação é uma técnica muito útil para a avaliação de desempenho de sistemas computacionais, especialmente se o sistema não estiver ainda disponível para testes e medições e serve para prever, no nosso caso, o desempenho de diferentes máquinas, oferecendo facilidade ao efetuar comparações variando a carga esperada no ambiente computacional, baseando-se em um modelo de carga (SCHRIBER & BRUNNER, 2006).

Simulações a eventos discretos se ocupam de sistemas caracterizados pelo fato de que as mudanças de estado somente ocorrem quando ocorre um evento e o tempo é discretizado. Assim simulações de sistemas de computadores escritas com o propósito de avaliação de desempenho são, em geral, simulações a eventos discretos (INGALLS, 2001).

Um programa de simulação pode ser organizado de diversas maneiras sendo as mais usuais:

- orientação a eventos: onde o programador deve estabelecer como o estado se altera com a ocorrência de eventos.
- orientação a processos: o programador deve identificar no sistema entidades que são submetida a processos. Neste caso ele deve também descrever as interações entre os processos.
- Orientação ao exame de atividades: neste caso, o programador deve identificar no sistema as atividades e as condições para seu início e término.

Uma simulação orientada a eventos modela as funcionalidades do sistema como uma série de eventos que ocorrem assincronamente e em intervalos irregulares (BANK, 2000). Por exemplo, a simulação de um servidor de arquivos de rede deve incluir eventos como a chegada de um pacote ethernet ou o término da escrita em disco. Este tipo de simulação modela uma grande variedade de sistemas e é tipicamente utilizado para modelar núcleos de processador ou outra lógica digital com uma única frequência de relógio. Note, porém, que as as abordagens podem ser combinadas em uma simulação que inclua eventos síncronos e assíncronos.

Existem três técnicas básicas para gerar cargas de trabalho para realizar simulações: estocástica, rastreamento (*trace-driven*) e baseada em execução. A seguir descrevemos cada uma delas (MENASCÉ & ALMEIDA, 2001):

- a simulação estocástica descreve a chegada de padrões de clientes e outros aspectos da carga por amostragem a partir de uma distribuição probabilística. Muitas cargas podem ser descritas precisamente pelo uso de uma distribuição apropriada. As cargas estocásticas são uma boa escolha quando a informação detalhada sobre a carga não está disponível ou quando existe necessidade de variar as características da carga. A geração de carga é eficiente e não necessita de grandes arquivos de dados.
- a simulação por rastreamento representa a carga como uma sequência de operações ou requisições através de dados reais. Por exemplo, para a simulação de um servidor Web, uma sequência de requisições HTTP deve ser rastreada, enquanto na simulação de uma CPU, as operações do micro-código devem ser rastreadas. Se dados de rastreamento que representam corretamente a carga estão disponíveis, obteremos bons resultados de simulação, sendo desnecessário escrever o código que modela a carga. A desvantagem

desta técnica é que montar uma coleção de rastreamentos é um esforço não trivial e o tamanho dos arquivos de dados é muito grande.

- a simulação baseada em execução é utilizada para modelar processadores em detalhes. O desempenho de um processador é afetado diretamente por outras características da arquitetura, como *caching, pipelining*, unidades funcionais e tecnologia de compilador (HENNESSY & PATTERSON, 1996). A entrada de uma simulação baseada em execução é o mesmo código executável que seria processado no sistema real.

3.6.1 Componentes de um sistema de simulação

Os sistemas de simulação possuem uma infraestrutura interna capaz de gerenciar cada fase da simulação e gerar resultados finais contendo estatísticas de execução (LAW & KELTON, 1991). Um sistema de simulação orientado a eventos deve possuir basicamente os seguintes componentes (CASSANDRAS & LAFORTUNE, 1999):

- Estado: consiste em uma lista onde as variáveis do estado atual do sistema são armazenadas. Estas variáveis mostram a atual situação do sistema. Na ferramenta implementada, o estado é representado por uma classe desenvolvida em linguagem Java (NIÑO & HOSCH, 2005) que armazena o estado de cada CPU (tempo total de execução e threads ativas), o estado de cada disco (requisições e utilização), memória física e *swap* utilizada, theads ativas (transações relacionadas e tempo de execução) e transações que entraram e saíram do sistema.
- Tempo: uma variável onde o tempo de simulação é armazenado e é responsável por definir o início e final da simulação.
- Calendário de Eventos: uma lista onde todos os eventos escalonados são armazenados juntamente com seu instante de ocorrência. Na ferramenta, o calendário de eventos é representado por uma classe Java cujo método principal atualiza os eventos que podem ocorrer a cada momento da simulação. Os possíveis eventos são os eventos de chegada e partida de processos, de chegada e partida de processos às CPUs e de chegada e partida nos discos.

- Registros de Dados: variáveis e/ou listas de armazenamento de dados a serem utilizados em estimativas. Eles correspondem a dados da utilização de cada recurso e tempo de espera de todas as filas do sistema.
- Rotina de Inicialização: responsável pela inicialização todas as estruturas de dados descritas no modelo no início da simulação.
- Rotina de Atualização do Tempo: identifica o próximo evento que irá ocorrer e avança o tempo atual da simulação para o tempo de ocorrência deste evento.
- Rotina de Atualização do Estado: atualiza o estado através de sorteio com base na definição do próximo evento a ocorrer.
- Rotinas de Geração de Variáveis Aleatórias: produzem, a partir de números aleatórios gerados pelo simulador, amostras das variáveis aleatórias associadas aos tempos de vida dos eventos, de acordo com as distribuições estatísticas pré-definidas.
- Rotina de Geração de Relatórios: calcula as estimativas das grandezas de interesse a partir dos dados coletados na execução de uma simulação. Esta tarefa usa dados dos registros de dados para gerar o resultado final da simulação. Na ferramenta gerada nesta dissertação o relatório é representado através de tabelas e gráficos, facilitando a compreensão dos resultados obtidos.
- Programa Principal: responsável pela coordenação geral de todos os componentes da simulação. Chama inicialmente a rotina de inicialização e durante a execução chama repetidamente as rotinas de atualização e altera o calendário de eventos. Ele também é responsável por terminar o programa quando solicitado.

Os componentes de simulação são os responsáveis pela execução de várias etapas da simulação, sendo que cada um tem um papel importante para o seu funcionamento. As etapas envolvem vários ciclos, que são executados até que a condição de parada seja satisfeita (toda entrada do sistema é processada ou o tempo máximo definido é atingido). Cada ciclo de simulação é composto das seguintes operações:

1. Remover a primeira linha do calendário;
2. Atualizar o tempo com o valor inicial $t_{1;}$
3. Atualizar o estado de acordo com a probabilidade de ocorrer o evento $p(x';x,e_1)$;
4. Deletar do calendário as linhas correspondentes a eventos infactíveis no novo estado;

5. Acrescentar ao calendário os eventos factíveis que ainda não estejam escalonados. O tempo associado ao novo evento é obtido adicionando-se ao tempo corrente a um valor sorteado da estrutura de relógio;
6. Reordenar o calendário por ordem crescente do valor do tempo em que os eventos foram escalonados.

A figura 7, mostra a sequencia de operações realizadas nos componentes de simulação:

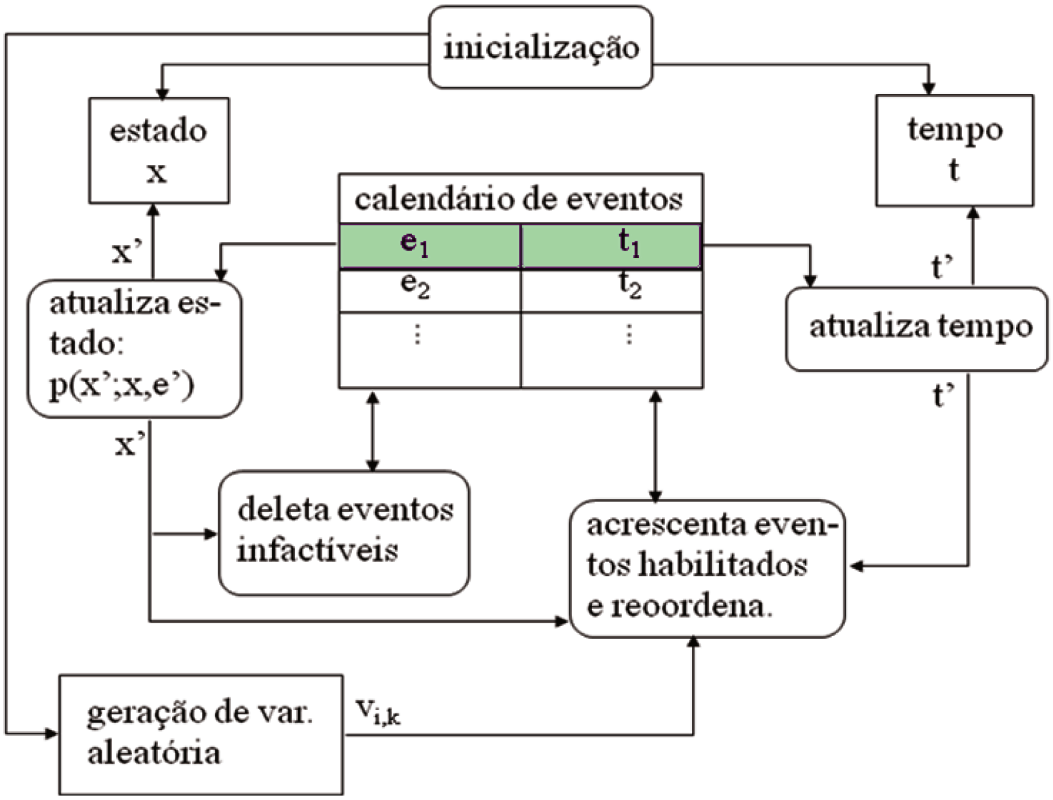

Figura 7 Sequência de operações realizadas nos componentes de simulação ((CASSANDRAS & LAFORTUNE, 1999)

3.7 Técnicas de validação

O processo de validação de resultados faz-se necessário para termos certeza que o modelo proposto representa bem uma situação de um processamento real do sistema. Existem três regras básicas a serem seguidas durante o processo de validação proposta por Banks (CASSANDRAS & LAFORTUNE, 1999), sendo elas:

- Não acredite nos resultados da simulação antes de serem validados por modelagem analítica ou medições.

- Não acredite nos resultados da modelagem analítica antes de serem validados por simulação ou medições.
- Não acredite nos resultados das medições antes de serem validados por simulação ou modelagem analítica.

Este método consiste em comparar resultados de pelo menos duas das técnicas de avaliação de desempenho. No trabalho realizado, os resultados obtidos através de simulação foram validados com os de medições diretas dos sistemas envolvidos.

3.8 Ferramenta *Extend*

A ferramenta Extend é um pacote de simulação da empresa Imagine That Inc (**ExtendSim**), que realiza a simulação de sistemas contínuos e sistemas discretos, disponibilizando ao usuário a facilidade de uma programação visual (icônica) do modelo. Além disso é um programa efetivo e robusto com uma animação simples, mas muito útil para acompanhamento da execução de todo o processo de simulação. Esta ferramenta emite relatórios detalhados em vários formatos com os resultados da simulação e utilização de cada recurso.

A ferramenta Extend é muito eficaz, apropriada para modelagem ou reengenharia de vários tipos de processos, mesmo os não computacionais (ALEXANDER, 2006). Trata-se de um pacote muito rico, completado com um tutorial e manuais online, várias bibliotecas (componentes do modelo) e capacidade de salvar e imprimir resultados. Ele inclui diversos modelos pré-construídos, tanto do sistema de manufatura como do sistema de serviços. Esta ferramenta pode ser pensada em certo aspecto como um pograma orientado a objetos. Ela usa blocos, representados por ícones, que contém dados pré-definidos, que podem ser redefinidos pelo usuário. Cada bloco pode ser acessado no nível básico de usuário, onde os dados são colocados e os parâmetros ajustados ou no nível estrutural, onde podem ser feitas mudanças no código fonte para alterar significativamente o comportamento do bloco.

Os blocos dentro do Extend são classificados em bibliotecas de acordo com a sua funcionalidade. As bibliotecas básicas incluem Eventos Discretos, Genéricos (contínuos) e *Plotter* (para *reports* e gravação de dados).

Para a validação do modelo, antes mesmo do desenvolvimento da ferramenta final de planejamento de capacidade, utilizamos esta ferramenta para realizarmos uma simulação mais

rápida e efetiva do modelo não nos preocupando a respeito de possíveis erros funcionais. Somente quando o modelo estava validado, comparando resultados reais e simulados, iniciamos o desenvolvimento da ferramenta final. Esse processo facilitou também a validação de possíveis *bugs* na ferramenta desenvolvida. No final, a ferramenta implementada apresentou os mesmos resultados que a simulação utilizando o Extend.

Na figura 8 podemos observar a tela principal de montagem de modelo nesta ferramenta de simulação:

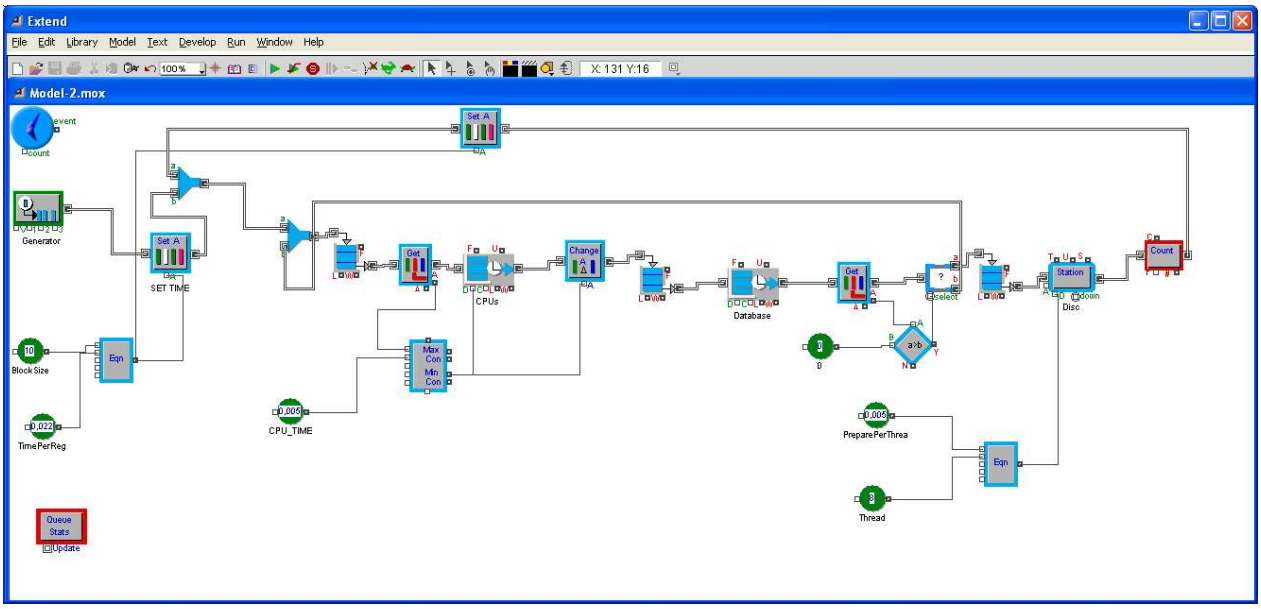

Figura 8: Tela de montagem de modelo na ferramenta Extend

3.9 Tecnologia HT (*HyperThreading*)

Durante a análise, possuíamos disponíveis para testes de validação vários servidores com a tecnologia HT, portanto foi necessário analisar o comportamento específico deste tipo de tecnologia de processadores. A tecnologia HT permite que um processador atue como se existissem 2 processadores físicos para o sistema operacional.

O desempenho de um processador HT não equivale a exatamente 2 processadores comuns, pois os processadores HT possuem somente o núcleo duplo, compostos de registradores e contadores de interrupções duplicados, porém compartilham os demais recursos de execução.

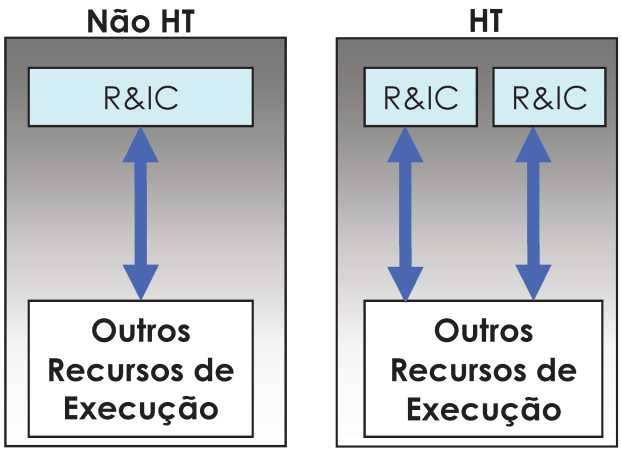

Figura 9: Comparação de processadores com e sem a tecnologia HT

A seguinte fórmula, obtida experimentalmente, considera a degradação de 20% do desempenho e consequentemente aumento no tempo de reposta da CPU para cada processo:

$$CPUTime_{HT} = CPUTime + (0.2 * CPUTime) * (jobs - 1)$$

O tempo total de CPU gasto, utilizando a tecnologia HT ($CPUTime_{HT}$) corresponde ao tempo gasto por um processador comum (*CPUTime*) adicionando um fator de degradação conforme o número de *jobs* (processos) que o processador está executando simultaneamente.

A tecnologia de um processador HT pode ser ativada ou desativada conforme necessidade. Em uma das máquinas analisadas mais a frente no nosso trabalho esta tecnologia estava desativada devido à alguns *bugs* detectados no momento da instalação do sistema operacional na máquina pelo pessoal responsável pelo suporte. Neste caso, identificamos comportamento similar à CPUs comuns que não possuem a tecnologia HT.

Capítulo 4
Ferramenta Proposta

Neste capítulo, apresentaremos a proposta de desenvolvimento da ferramenta de simulação voltada ao processo de planejamento de capacidade. A proposta como um todo é representada pelas seguintes fases em sequência cronológica:

1. Definição do modelo conceitual de desempenho: nesta fase o objetivo é gerar um modelo que represente fielmente um sistema computacional considerando todos os pontos de estudo que realmente afetam o desempenho (JAIN, 1991) de um sistema e que podem ser considerados no processo de simulação. Este modelo é tratado detalhadamente na seção 4.1.
2. Desenvolvimento de um *software* para avaliação de desempenho de um ambiente computacional, baseado em *benchmarking*, gerando diversos tipos de carga e medindo a utilização dos recursos de execução nos servidores. Este software foi implementado e utilizado na validação do modelo de simulação proposto, considerando várias cargas distintas no *hardware* testado. A descrição de como este *software* foi implementado pode ser vista na seção 4.2.
3. Validação e calibração do modelo proposto através de um processo de *benchmarking*. Usando o modelo proposto foram simulados programas padronizados (*benchmarks*) e os desempenhos obtidos por simulação foram medidos e comparados com os resultados obtidos através da execução em hardware, utilizando-se nesta fase, para simular o modelo, a ferramenta de simulação Extend. O uso de uma ferramenta de simulação já existente no mercado nesta fase foi necessária para agilizar o trabalho de validação do modelo e para evitar problemas de implementação da ferramenta final. A seção 4.3 descreve detalhadamente esta atividade.
4. Aplicação do modelo validado em um sistema de lotes que trata do processo de tarifação de uma empresa brasileira de grande porte do setor de telecomunicações. Esta fase busca provar que através da generalização do modelo utilizado, é possível representar qualquer sistema em lotes. Detalhes desta aplicação podem ser vistos no capítulo 4.

5. Implementação de uma ferramenta de simulação desenvolvida em linguagem Java baseada no modelo validado, simplificando a modelagem e execução da simulação. A ferramenta tem o objetivo de abstrair todo o conhecimento necessário para a montagem do modelo, facilitando e tornando possível a execução do planejamento de capacidade para profissionais que não possuem conhecimento sedimentado na área de modelagem e simulação. Este item é tratado no capítulo 5.
6. Comparação de resultados obtidos na ferramenta Extend com os da ferramenta implementada em Java. Esta fase é relativamente simples pois o algoritmo utilizado pela ferramenta Extend é o mesmo utilizado na ferramenta proposta, portanto os resultados devem ser idênticos. O Capítulo 6 mostra esse processo de comparação de resultados.

4.1 Modelo Conceitual

O modelo de simulação (MENASCÉ, ALMEIDA & DOWDY, 2004) proposto nesta dissertação teve como foco obter uma precisão alta com o mínimo esforço de processamento possível, garantindo assim um desempenho maior em sua execução. Para o problema de planejamento de capacidade, existe uma grande variedade de alternativas de modelagem e para aumentar a precisão é necessário aumentar o número de elementos considerados do sistema. Com isso os requisitos de coleta de dados do sistema real também aumenta. É preciso estabelecer um equilíbrio razoável entre a precisão do modelo e a facilidade e desempenho de uso, para que seja permitida a análise de várias alternativas com pouco esforço e baixo tempo de resposta.

No modelo proposto a seguir, os processos (*jobs*) passam várias vezes pelo sistema computacional concorrendo por recursos de memória, CPU e I/O (discos) até que o processamento seja totalmente executado. Alguns processos podem ou não utilizar determinados recursos em seu ciclo de execução, como é o caso de uso de discos.

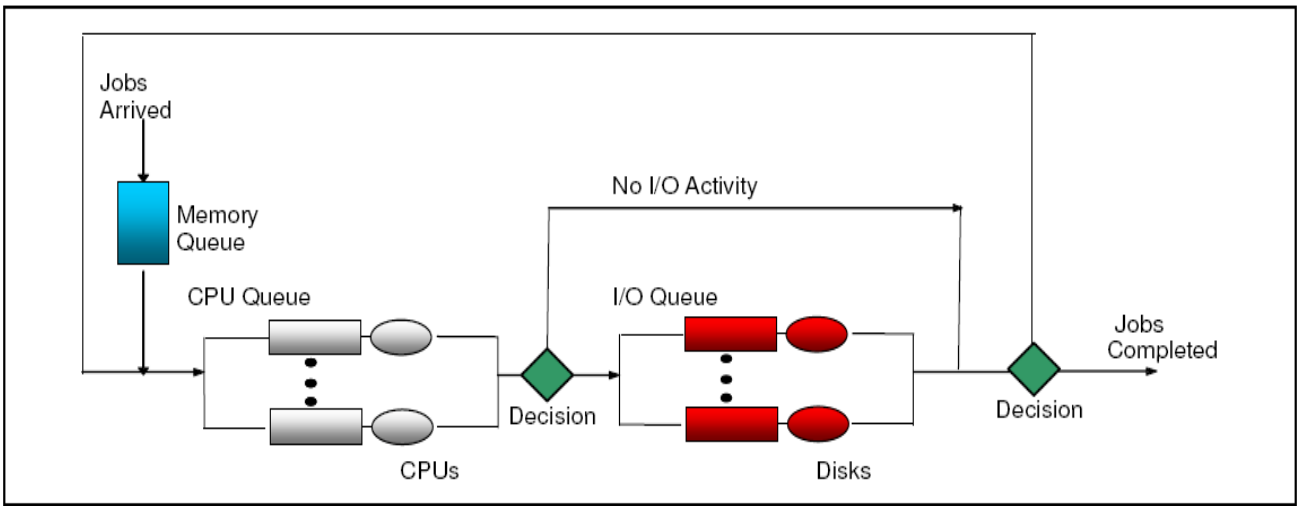

Figura 10: Modelo de filas proposto

No modelo proposto na figura 10, onde cada recurso tem sua própria fila de espera e, no geral, são entidades que representam os componentes de *hardware*. Desta forma, o modelo representa, como um todo, o servidor onde um sistema específico é executado.

Como entrada para este modelo, temos os *jobs* (transações) (DENNING, 1991) do sistema em lotes que definem todos os fatores de carga de trabalho do sistema a ser analisado. Os *jobs* chegam ao sistema e usam um espaço particular de memória e mesmo que essa memória possa ser acessada um grande número de vezes, esse acesso é tão rápido que pode ser considerado somente uma única vez na simulação e em muitos casos até desconsiderado desse processo (LO, 1980).

Depois do acesso à memória, a transação passa por alguns ciclos de CPU e acessos aos discos disponíveis no *hardware*, onde cada ciclo é executado conforme o *time slice* da CPU, que é dependente do sistema operacional instalado no servidor. Nos nossos testes consideramos o sistema operacional Unix, cujo tempo de fatia de CPU médio é de aproximadamente 100 ms, mas o modelo pode ser adaptado à qualquer Sistema Operacional, bastando alterar o parâmetro de *time slice* antes da execução na ferramenta de simulação. Este parâmetro não é muito importante nesse processo, porque não causa muita variação no tempo de execução retornado pela simulação.

Quando uma transação tem seu processamento finalizado, ela sai do sistema e é considerada como processada corretamente.

Podemos ter muitos processos neste sistema sendo executados no mesmo momento através do uso de *threads*. As *threads* sempre competem pelos componentes de hardware disponíveis e geram o paralelismo de execução.

Dois tipos de arquiteturas de servidores foram estudadas, a com processadores simples x86 e a com múltiplos processadores com tecnologia *HyperThreading*. A maior parte dos sistemas possui algum sincronismo entre as *threads*, isolando determinada execução a somente um processo (não há concorrência). Esse mecanismo é proporcionado por código de programação através de um algoritmo de semáforo implementado por quase todas as linguagens de programação para garantir que somente uma *thread* acessa algum recurso por vez. O desafio para se ter um bom desempenho em um sistema computacional é manter a taxa de sincronismo entre as *threads* a mais baixa possível, mas sempre devemos considerá-la durante a simulação, porque muitas vezes é impossível ela não existir, devido à lógicas necessárias de controle de processos no algoritmo da própria aplicação. A inovação deste modelo é a consideração do impacto do sincronismo, sendo representado por um fator de 0 a 1, onde 0 representa um sistema altamente escalável e 1 um sistema totalmente *monothread* (somente uma thread e um processador é utilizado em todo o ciclo de processamento). Se o sincronismo é muito alto, não temos efeitos de escalabilidade no desempenho do sistema ao aumentamos o número de threads de processamento e não será possível otimizar o sistema somente através de parametrizações para um alto desempenho. O sincronismo tem diferentes impactos dependendo da arquitetura de cada servidor envolvido (ROSE, *et. al.,* 1996). Por exemplo, um processador comum x86 é bem simples de se representar, pois pode somente manipular um processo por vez. Se um processo chega na CPU e a CPU está sendo utilizada por outro processo ele tem que esperar na fila de execução até o processo liberar a CPU ou o *time slice* do sistema operacional expirar. No caso dos processadores HT é possível manter mais do que um contexto de processo em um só processador, otimizando assim, o chaveamento entre os processos e consequentemente o desempenho.

4.2 Benchmarking

Existem duas definições importantes a serem consideradas neste tópico:

- *Benchmarking:* é o processo de comparação entre dois ou mais sistemas através de medições de resultados de desempenho, usando programas padronizados, onde os resultados são conhecidos.
- *Benchmarks*: são as cargas de trabalho utilizadas nas medições realizadas no processo de *benchmarking*.

Na avaliação de desempenho de sistemas, o *benchmarking* refere-se à execução de um conjunto de programas representativos com características bem definidas em diferentes computadores e redes, medindo os resultados obtidos em cada um deles. Ele é o principal método para medir o desempenho de uma máquina física real (KRISHNASWAMY & SCHERSON, 2000). Estes resultados são utilizados para avaliar o desempenho de determinado sistema ao executar uma carga de trabalho bem definida em cada um dos recursos (LAM & CHAN, 1987). Grande parte da popularidade dos *benchmarks* padrão disponíveis no mercado vêm do fato de que eles possuem objetivos de desempenho e cargas de trabalho bem definidos, que podem ser medidos e repetidos obtendo os mesmos resultados quando são executados sobre a mesma arquitetura de *hardware*.

Para ser útil, um *benchmark* precisa ser (GRAY, 1993):

- Relevante: precisa oferecer medidas de desempenho significativas dentro de um domínio de problema específico.
- Inteligível: os resultados do *benchmark* devem ser simples e fáceis de entender.
- Escalável: os testes de *benchmark* precisam ser aplicáveis a uma grande variedade de sistemas, em termos de custo, desempenho e configuração.
- Aceitável: os *benchmarks* precisam apresentar resultados imparciais, reconhecidos por usuários e fornecedores.

O processo de *benchmarking* foi muito importante para validar com precisão o modelo proposto, gerando cargas específicas em cada recurso computacional, medindo os resultados reais, comparando-os com os resultados simulados.

No trabalho descrito nesta dissertação, além da ferramenta de simulação, foi desenvolvida também uma ferramenta de *benchmarking* específica capaz de testar cada um dos recursos computacionais separadamente para uma melhor validação do modelo proposto. A ferramenta foi desenvolvida em linguagem Java e é capaz de representar diversos tipos de cargas padronizadas no ambiente de teste. Foram considerados 5 tipos de cargas baseados na combinação de 3 algoritmos, sendo eles:

- CPU: Geração de números primos utilizando paralelismo através de *threads* acessando vários objetos escritos na linguagem Java, evitando qualquer tipo de sincronismo no uso de CPU.
- IO: Escrita e leitura de arquivos em massa e em paralelo.
- Sincronismo: Geração de números aleatórios através de classe estática java.lang.Math (uma única classe disponível para várias *threads* gerando números paralelamente). Neste caso o sincronismo gerado é bastante alto conforme aumentamos o número de *threads* de execução.

Cada um dos algoritmos definidos anteriormente pode ser executado em paralelo com outro algoritmo.

No caso do tipo de carga possuir a execução de mais de um algoritmo simultâneo, o número de threads responsáveis pela execução dos algoritmos é dividida igualmente.

Como saída, o programa gera uma planilha com resultados de tempo de resposta para cada número previamente definido de *threads* de processamento e coleta dados de desempenho e utilização de CPU, discos, interrupções, trocas de contexto, processos em execução/bloqueados no sistema operacional.

Com isso foi possível gerar cargas específicas separadamente para cada recurso no início da análise, observando o resultado do tempo de resposta no sistema real e ajustando os parâmetros de simulação para que o modelo representasse a situação real final com precisão. Os tipos de carga compostos de vários algoritmos foram úteis para analisar a concorrência de uso de recursos distintos simultaneamente. Na próxima seção iremos mostrar como esses tipos de carga foram utilizados na validação do modelo.

4.3 Validação, calibração e análise

A validação, calibração e análise do modelo de simulação descrito anteriormente baseou-se primeiramente na execução do sistema de *benchmarking* em diversos servidores. Foram realizadas várias comparações de resultados do modelo com o sistema real para cada tipo de carga definido anteriormente. A calibração do modelo consistiu em definir os parâmetros de entrada tais como taxa de sincronismo, número e tipo de processadores, tipo de carga e carga total do sistema.

4.3.1 Cenários para o benchmarking

O processo de *benchmarking* foi executado em 4 servidores distintos aqui nomeados como S1, S2, S3 e S4, com os seguintes perfis de *software* e *hardware*:

- S1: Sistema operacional Linux c/4 processadores XEON 2.8 GHz cada (HT ativado)
- S2: Sistema operacional Linux c/2 processadores XEON 2.4 GHz cada (HT ativado)
- S3: Sistema operacional Linux c/4 processadores XEON 3.66 GHz cada (HT desativado)
- S4: Sistema operacional Solaris c/8 processadores 400 MHz cada

O objetivo da análise dos resultados obtidos é definir o modelo ideal para três arquiteturas distintas de máquinas:

- CPU c/tecnologia HT: S1 e S2
- CPU c/tecnologia HT desativada: S3
- CPU Comum: S4

4.3.2 Validação e resultados obtidos

Os tempos de processamento obtidos pelo sistema de benchmarking e por simulação são comparados a seguir. Para cada cenário, e para cada número de *threads*, foram calculadas médias dos tempo de execução a partir de um número suficientemente grande de execuções. Em seguida foram calculadas as diferenças percentuais entre tempos de execução e calculadas suas médias na medida em que se variava o número de *threads*. Desse modo foi calculado um erro percentual

médio para cada cenário, sendo também calculado o intervalo de confiança a partir de distribuição T-Student. A seguir os diversos cenários são descritos, sendo ao final apresentada uma tabela que sintetiza os resultados obtidos.

4.3.2.1 Servidor S1

Neste servidor, os resultados obtidos através da simulação para os tipos de carga CPU e CPU_SYNC foram muito próximos dos medidos experimentalmente no sistema real, como pode ser visto na figura 11. Não foi possível realizar testes de I/O nesta máquina devido a não disponibilidade de janelas de processamento no ambiente de teste, pois o servidor era muito utilizado para testes de desempenho de aplicações pela fábrica de software.

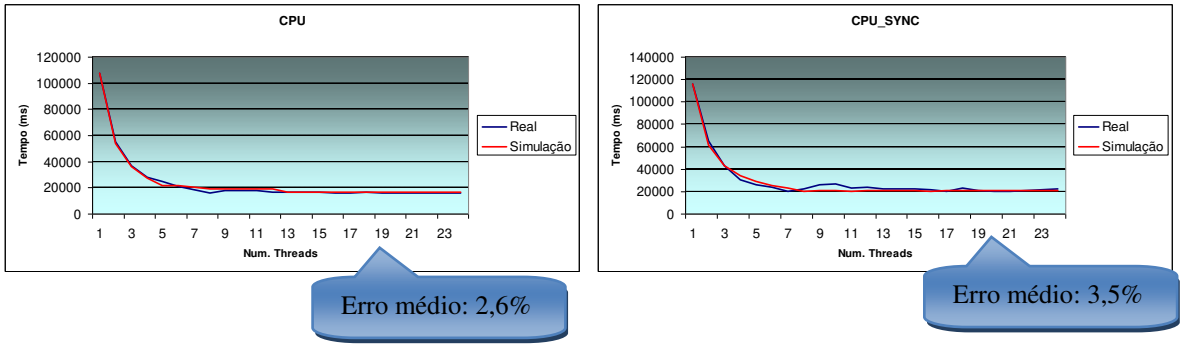

Figura 11: Resultados da simulação x sistema real do servidor S1

4.3.2.2 Servidor S2

No servidor S2, os resultados obtidos através de simulação para os tipos de carga CPU e CPU_SYNC foram muito próximos dos obtidos através de medições no sistema real, como pode ser visto na figura 12:

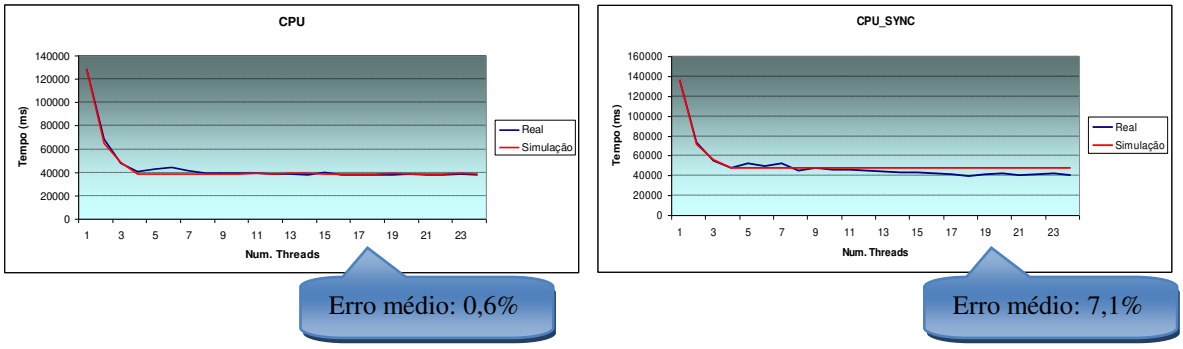

Figura 12: Resultados da simulação x sistema real do servidor S2

Foi encontrado um problema com os discos deste servidor. Os tempos de resposta variaram muito para os testes que envolviam o algoritmo de IO durante os testes realizados, sendo que não havia nenhum processo concorrendo com recursos da máquina, conforme figura 13. De acordo com o que a equipe responsável pelo sistema operacional da máquina, haviam tempos de leitura e escrita muito altos em alguns discos da máquina e consequentemente enfileiramento de requisições de I/O. Foi necessário abrir um chamado no fabricante e não foi possível a execução de novos testes até então.

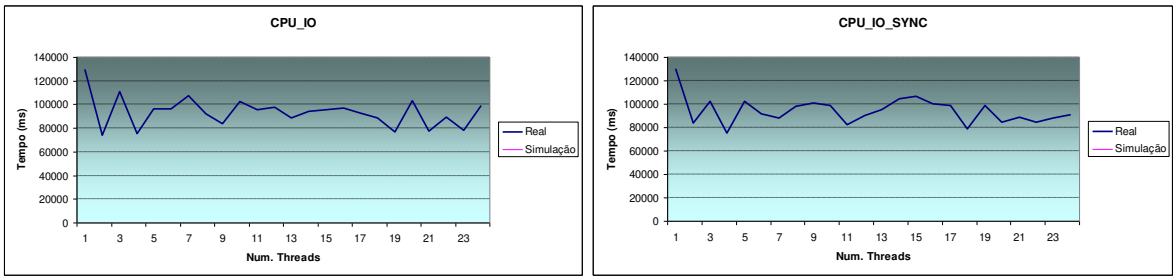

Figura 13: Resultados do sistema real do servidor S2 considerando uso de CPU+ I/O

Mesmo assim, com a simulação exclusiva de carga de I/0 no servidor, em média tivémos um erro médio muito baixo (3,8%) comparando resultados simulados e da aplicação real, provando que mesmo com problemas de I/O não considerados no modelo, podemos ter um resultado muito próximo da realidade.

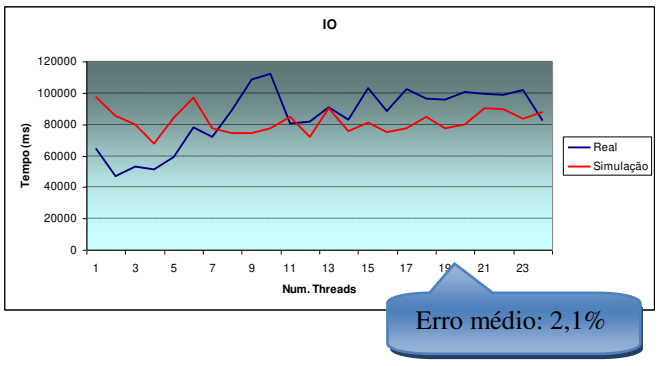

Figura 14: Resultados da simulação x sistema real do servidor S2 considerando somente a carga de I/O no ambiente

4.3.2.3 Servidor S3

No servidor S3, os resultados obtidos através de simulação para os tipos de carga CPU, CPU_SYNC, CPU_IO e CPU_IO_SYNC foram muito próximos dos obtidos através de medições

no sistema real, conforme as figuras 15 e 16, apresentando uma pequena margem maior de erro para casos de CPU_IO intensivo devido à variação maior do tempo de resposta dos discos, porém estes erros são perfeitamente aceitáveis em um planejamento de capacidade, onde se espera erros de 10 a 30% (MENASCÉ, ALMEIDA & DOWDY, 1994).

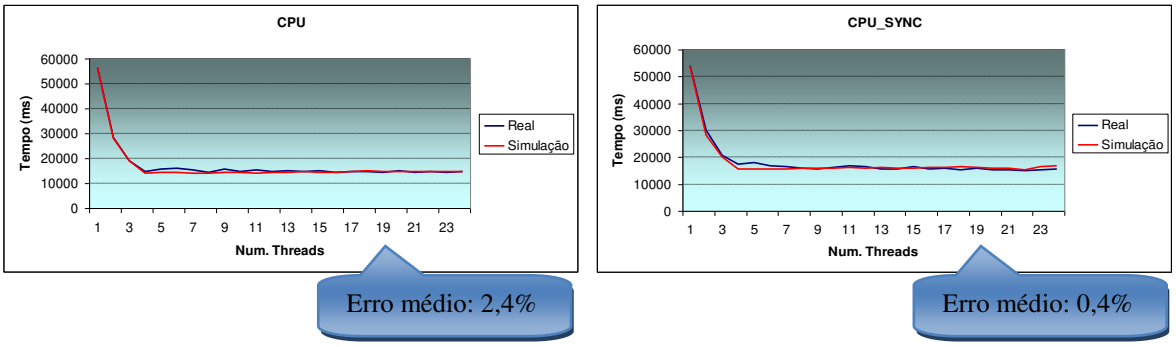

Figura 15: Resultados da simulação x sistema real do servidor S3 considerando as cargas CPU e CPU_SYNC

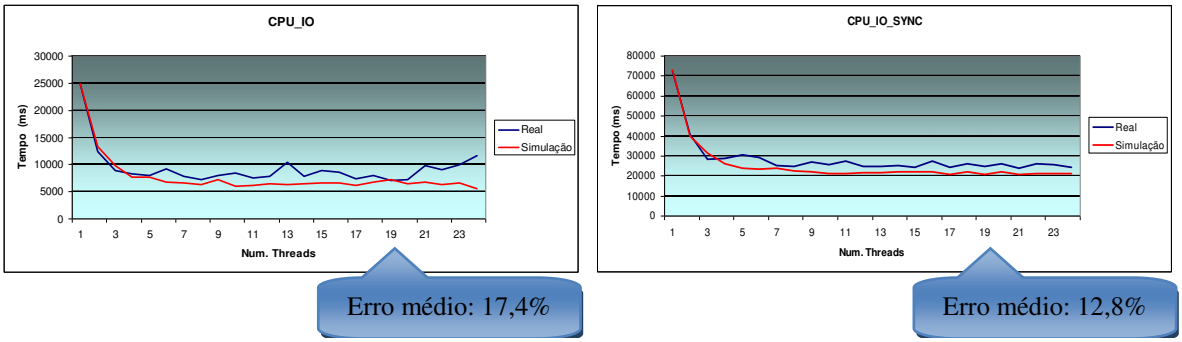

Figura 16: Resultados da simulação x sistema real do servidor S3 considerando as cargas CPU_IO e CPU_IO_SYNC

A simulação intensiva somente da carga pelo algoritmo de IO também demonstrou um resultado muito próximo dos obtidos através de medições no sistema real, conforme a figura 17.

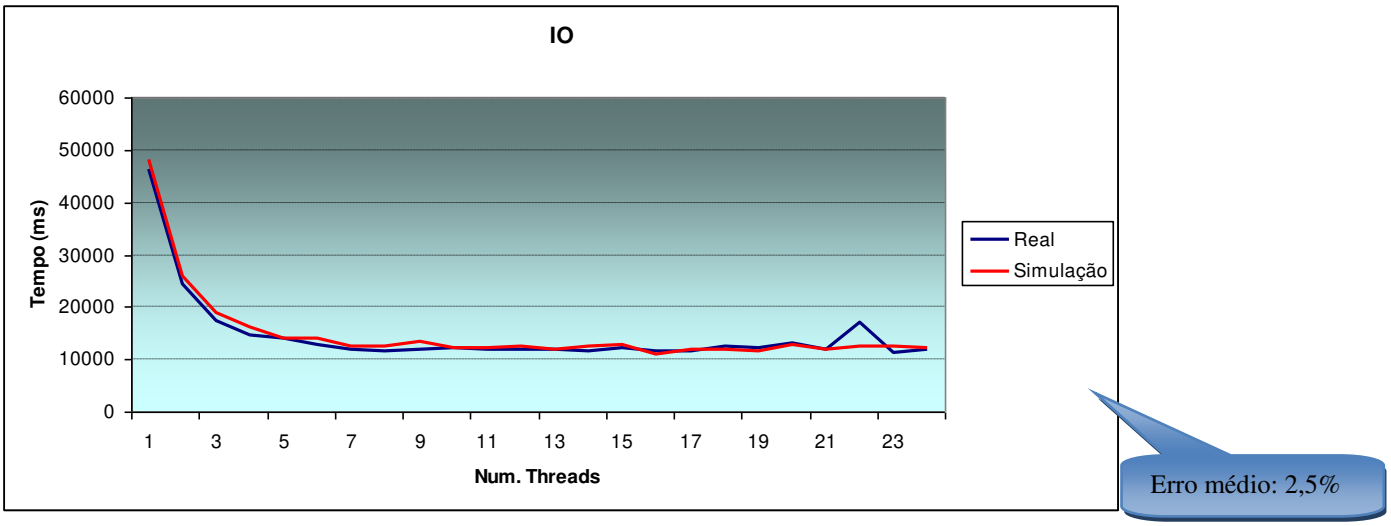

Figura 17: Resultados da simulação x sistema real do servidor S3 considerando a carga de I/O

4.3.2.4 Servidor S4

No servidor S4, os resultados obtidos através de simulação para os tipos de carga CPU, CPU_SYNC, CPU_IO e CPU_IO_SYNC foram muito próximos dos obtidos experimentalmente no sistema real com uma pequena margem de erro também para casos de CPU+IO intensivo, conforme as figuras 18 e 19.

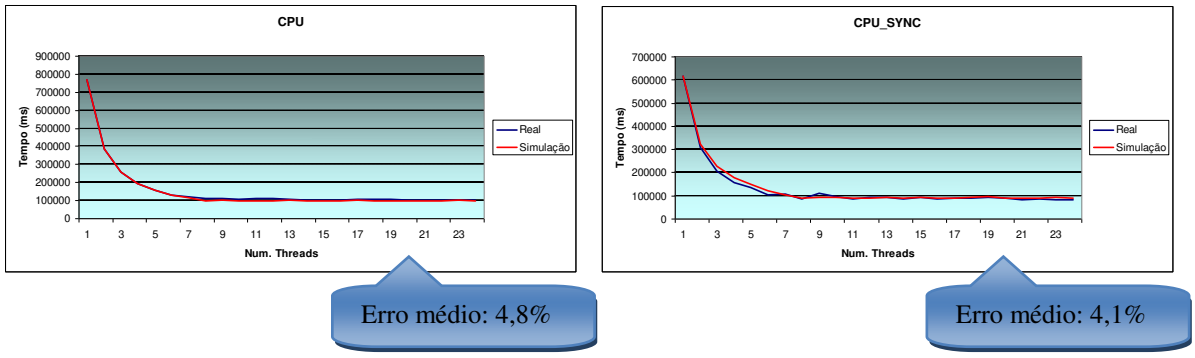

Figura 18: Resultados da simulação x sistema real do servidor S4 considerando as cargas CPU e CPU_SYNC

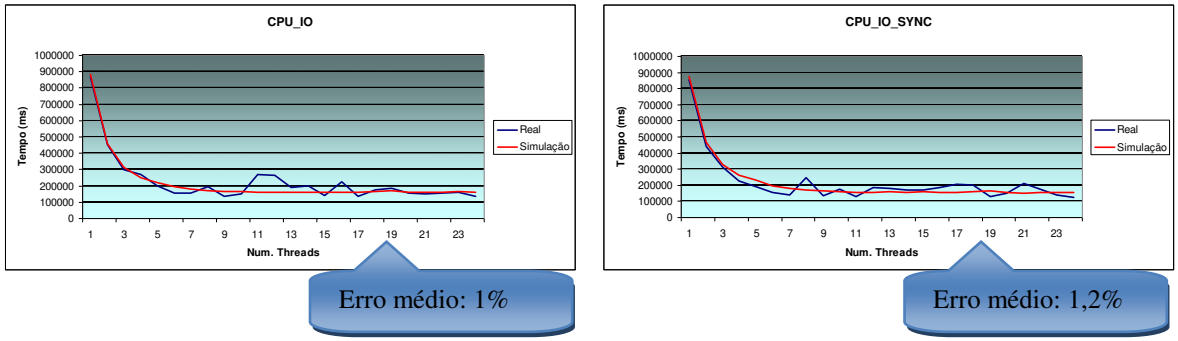

Figura 19: Resultados da simulação x sistema real do servidor S4 considerando as cargas CPU_IO e CPU_IO_SYNC

O teste intensivo somente considerando carga de IO demonstrou um resultado também muito próximo do obtido experimentalmente, como pode ser visto na figura 20.

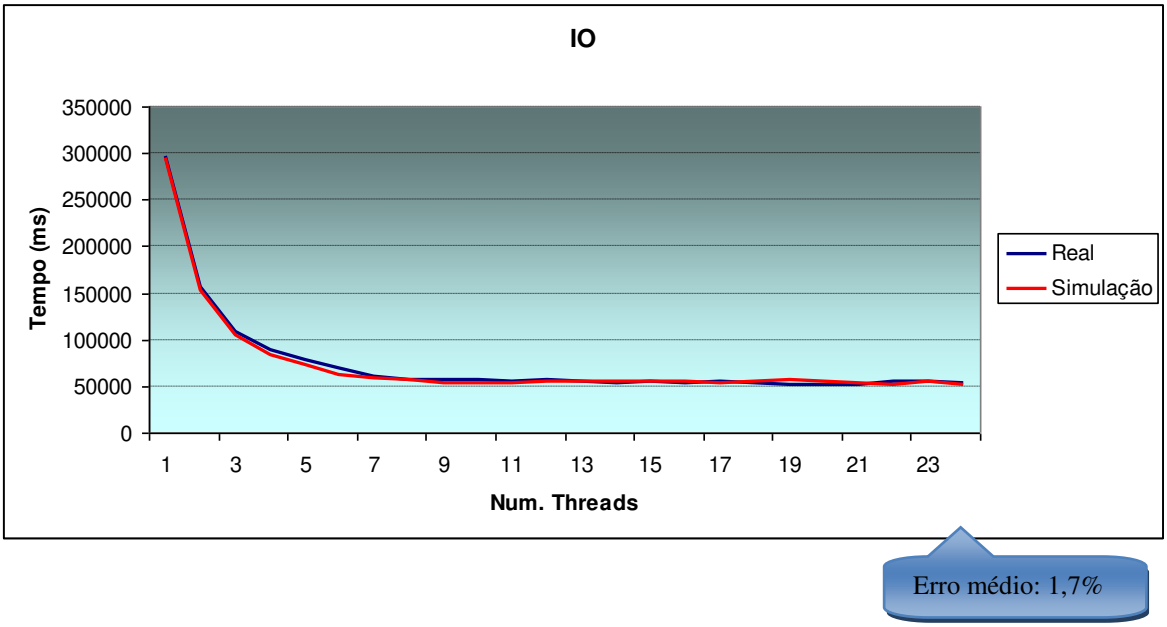

Figura 20: Resultados da simulação x sistema real do servidor S4 considerando a carga de IO

4.3.2.5 Síntese dos resultados obtidos

Utilizando-se a distribuição T-Student para calcular o intervalo de confiança do erro médio com grau de confiança de 99%, obtemos os seguintes resultados para cada teste. Os resultados estão satisfatórios para a maioria dos testes, com uma maior variação nos casos de simulação intensiva de I/O:

Tabela 1 Médias e intervalos de confiança para os diversos cenários

			Intervalo de confiança (T-Student) 99%	
Servidor	Teste	Erro Médio	De:	Até:
S1	CPU	-2,65%	-6,16%	0,87%
S1	CPU+Sync	3,47%	-1,78%	8,72%
S2	CPU	0,55%	-1,73%	2,83%
S2	CPU+Sync	-7,10%	-12,04%	-2,17%
S2	IO	-2,06%	-18,76%	14,64%
S3	CPU	2,36%	0,21%	4,51%
S3	CPU+Sync	0,44%	-2,60%	3,49%
S3	CPU+IO	17,41%	9,00%	25,82%
S3	CPU+IO+Sync	12,81%	8,48%	17,15%
S3	IO	-2,53%	-7,11%	2,05%
S4	CPU	4,76%	2,58%	6,93%
S4	CPU+Sync	-4,06%	-8,07%	-0,06%
S4	CPU+IO	0,99%	-9,23%	11,21%
S4	CPU+IO+Sync	-1,18%	-12,10%	9,74%
S4	IO	1,73%	-0,56%	4,02%

4.3.3 Considerações sobre os resultados obtidos

Através dos resultados obtidos, foi possível chegar às seguintes considerações importantes para o modelo proposto:

- Os programas normalmente podem ter uma sincronização de código através de semáforos e acesso ao disco influenciando muito o desempenho final do sistema, principalmente com a utilização de muitas threads de processamento. Esse ponto foi sanado com a calibração do modelo através da definição do fator de sincronismo de código.
- Existem diferenças consideráveis entre máquinas com CPUs HT e CPUs comuns: o *throughput* (vazão) é diferente e bem menor nos processadores comuns. O desempenho de um processador com tecnologia *HyperThreading* não corresponde a exatamente o desempenho de 2 processadores comuns, pois possui somente núcleos duplicados, mantendo 2 estados de processamento ao mesmo tempo, otimizando somente a troca de contexto no sistema operacional. Neste caso, a geração experimental da fórmula de degradação de desempenho em processadores HT, tornou possível uma representação precisa do sistema.

- A sincronização de código através de semáforos de controle degrada exponencialmente o tempo de resposta final conforme aumentamos o número de processos concorrentes através do uso de *threads*. Este comportamento foi observado medindo experimentalmente o tempo de resposta dos sistemas que possuiam este recurso.
- A utilização de discos influencia muito o desempenho dos sistemas de I/O intensivo, como observamos nos resultados obtidos. Um problema de desempenho nestes dispositivos pode causar perdas consideráveis de desempenho dos sistemas.
- Os resultados obtidos, apesar de possuirem variações maiores em alguns casos, apresentaram uma precisão totalmente aceitável no processo de planejamento de capacidade de sistemas. As variações foram maiores em casos de utilização intensiva de disco. Em trabalhos futuros podemos considerar esse tipo de carga de uma forma mais detalhada, observando os tipos de montagem de *storages*, *caches* e *buffers* do sistema operacional, etc (SHRIVER, 1997). Estes detalhes tendem a deixar a simulação um pouco mais precisa, porém trarão um desempenho inferior no processo de simulação, portanto devem ser estudados mais detalhadamente para que realmente se tornem efetivos.

Capítulo 5
Caso de estudo

Após o modelo ser validado e calibrado através de uma ferramenta de *benchmarking*, este foi aplicado em um sistema de tarifação de uma grande empresa brasileira do setor de telecomunicações. Este estudo foi feito de acordo com a empresa que estava desenvolvendo o sistema. Esta aplicação realiza a tarifação de chamadas telefônicas realizadas por meio da rede telefônica da própria desta empresa e interconexões com redes de outras empresas. Os dados das chamadas telefônicas são armazenados em CDRs (*Call Data Records*) pelos bilhetadores (registradores de chamadas) antes de chegarem ao sistema de tarifação. Os CDRs possuem dados como a origem, destino, hora de início e duração das chamadas e são gerados no bilhetador de chamadas e posteriormente enviados em um diretório específico da aplicação de tarifação. Quando um arquivo CDR é gerado e chega ao diretório de entrada definido no sistema, a aplicação cria uma *thread* que trata este arquivo e realiza todo o processo de tarifação. Este processo é considerado *CPU Bound* (a maior parte do tempo utiliza somente recurso de ciclos de CPU). O acesso a disco desta aplicação é muito pequeno comparado com a utilização de CPU.

Existe também uma aplicação externa que é responsável em verificar se ainda existem arquivos neste diretório para processamento e armazená-las. A aplicação em questão armazena os blocos a serem processados no banco de dados (processo muito rápido) e cria referência a eles em uma fila de processamento interna. Após este processo inicial, o software começa a aplicar as tarefas de tarifação em cada bloco de acordo com as regras definidas previamente no banco de dados de acordo com regras de negócio da companhia de telecomunicação. Estas regras são aplicadas diretamente aos blocos de CDRs e no final do processamento temos todos os CDRs tarifados. Neste momento, os CDRs tarifados podem ser enviados a outro processo como faturamento ou cobrança da empresa de telecomunicação.

O processo descrito acima é representado pela figura 21:

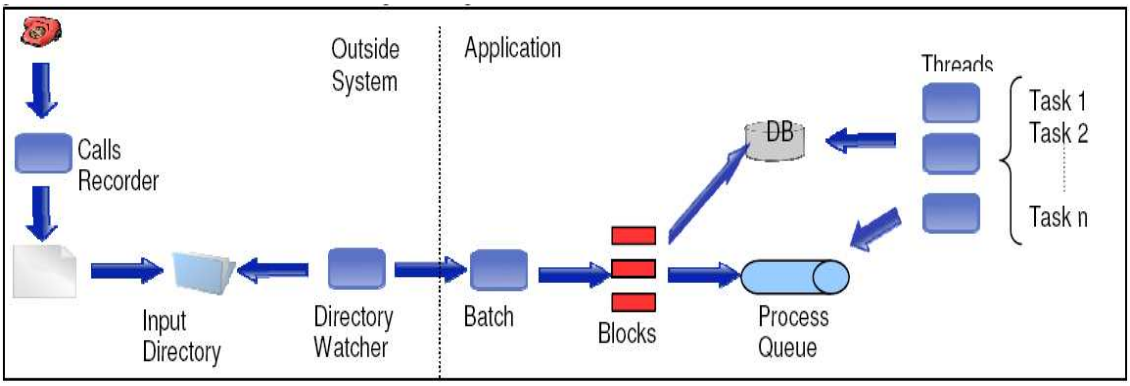

Figura 21: Sistema *batch* de tarifação telecom

O modelo utilizado foi o modelo proposto nas seções anteriores, sendo que somente a parte da direita da Figura 21 (lado da aplicação) foi considerada na simulação.

Foram realizadas três simulações distintas, considerando o impacto de novos recursos computacionais, sendo elas:

- Sim1: simulação considerando somente CPU (Erro médio: 26,1%)
- Sim2 : simulação considerando CPU+SYNC (Erro médio: 8,4%)
- Sim3 : simulação considerando CPU+SYNC+IO (Erro médio: 2,4%)

A utilização de 3 tipos de simulação distintos foi necessária para provarmos que com estas considerações foi possível representar precisamente um sistema de execução em lotes, para várias execuções com configurações de número de *threads* variadas.

O modelo mais preciso foi o da simulação SIM3 que considerava CPU, sincronização de código e I/O. O resultado obtido através das simulações se aproximou muito do resultado experimental do sistema real, o que nos fez considerá-lo como preciso o suficiente para uma análise de capacidade de sistemas deste tipo. Poderíamos considerar outros tipos de gargalos no sistema, mas a simulação ficaria mais lenta e utilizaria mais recursos computacionais ao ser executada. Além disso, o custo de implementação da ferramenta seria bem maior, não compensando o esforço pelo resultado a ser obtido.

Os resultados podem ser vistos graficamente conforme a figura 22:

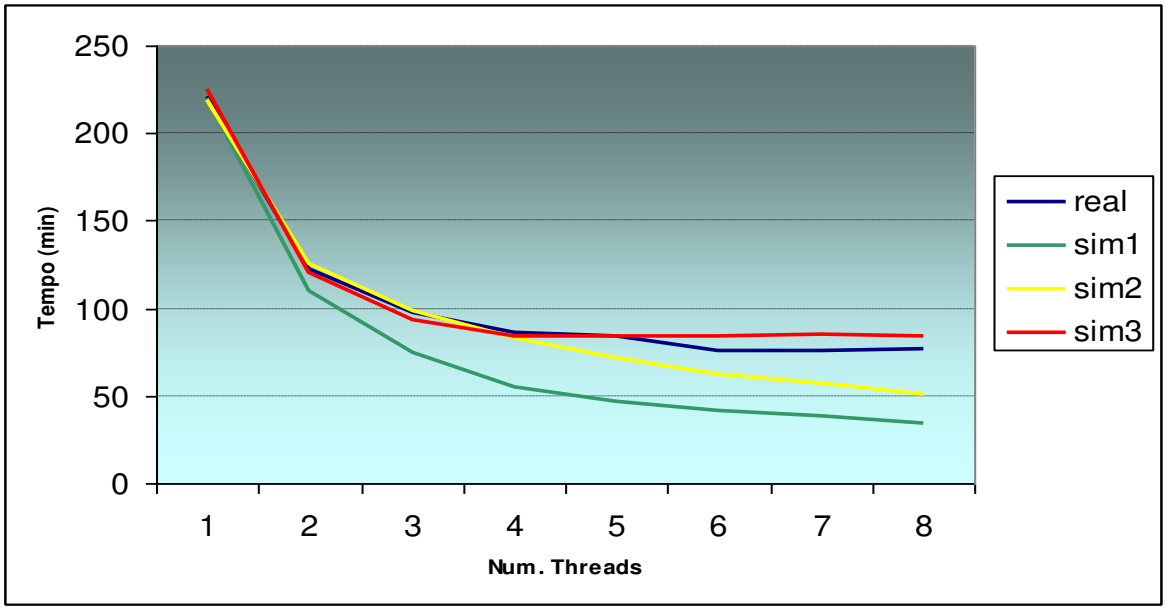

Figura 22: Resultados de simulações x medições no sistema de tarifação telecom

Capítulo 6
Implementação e execução da ferramenta

6.1 Implementação

A implementação da ferramenta de planejamento de capacidade foi realizada utilizando a linguagem de programação Java, altamente utilizada pelas fábricas de software principalmente pela portabilidade que ela oferece, permitindo que o programa desenvolvido nesta linguagem possa ser utilizado em qualquer sistema operacional sem precisar de adaptação e recompilação de código. Esta linguagem que antes era considerada lenta, hoje possui desempenho muito superior à muitas linguagens de programação mais antigas e não portáveis. Atualmente também existem inúmeros software chamados de *profilers* que servem para detectar problemas de má implementação com foco em desempenho no código dos aplicativos implementados nesta linguagem, gerando um resultado no desempenho final do aplicativo analisado, quando são otimizados os códigos apontados como maiores ofensores.
A ferramenta utilizada na programação deste aplicativo foi a Netbeans que possui uma interface que permite desenvolver interfaces gráficas com maior agilidade e facilidade além de possuir recursos internos como *profilers* integrados para garantir um bom desempenho e aplicação de boas práticas de programação.

6.1.1 Descrição dos componentes do simulador

A seguir apresentamos cada um dos componentes internos do simulador implementado, representados internamente por classes Java. O Apendice 1 mostra os diagramas UML de cada uma das classes Java aqui descritas.

com.cptool.Calendar

Classe implementada em linguagem Java que representa o calendário de eventos do simulador e define a ocorrência destes. Possui atributos que armazenam o estado dos eventos do calendário e define o próximo evento que ocorrerá através do método getFirstEvent().

com.cptool.DistGenerator

Classe geradora de números aleatórios para as seguintes distribuições estatísticas:

- Poisson
- Exponencial
- Geométrica
- Erlang

Esta classe é utilizada no momento da definição de qual distribuição estatística será utilizada para a geração da carga no sistema.

com.cptool.Event

Classe que representa um evento de simulação, armazenando informações de quando este ocorrerá e qual recurso irá utilizar (CPU, Disco, Memória, etc). Cada evento armazena a transação que está sendo tratada por ele e a *thread* que está executando esta transação além do servidor responsável pela requisição.

com.cptool.Logger

Classe responsável pela geração de *logs* contendo todos as etapas realizadas durante a simulação do sistema, quando o modo *trace* de simulação está ativado. Os *logs* foram muito importantes no momento da validação da ferramenta implementada e comparação dos resultados obtidos por ela e a ferrameta Extend. Em alguns casos, os *logs* gerados podem se tornar muito extensos resultando um baixo desempenho de execução da simulação, pela escrita intensiva em disco, porém no dia a dia do trabalho de planejamento de capacidade não são utilizados, não impactando a execução da ferramenta.

com.cptool.Simulator

Classe principal da ferramenta que representa o simulador como um todo, controlando as chegadas e as saídas de transações no sistema analisado. Esta classe possui como principais atributos os dados de cada um dos servidores e recursos computacionais, além de apresentar operações de atualização de tempo e calendário de eventos.

com.cptool.State

Classe que representa um estado da simulação, contendo informações como o número de transações ativas que ainda devem ser processadas, transações que já entraram no sistema e transações que já sairam após a execução completa do ciclo de processamento.

com.cptool.ThreadObj

Classe que representa uma *thread* de processamento, armazenando dados como o número da transação a ser processada e o tempo de processamento cumulativo desde o início da simulação. Este tempo é importante para que seja feito o tratamento do balanceamento entre as *threads*, procurando manter um tempo de processamento bem próximo entre todas elas.

com.cptool.simObjects.SimObject

Classe que representa um objeto geral de simulação. Um objeto de simulação é um recurso computacional e pode ser uma CPU, uma área de memória ou um disco. As classes que representam cada um destes recursos extendem as características desta classe genérica e implementam características específicas.

com.cptool.simObjects.CPU

Classe que representa um processador e sua carga de processamento. Este processador pode ou não ser definido como *HyperThreading*. Esta classe possui somente um atributo *loadFactor* que define a carga do processador e é utilizado para o balanceamento de carga entre os processadores dos servidores.

com.cptool.simObjects.Disk

Classe que representa um disco, o número de requisições feitas a este, volume de dados transferidos (escritos+lidos) e tempo de utilização total.

com.cptool.simObjects.Memory

Classe que representa a memória física do sistema, armazenando informações como memória utilizada, memória total disponível para os processos e tempo médio de acesso por *byte* acessado, utilizados nos cálculos da simulação.

com.cptool.simObjects.Queue

Classe que representa uma fila de espera para um determinado recurso. Essa classe é genérica para qualquer tipo de recurso e possui atributos tais como o número de transações que estão na fila, número de chegadas e saídas de processos na fila, tempo total de utilização, etc. Possui métodos que definem qual será a próxima transação a ser atendida na fila, mostram o tamanho atual da fila e o tempo de espera total por todas as transações. Estes dados são muito utilizados no momento da geração dos resultados da simulação para cada fila.

com.cptool.simObjects.Server

Classe que representa um servidor que fará o processamento a ser definido na simulação. Mantém atributos que especificam o nome do servidor, as threads ativas, transações em processamento e o contador de entradas e saídas de transações no servidor.

com.cptool.results.Queue

Os métodos desta classe armazenam estatísticas de cada uma das filas dos recursos disponíveis no sistema, tais como tamanho médio, tempo médio de espera, tamanho máximo atingido, tempo máximo de espera na fila, chegadas, saídas e utilização média. A tela de resultados consulta as estatísticas de cada uma das filas diretamente nesta classe após o término da execução da simulação.

com.cptool.results.Resource

Esta classe representa e armazena estatísticas de utilização de cada um dos recursos (CPU, memória e discos). Ela é utilizada intensivamente na exibição dos resultados obtidos na simulação.

com.cptool.results.Results

Nesta classe são mantidos os resultados gerais das simulações que são usados na consulta na tela de resultados, após a realização da simulação. Ela possui referência para o resultado de cada recurso e cada fila utilizados no planejamento de capacidade.

com.cptool.results.ServerResult

Para o tipo de simulação considerando uma rede de servidores, esta classe armazena resultados de tempo de execução, chegadas e partidas para cada um dos servidores.

com.cptool.results.Simulation

Classe que controla e armazena os resultados do processo de simulação como um todo, guardando informações do tempo total de execução e o número de chegadas e saídas de transações no sistema.

6.1.2 Bibliotecas Utilizadas

No desenvolvimento da ferramenta descrita nesta dissertação de mestrado foram utilizadas duas bibliotecas que proporcionaram facilidade e rapidez no processo de implementação do simulador, permitindo que não nos preocupássemos com detalhes que não eram o foco no trabalho, dando um grau de profissionalismo maior para a ferramenta. Seus objetivos principais foram mostrar graficamente estruturas da simulação, facilitando o entendimento da definição da topologia do sistema e visualização de resultados. Estas bibliotecas são descritas a seguir:

JFreeChart

JFreeChart é uma biblioteca baseada em *software* livre desenvolvida em linguagem Java, responsável pela geração de gráficos. Ela é capaz de gerar gráficos muito bem apresentáveis com pouco esforço de programação e baixo tempo de implementação. Esta biblioteca facilitou muito a criação dos gráficos que representam os resultados da simulação.
Como principais características desta biblioteca, podemos identificar as seguintes:

- É uma consistente e bem documentada biblioteca, suportando uma variedade muito grande de tipos de gráficos.

- Possui um design que é muito fácil de ser extendido, que permite seu uso em aplicações do lado do servidor e também aplicações executadas no lado cliente.
- Suporta muitos tipos de saída, incluindo components Swing, arquivos de imagem (PNG e JPG), e formatos de arquivo tipo vertor de gráficos (incluindo PDF, EPS e SVG).
- É uma biblioteca open source ou, mais especificamente, um *software* livre. Ela é distribuida sobre os termos da licença LGPL (GNU Lesser General Public Licence), que permite seu uso em aplicações proprietárias.

JGraph

JGraph é uma biblioteca desenvolvida em linguagem Java, utilizada para representar grafos de uma forma visual. Esta ferramenta facilitou a implementação da tela que define a topologia do *hardware* para a simulação que considera uma rede de servidores.
Como principais características desta biblioteca, podemos citar:

- Ela é puramente 100% desenvolvida em Java e integra na classe hierárquica de componente Swing, baseada no *pattern* Swing MVC.
- Totalmente documentada com muitos exemplos demonstrando várias funcionalidades.
- Pode ser usada em aplicações do lado do servidor e também aplicações executadas no lado cliente.
- Integra com outras tecnologias através de arquivos do tipo XML para banco de dados e outros sistemas, como LDAP, JMX e J2EE
- Permite funções como zoom, arrastar e soltar, etc.

6.2 Ajuste de desempenho

Para que a ferramenta garantisse um baixo tempo de resposta para o usuário final e utilizasse o mínimo possível de memória física, foram feitos vários testes de desempenho no algoritmo de simulação a fim de reduzir os principais gargalos de desempenho que geravam alto consumo de

CPU e memória e consequentemente um aumento no tempo de execução. Para isso utilizamos a ferramenta JProfiler® da empresa ej-Technologies que identifica qual dos métodos consome mais CPU e memória e quais possuem maior tempo de execução. Com isso foi possível identificar e alterar as partes do código fonte que eram as mais ofensoras de desempenho para que fossem executadas da melhor maneira possível observando boas práticas de programação na linguagem Java.

A ferramenta JProfiler® foi escolhida pela sua facilidade de uso e suas funcionalidades. Ela é uma ferramenta paga e foi adquirida pela empresa que desenvolvia o software de tarifação telecom para que fossem feitos testes de desempenho também no próprio sistema de tarifação. Existem também outras ferramentas disponíveis no mercados, algumas inclusive gratuitas, que podem ser utilizadas mesmo no momento de desenvolvimento da aplicação, evitando problemas de desempenho futuro.

A seguir iremos tratar os dois tipos de *profiling* principais e quais resultados obtivemos com cada um deles:

6.2.1 *Profiling* de CPU

A execução do *profiling* de CPU tem o principal objetivo de identificar os principais métodos e classes que consomem mais ciclos de CPU. Encontramos alguns problemas de desempenho na ferramenta implementada no início, devido à utilização indevida de algumas estruturas de loop, estruturas de dados que não tinham bom desempenho e lógicas desenecessárias em alguns casos. Com a idenficação e correção dos problemas encontrados, o tempo de resposta da ferramenta caiu para menos da metade do tempo inicial.

Atualmente, como mostra a figura 23, os métodos que mais consomem CPU são métodos de atualização do calendário de eventos (12%) e atualização do estado de simulação (11%). Estes métodos foram otimizados e é esperado que consumam mais CPU do que os outros, já que correspondem ao núcleo de toda a simulação.

Observa-se também que o consumo de CPU está bem distribuido entre os métodos existentes e não temos um só método que consome a maior parte da CPU. Normalmente quando chegamos neste cenário temos uma aplicação otimizada. Pelos tempos de execução obtidos nas simulações

realizadas podemos concluir que realmente chegamos no *target* de desempenho esperado para o processo de Planejamento de Capacidade.

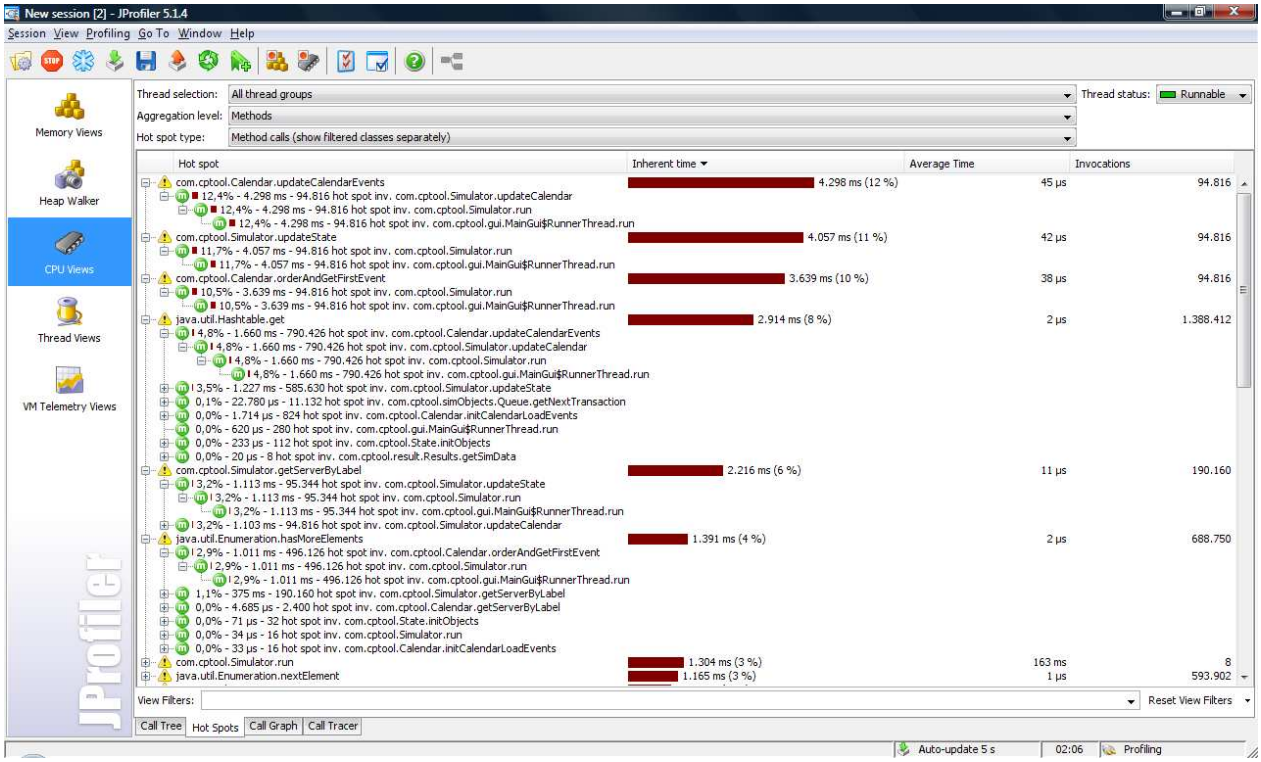

Figura 23: Tela com o resultado do *profiling* de CPU

É também possível visualizar um grafo de chamadas e verificar como é composto o tempo de resposta de determinada thread de execução em um método específico, conforme mostra a figura 24. Esta visualização gráfica do processo facilita muito o entendimento dos gargalos atuais do sistema desenvolvido.

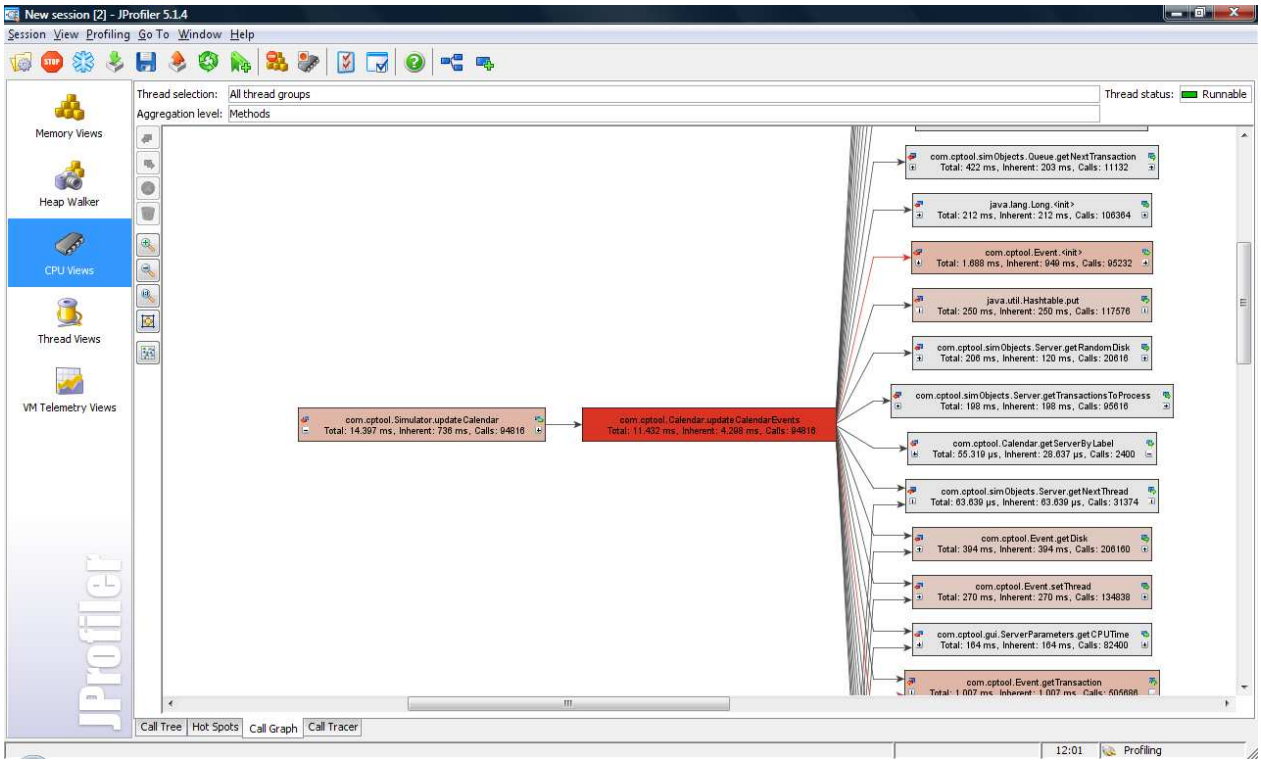

Figura 24: Grafo contendo os tempos das chamadas dos principais métodos

6.2.2 *Profiling* de Memória

O *profiling* de memória foi importante para que fosse identificado qual método é o que mais consome memória armazenando e mantendo muitos objetos na memória. Muitas vezes, isso é gerado devido ao uso de estruturas de dados desnecessárias ou erros de programação, mantendo variáveis ativas em um escopo desnecessário.

Com o *profiling* de memória foi possível identificar pontos da simulação onde era consumido muita memória desnecessariamente durante a alocação dos objetos de simulação. Atualmente, a aplicação utiliza muito pouca memória física, chegando a 100 MBytes aproximadamente nas simulações mais complexas envolvendo uma rede de servidores.

Atualmente o método que mais consome memória é o método que mantém o calendário de eventos, porém como os eventos são consumidos conforme a simulação é realizada, a simulação libera a referência a esses objetos rapidamente, fazendo que com o próximo *garbage collection* (mecanismo de limpeza de memória da *Java Virtual Machine*), esses objetos sejam eliminados da memória, liberando-a para novos processos.

Na figura 25 podemos ver a tela que mostra quais métodos mais consomem memória no simulador:

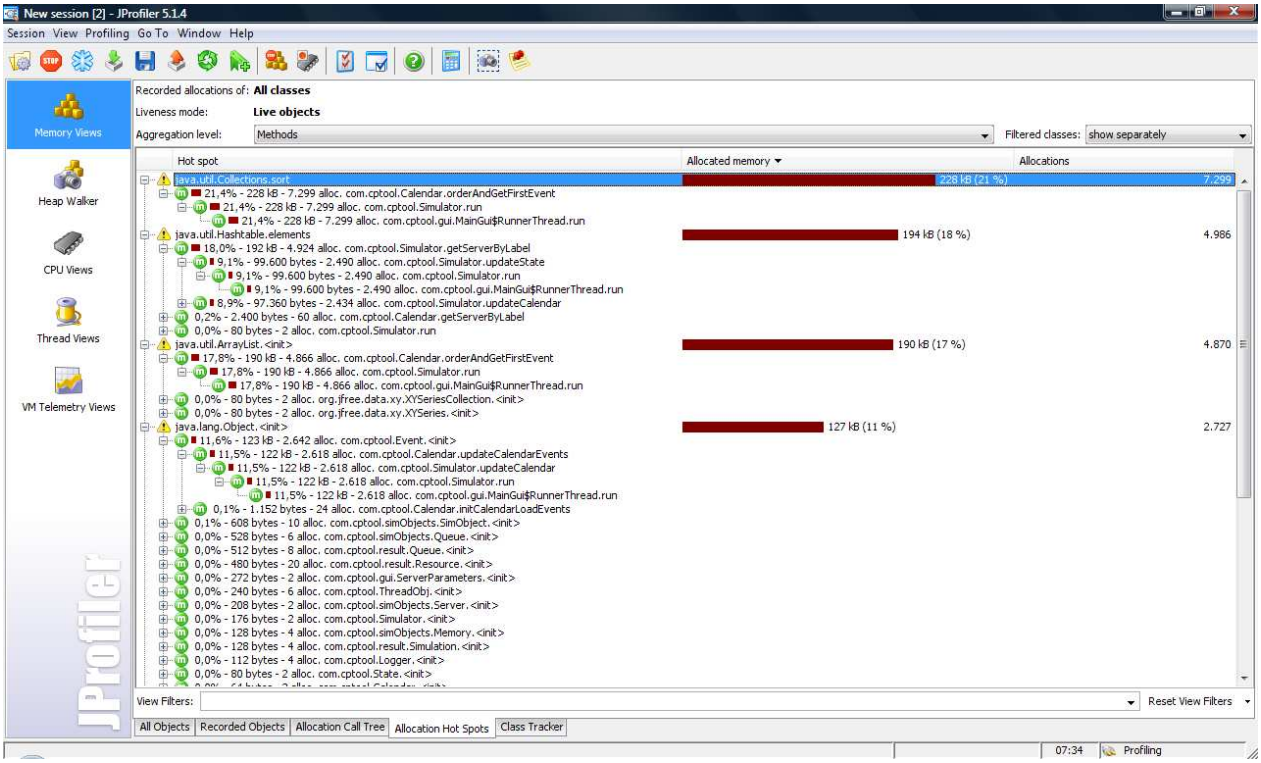

Figura 25: Tela com o resultado do *profiling* de Memória

6.3 Execução

A execução da ferramenta desenvolvida é um processo muito simples e composta claramente de alguns passos que devem ser seguidos para que sejam obtido resultados precisos. O processo de execução da ferramenta pode ser detalhado nas seguintes etapas:

6.3.1 Calibração sistema/simulação

A primeira etapa do processo consiste em calibrar o modelo com dados reais ou dados esperados de carga e tempo de resposta do sistema que ainda vai ser desenvolvido. Primeiramente, é necessário conhecer ou prever quanto tempo uma transação irá demandar de cada um dos processadores, dos discos e da memória (McCARTHY, 1996). Através destes dados, podemos fazer simulações para a validação destes dados, comparando com resultados esperados ou reais

de execução do sistema alvo existente, ignorando o sincronismo de código da aplicação, executando os testes com o mínimo de paralelismo possível.
Se tivermos disponível o sistema real nesta etapa, o ideal é executá-lo somente com uma *thread* e medir os tempos gastos em cada um dos recursos com a carga mínima possível. Essa etapa tem o objetivo principal de informar ao sistema de simulação como o sistema real se comporta em nível de transação, informando o tempo gasto em cada um dos recursos.

6.3.2 Validação do sincronismo de código

A segunda etapa do processo de execução é a validação do sincronismo presente no código do sistema que está sendo analisado. É uma tarefa relativamente fácil e simples de ser realizada, onde é suficiente comparar a estimativa ou resultado medido real com o resultado da calibração para mais de uma *thread* até o número de processadores disponíveis no ambiente de testes. Com a comparação dos resultados, podemos calcular ou estimar o sincronismo de código e a validade dos resultados obtidos com as simulações.
Caso o sistema ainda não exista, trata-se de uma estimativa conforme conhecimento do negócio e quanto se espera que o desempenho do código a ser implementado deve escalar com o aumento de processos paralelos. Neste caso o usuário precisa definir o valor do parâmetro syncFactor, cujo valor é definido de 0 a 1, onde o valor 0 representa um sistema com nenhum sincronismo de código e o valor 1 representa um sistema impossível de escalar, onde somente uma thread é executada por vez devido a uma regra específica de negócio ou limitação de implementação.

6.3.3 Execução do Planejamento de Capacidade

Depois das duas etapas anteriores, é possível visualizar uma curva de desempenho de um sistema para diferentes configurações de *hardware* e saber o impacto da variação do número de *threads*. É possível, por exemplo, conhecer o comportamento da aplicação em um *hardware* que tem o dobro de recursos de CPU ou de discos. É possível identificar o principal gargalo da aplicação e qual é o próximo limitante que vai aparecer se aumentarmos o poder de processamento para eliminar o primeiro gargalo (MENASCÉ & ALMEIDA, 2001). Com estas etapas, podemos

predizer o desempenho de um sistema em vários tipos de *hardware,* com recursos computacionais bem distintos.

A complexidade da execução de cada etapa depende muito do conhecimento geral do comportamento dos sistemas envolvidos e o resultado das variações em número e qualidade dos recursos computacionais que queremos conhecer. Não importa simplesmente o fato do sistema ser muito complexo e possuir vários recursos de hardware para sua execução, porque o modelo e o processo de simulação são muito rápidos para o modelo de simulação proposto e uma re-execução alterando parâmetros pode nos trazer informações precisosas de qual *hardware* a empresa deve comprar para garantir o desempenho do sistema.

Com a execução desta etapa, temos como resultado o planejamento de capacidade do sistema analisado, contendo a previsão de utilização de cada fila e cada recurso de hardware envolvido no ambiente computacional alvo e o tempo de execução esperado.

6.4 *Single Simulation*

No modo *Single Simulation*, é realizada a simulação do comportamento de somente um servidor e este é o principal foco deste trabalho no início das pesquisas, já que o sistema de tarifação Telecom alvo de análise deve ser executado somente utilizando uma máquina. Na tela principal deste tipo de simulação é possível adicionar parâmetros do simulador, do *hardware* envolvido e da carga de trabalho no sistema, como pode ser visto na figura 26:

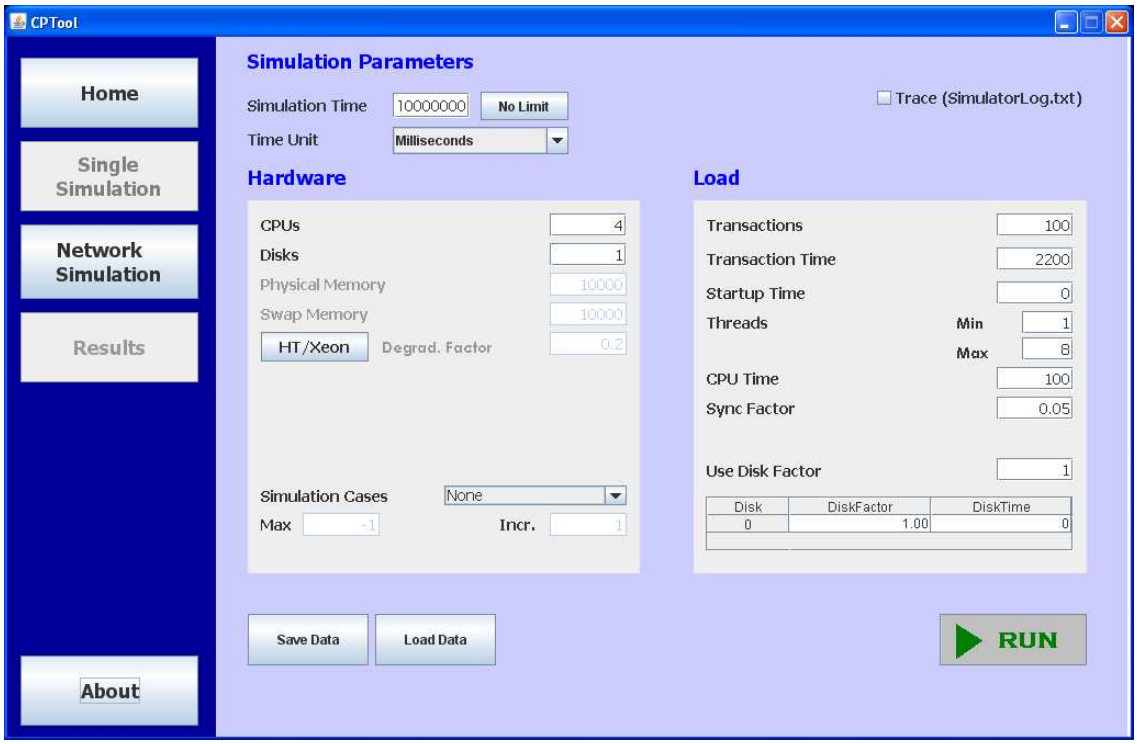

Figura 26: Tela onde são definidos os parâmetros de hardware e carga no modo Single Simulation

O usuário da ferramenta tem a opção de definir um limite de tempo a ser simulado ou deixar o processamento fluir até que não hajam mais processos a serem executados pelo sistema alvo. É possível também salvar e carregar dados que definem uma simulação em arquivos com formato específico para que futuras simulações possam ser realizadas economizando o tempo necessário para a parametrização. É permitido também definir o modo de trace como ativo, gerando um log para cada ocorrência de evento caso o usuário queira entender melhor a execução de cada passo da simulação. Esta opção, quando habilitada, aumenta consideravelmente o tempo de resposta do teste, pois são geradas muitas linhas neste arquivo e o gargalo do programa de simulação passa então a ser a escrita intensiva em disco.

6.5 *Network Simulation*

Com o modo *Network Simulation* podemos verificar o comportamento de uma rede de computadores (MONGUL, 1995) e considerar também os tempos gastos com tráfego de pacotes

na rede. Esta funcionalidade foi adicionada recentemente e utiliza os mesmos métodos da *single simulation* adicionando o detalhe do tempo gasto em uma rede entre os servidores envolvidos.

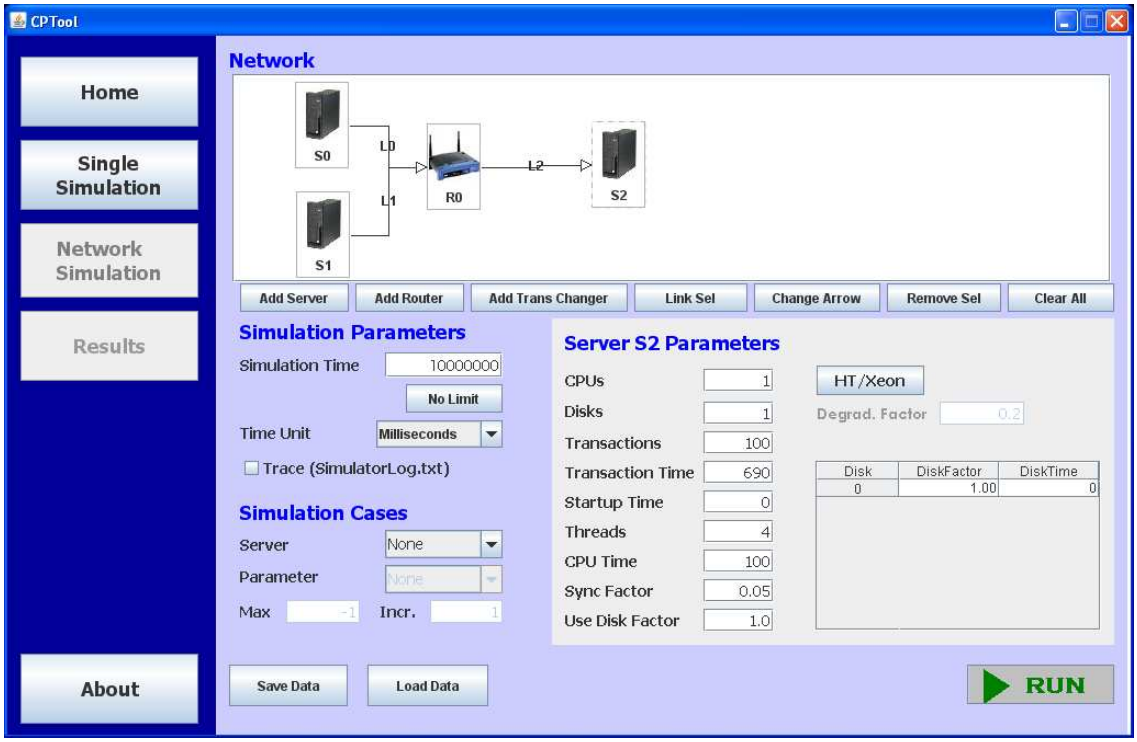

Figura 27: Tela onde são definidos os parâmetros de hardware e carga no modo Network Simulation

Nesta tela, representada pela figura 27, é possível definir quais são os servidores disponíveis na parte superior de uma forma icônica, as ligações onde a transação flui entre eles e as parametrizações de carga e hardware de cada máquina. Também é possível, assim como o modo *Single Simulation*, salvar e carregar dados em arquivos específicos, além de utilizar o modo *Trace*.

Este tipo de simulação ainda não foi validada em nenhum software real de grande porte, porém em trabalhos futuros poderemos ter a necessidade de utilizá-la e esta extensão do modelo poderá ser testada e adaptada se necessário.

6.6 *Result Viewer*

A tela *Result Viewer*, representada pela figura 28, disponibiliza, após a execução de uma simulação, dados como a utilização de recursos, tempo de resposta e utilização das filas de cada um dos recursos em cada um dos servidores envolvidos na simulação.

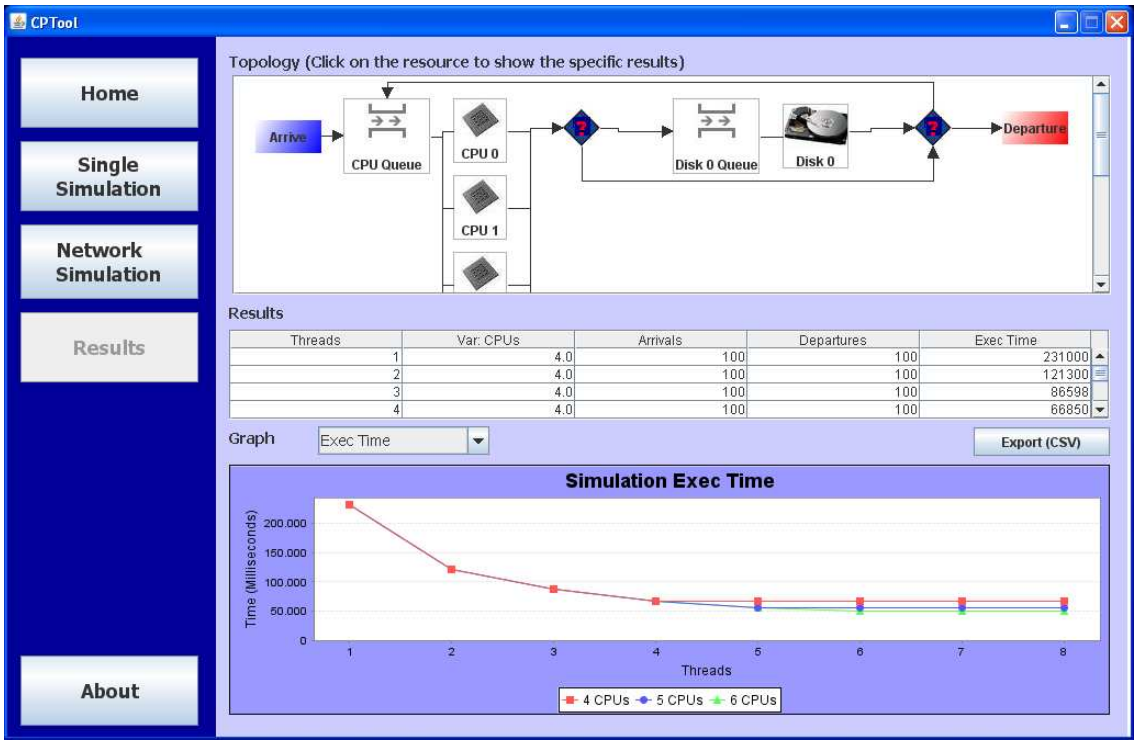

Figura 28: Tela onde são mostrados os resultados da simulação

Esta tela é interativa e ao clicar em cada um dos recursos é possível ver os resultados de simulação de cada um no formato tabela e abaixo o gráfico de cada recurso variando todos os casos de hardware/paralelismo da simulação.

6.7 Análise dos tempos de execução da ferramenta

Os resultados apresentados aqui comprovam que a ferramenta possui um tempo de execução bem baixo, mesmo para testes que envolvem uma complexidade maior e muitas variações de casos de simulações. O tempo de resposta da simulação pode ser menor do que o da execução do programa real sendo analisado, já que o tempo gasto em uma simulação orientada a eventos não aumenta sua complexidade conforme variamos o tempo gasto para cada transação. Quanto maior o tempo gasto para o processamento de cada transação no sistema real, maior vai ser essa diferença. O que influencia no tempo de simulação são o número de variações de simulações

alterando os recursos disponíveis e principalmente o número de transações envolvidas no processo.

Esses resultados podem ser vistos na tabela 2 onde são mostrados os tempos de execução para cada caso de simulação, variando ou não o número de transações, CPUs, discos e *threads*. Observa-se que o tempo de resposta da simulação não aumenta consideravelmente quando temos mais recursos computacionais envolvidos no servidor.

O tempo tende a variar linearmente conforme aumentamos o número de casos de teste mas a complexidade da simulação tende a aumentar exponencialmente conforme o número de transações que a simulação trata. O gerenciamento interno do estado das transações torna-se custoso para a simulação, porém, como mostrado na tabela o tempo ainda é muito baixo para os piores casos e muitas vezes não precisamos simular tantas transações quando fazemos o trabalho de planejamento de capacidade. Como por exemplo, no sistema de tarifação analisado, eram processadas em média 100 transações somente. Ainda em casos onde o número de transações for muito grande, inviabilizando a simulação podem serem feitas simplificações, processando menos transações, e através de cálculos chegar facilmente nos resultados finais esperados.

Os tempos de resposta foram em todos os casos muito inferiores aos obtidos através da ferramenta Extend devido à otimizações na simulação específica deste processo voltado ao planejamento de capacidade. Como a ferramenta Extend é uma ferramenta genérica de simulação, não é possível abstrair alguns passos de simulação que puderam ser otimizados no nosso caso. Com isso obtivemos tempos de resposta menores e perfeitamente aceitáveis para uma ferramenta de simulação. As simulações foram executadas em um laptop com um processador Intel® Core 2 Duo com 2.4 Ghz, 2GB de RAM e um HD com tecnologia SATA.

A seguir, na tabela 1, podemos ver o resultado para cada tipo de simulação.

Tabela 2 Comparação de tempos de resposta da ferramenta versus variações de recursos

Transações	Variações	CPUs		Disks		Threads		Tempo de resposta
		Min	Max	Min	Max	Min	Max	
100	8	4	4	1	1	4	4	00:00:01
500	8	4	4	1	1	4	4	00:00:01
1000	8	4	4	1	1	4	4	00:00:01
2000	8	4	4	1	1	4	4	00:00:02
3000	8	4	4	1	1	4	4	00:00:04
100	8	4	4	1	1	1	8	00:00:01
500	8	4	4	1	1	1	8	00:00:03
1000	8	4	4	1	1	1	8	00:00:06
2000	8	4	4	1	1	1	8	00:00:15
3000	8	4	4	1	1	1	8	00:00:39
100	8	8	8	1	1	1	8	00:00:01
500	8	8	8	1	1	1	8	00:00:03
1000	8	8	8	1	1	1	8	00:00:06
2000	8	8	8	1	1	1	8	00:00:16
3000	8	8	8	1	1	1	8	00:00:39
100	8	4	4	4	4	1	8	00:00:01
500	8	4	4	4	4	1	8	00:00:03
1000	8	4	4	4	4	1	8	00:00:06
2000	8	4	4	4	4	1	8	00:00:16
3000	8	4	4	4	4	1	8	00:00:39

Nos testes apresentados na tabela 3, observamos que a variação do número de CPUs ou de discos causa o mesmo impacto no tempo final de simulação, sendo este quase nulo. Novamente o que influencia o tempo de resposta final é o número de transações simuladas.

Tabela 3 Comparação de tempos de resposta variando um recurso e o número de threads

Transações	Variações	CPUs		Disks		Threads		Tempo de resposta
		Min	Max	Min	Max	Min	Max	
100	32	1	4	1	1	1	8	00:00:02
500	32	1	4	1	1	1	8	00:00:06
1000	32	1	4	1	1	1	8	00:00:20
2000	32	1	4	1	1	1	8	00:01:03
3000	32	1	4	1	1	1	8	00:02:32
100	32	4	4	1	4	1	8	00:00:02
500	32	4	4	1	4	1	8	00:00:06
1000	32	4	4	1	4	1	8	00:00:19
2000	32	4	4	1	4	1	8	00:01:04
3000	32	4	4	1	4	1	8	00:02:33

6.8 Comparação de resultados obtidos: *Extend* x ferramenta proposta

Os resultados obtidos na ferramenta proposta simulando o sistema de tarifação Telecom foram idênticos aos obtidos através da ferramenta Extend e podem ser vistos através do comparativo na tabela 4 para cada um dos casos de simulação:

Tabela 4 Resultados Extend x Ferramenta proposta

				Extend				Ferramenta Proposta			
Transações	CPUs	Disks	Threads	Tempo de processamento Simulado	Utilização média de CPU	Utilização média de Disco	Tempo de Simulação	Tempo para processamento Simulado	Utilização média de CPU	Utilização média de Disco	Tempo de Simulação
100	4	1	4	00:12:16	90,83%	36,68%	00:00:05	00:12:16	90,83%	36,68%	00:00:01
100	4	2	4	00:11:44	94,96%	19%	00:00:06	00:11:44	94,96%	19%	00:00:01
100	8	1	4	00:12:16	45,41%	36,68%	00:00:06	00:12:16	45,41%	36,68%	00:00:02
100	8	1	8	00:07:30	90,27%	73,32%	00:00:05	00:07:30	90,27%	73,32%	00:00:01

Observa-se também que o tempo de processamento simulado (estimativa de quanto a aplicação real leva para processar as transações) é muito maior do que o tempo de simulação e que o tempo de simulação da ferramenta proposta é inferior ao tempo de simulação da ferramenta de simulação Extend, devido ao trabalho de tuning específico de desempenho para o modelo de planejamento de capacidade proposto.

Capítulo 7
Conclusões

O modelo obtido através deste trabalho descrito permitiu simular o comportamento de desempenho dos programas testados com a precisão esperada para um trabalho de planejamento de capacidade de um sistema computacional do tipo *batch*.

A ferramenta contempla o modelo proposto fielmente, pois os resultados obtidos, antes validados com a ferramenta Extend, foram exatamente os mesmos obtidos na ferramenta desenvolvida. Ela também é de fácil utilização e muito intuitiva, mesmo para pessoas que não conhecem especificamente o processo de Planejamento de Capacidade e possui tempos de resposta muito rápidos e aceitáveis, mesmo quando executada em computadores pessoais, assim como proposto inicialmente.

Os resultados mostram que esta ferramenta ajuda muito no trabalho de planejamento de capacidade de um sistema *batch* da tecnologia da informação e que, considerando pontos de sincronismo e paralelismo, podemos predizer o comportamento real com uma margem de erro muito baixa. As técnicas de simulação com alguns cálculos intrínsecos, trazem benefícios sem causar impactos e dificuldades na execução do processo de Planejamento de Capacidade.

O método que a ferramenta utiliza generaliza a análise para todos os servidores presentes no mercado. Ela não se preocupa em mapear o comportamento de cada *hardware* existente, abstraindo a idéia do *hardware* final, baseando-se em um *hardware* conhecido do usuário e com características semelhantes. Preferimos trabalhar com comparações entre as arquiteturas porque temos muitos modelos diferentes de novos produtos a cada dia e esta ferramenta poderia se tornar inutitlizável se, por exemplo, uma nova tecnologia semelhante a disponível fosse lançada e quiséssemos saber o comportamento neste novo cenário. Podemos também tratar a diferença de desempenho das novas arquiteturas de *hardware* lançadas no mercado através de comparativos do *benchmarking* SPEC tornando, desta forma, a ferramenta sempre atualizada, permitindo ao usuário conhecer como sua aplicação será executada nesta nova arquitetura, apenas com alguns cálculos comparativos.

Atualmente, várias fábricas de *software* fazem esse trabalho sem nenhuma ferramenta especializada, utilizando somente cálculos matemáticos através de planilhas que são muito

imprecisas. Este é um processo muito complexo e importante que necessita mais atenção dos analistas de sistemas e muitas vezes, as grandes companhias não se preocupam muito em utilizar estas ferramentas que são muito complexas e caras, mas o prejuízo gerado por uma análise incompleta pode ser muito grande e irreversível (McCARTHY, 1996). Com a ferramenta proposta, conseguimos alcançar resultados muito próximos das ferramentas comerciais sem considerar dados não tão importantes durante a simulação.

Ela não utiliza muita potência computacional sendo muito barato seu uso em empresas de menor porte que precisam realizar a tarefa de Planejamento de Capacidade com uma análise mais precisa e não possuem recurso financeiro suficiente para comprar as ferramentas disponíveis no mercado, cujo preço é por volta de milhões de dólares. O principal objetivo para esta ferramenta é ser um *software* do tipo *freeware* com uma simplicidade de utilização, não perder o poder de precisão das ferramentas comerciais e ter um tempo de resposta baixo e aceitável.

Futuros trabalhos poderão incluir novos tipos de sistemas tais como WEB e novos modelos de desempenho. O tipo de simulação voltado a redes de computadores deverá ser testado também assim que obtivermos novos sistemas disponíveis para testes que são executados em redes de computadores.

O simulador desenvolvido apenas trata um tipo de transação, o que foi suficiente para resolver o problema em questão. Nós devemos implementar a funcionalidade de simular várias transações concorrentes com perfis diferentes uma das outras, já que em muitos sistemas é impossível generalizar o trabalho a somente um tipo de transação.

Referências Bibliográficas

ALEXANDER, C. W. Discrete event simulation for batch processing. In: Winter Simulation Conference, 2006, California. **Proceedings of the 38th Conference on Winter Simulation.** California: Winter Simulation Conference, 2006. 1929-34.

ALISPAW, J. **The Art of *Capacity Planning*.** Scaling Web Resources. O'Reilly Media, 2008, 154f. Disponivel em:<http://shop.oreilly.com/product/9780596518585.do>. Acesso em: 10/03/2012.

ALMEIDA, J. Métodos Quantitativos para Ciência da Computação Experimental – Regressão Linear. **DCC – UFMG**, 2011. Disponível em: <http://homepages.dcc.ufmg.br/~jussara/metq/aula6.pdf>. Acesso: 10/03/2012.

BAGCHI, S. *et al,* 2006. *Capacity Planning* tools for web and grid environments. In: Valuetools '06, 2006, Italy. **Proceedings of the 1st international Conference on Performance Evaluation Methodolgies and Tools.** v.180. ACM, New York, NY, 25. Disponivel em: http://doi.acm.org/10.1145/1190095.1190127 >. Acesso em: 30/11/2008.

BANK, J. *et al.* **Discrete Event Simulation**, Prentice Hall, Upper Saddle River, New Jersey, 2000.

BUZEN, J. P. Operational Analysis: An Alternative to Stochastic Modeling. **Performance of Computer Installations**. North Holland, pp. 175-194, 1978.

CARPER, I. L.; HARVEY, S.; WETHERBE, J. C. 1983. Computer *Capacity Planning*: strategy and methodologies. SIGMIS Database 14, 4 (Jul. 1983), 3-13. Disponivel em: http://doi.acm.org/10.1145/1113487.1113488> Acesso em: 15/10/2008.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems,** Kluwer, 1999.

Compuware Gomez - Application Performance Management. Disponível em: <http://www.compuware.com/application-performance-management/>. Acesso em: 10/03/2012.

DENNIN, P. J.; BUZEN, J. P. The Operational Analysis of Queueing Network Models. **Computing Surveys**, v. 10, n. 3, p. 225-261, 1978.

DENNING, P. J. Queueing Networks of Computers. **American Scientist**, 1991.

Design, Measurement, Simulation and Modeling. John Wiley and Sons, Nova York, 1991.

DHANANJAY, J. ; SRINIVASA, B.; KARTHIK, R. Supply chain sustenance in the Semiconductor Industry considering the present economic downturn. **TATA Consultancy Services.** India, 2009. Disponível em: <http://www.tcs.com/SiteCollectionDocuments/White%20Papers/Hi-Tech_WP_Supply%20Chain%20Sustenance_07_09.pdf>. Acesso em: 10/03/2012.

Extendsim – Simulation Software. Disponível em: <http://www.extendsim.com/index.html>. Acesso em: 10/03/2012.

FERRARI, D.; SERAZZI, G.; ZEIGNER, A. **Measurement and Tunning of Computer Systems**, Prentice Hall, Upper Saddle River, New Jersey, 1983.

FREIRE, R. *Capacity Planning*: conceito, importância e visão geral.IMaster, São Paulo, 2011a. Disponível em: **http://imasters.com.br/artigo/19611/gerencia/capacity_planning_conceito_importancia_e_visao_geral/**. Acesso em: 10/03/2012.

FREIRE, R. A estatística no planejamento de capacidade. **IMaster**, São Paulo, 2011b. Disponível em: < http://imasters.com.br/artigo/21843/gerencia-de-ti/a-estatistica-no-planejamento-de-capacidade>. Acesso em: 10/03/2012.

FREIRE, R. Volumes de negócio e métricas de *Capacity Planning*. **IMaster**, São Paulo, 2011c. Disponível em: <http://imasters.com.br/artigo/20618/gerencia-de-ti/volumes-de-negocio-e-metricas-de-capacity-planning>. Acesso em: 10/03/2012.

GALVES, J. D. G. Planos de contingência em TI. **TEC Hoje**, Belo Horizonte, 2006. Disponível em:< http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/219>. Acesso em: 10/03/2012.

GRACON, T. J.; NOLBY, R. A.; SANSOM, F. J. A high performance computing system for time critical applications. In: Spring Joint Computer Conference, 1971. ACM, New York, NY, 549-560. Disponivel em: http://doi.acm.org/10.1145/1479064.1479163>. Acesso em: 22/11/2008.

GRAY, J. (Ed.). **The Benchmark Handbook for Database and Transaction Processing Systems**, 2 ed., Califórnia: Morgan Kaufmann, 1993.

HENNESSY, J.; PATTERSON, D. **Computer Architecture:** A Quantitative Approach. Califórnia: Morgan Kaufmann, 1996.

INGALLS, R. G. 2001. Introduction to simulation. In: Winter Simulation Conference, 2001, Washington. **Proceedings of the 33nd Conference on Winter Simulation.** Washington: IEEE Computer Society, 2001. 7-16.

JAIN, R. **The Art of Computer System Performance:** Analysis, Techniques for Experimental

KLEINROCK, L. Queueing Systems. **Theory,** Nova York: John Wiley and Sons, v. I, 1975.

KRISHNASWAMY, U.; SCHERSON, I. A Framework for Computer Performance Evaluation Using Benchmark Sets, **IEEE Trans. Computers**, v. 49, n 12, 2000.

LAM, S.; CHAN, K. **Computer *Capacity Planning*:** Theory and Practice. Londres: Academic Press, 1987.

LAW, A. M.; KELTON, W. D. **Simulation Modeling and Analysis**, 2 ed., McGraw-Hill, Nova York, 1991.

LILJA, D. **Measuring Computer Performance**. Cambridge: Cambridge University Press, 2000.

LO, T. L. Computer *Capacity Planning* using queueing network models. In: PERFORMANCE '80, 1980. Canada. **Proceedings of the 1980 international Symposium on Computer Performance Modelling, Measurement and Evaluation**. ACM, New York, NY, 145-152. Disponivel em: <http://doi.acm.org/10.1145/800199.806158>. Acesso em: 09/11/2008.

LO, T.L. **Computer *Capacity Planning* using Queueing Network Models**. Proc. Performance 80 Corn r Toronto, 1980. 145 – 152.

McCARTHY, V. **Performance Tools Kill the Gremlins on Your Net**, Datamation, 1996.

MENASCÉ, D. A. **Using Performance Models no Dynamically Control E-Business Performance**. In: Proc. 11th GI/ITG Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems, Aachen, Germany, 11-14, 2001.

MENASCÉ, D.A.; ALMEIDA, V.A.F. ***Capacity Planning* for Web Services:** Metrics, Models, and Methods. 1st, Prentice Hall, 2001.

MENASCÉ, D. A.; GOMAA, H. A Method for Design and Performance Modeling of Client/Server Systems. **IEEE Tr. Software Eng.**, v. 26, n. 11, p. 1066-85, 2000.

MENASCÉ, D. A.; ALMEIDA, V. A. F.; DOWDY, L. W. ***Capacity Planning* and Performance Modeling:** From Mainframes to Client-Server Systems, Prentice Hall, Upper Saddle River, New Jersey, 1994.

MENASCÉ, D.A.; ALMEIDA, V.A.F.; DOWDY, L.W. **Performance by Design:** Computer *Capacity Planning* by Example. Prentice Hall, 2004.

MONGUL, J. Network Behavior of a Busy Web Server and its Clients. **Res. Rep. 95/5**, DEC Western Res. Lab., 1995.

NIÑO, J; HOSCH, F. **An introduction to programming and Object Oriented design using Java 1.5.** 2nd ed. Wiley, 2005.

PÉREZ, J. J. M. *Capacity Planning*: Introducción (I). **JJMora@arrakis.es**, 2010. Disponível em: <http://jjmora.es/capacity_planning_introduccion_i/>. Acesso em: 10/03/2010.

ROSE, F. *et. al.* A Model for the Coanalysis of Hardware and Software Architectures. In: International Conference on Hardware Software Codesign, 1996, Washington. **Proceedings of the 4th International Workshop on Hardware/Software Co-Design.** Washington: IEEE Computer Society, 1996. 94.

ROSE, C. A Measurement Procedure for Queueing Network Models of Computer Systems. **Computing Surveys**, v. 10, n. 3, 1978.

SCHRIBER, T. J.; BRUNNER, D. T. Inside discrete-event simulation software: how it works and why it matters. In: Winter Simulation Conference, 2006, California. **Proceedings of the 38th Conference on Winter Simulation**. California: Winter Simulation Conference, 2006. 118-128.

SHRIVER, E. **Performance Modeling for Realistic Storage Devices**. 1997. Tese (phD) Computer Science, New York University. 1997.

SURUAGY, J. A. Ferramentas para Avaliação de Desempenho de Sistemas. Disponivel em: < http://200.128.80.130/suruagy/fads/introducao2004.ppt>. Acesso em: 14/12/2008

System Performance Evaluation Corporation. Diponivel em: <www.spec.org>, Acesso em: 18/10/2008.

TAURION, C. Ainda precisamos de *Capacity Planning* em cloud?. TEC Hoje, Belo Horizonte, 2006. Disponível em: <**http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/1106**>. Acesso em: 10/03/2012.

TeamQuest. Disponível em: <http://www.teamquest.com/>. Acesso em: 10/03/2012.

WALDBAUM, G. Experimenting with a trace-driven simulator of a batch computing system. In: SICOSIM4. 1973. New Mexico. Proceedings of the 4th Annual Symposium on Sigcosim: Management and Evalution of Computer Technology. ACM, New York, 34-41. Diponivel em: < http://doi.acm.org/10.1145/1147396.1147401>. Acesso em: 22/11/2008.

XU, J.; CHUNG, M. J. Predicting the performance of synchronous discrete event simulation systems. In: International Conference on Computer Aided Design, 2001, NJ. **Proceedings of the 2001 IEEE/ACM international Conference on Computer-Aided Design**. NJ: IEEE Press, Piscataway, 2001. 18-23.

Apêndice I
Diagramas UML

A linguagem UML é uma linguagem de modelagem não proprietária de terceira geração largamente utilizada nos dias atuais. Ela permite a definição de como o sistema será implementado, disponibilizando uma padronização de documentação através de diagramas pré-definidos e bem especificados. Entre os diagramas disponíveis está o diagrama de classes que define principalmente as váriáveis que uma classe da aplicação utiliza e os métodos que modificam estas variáveis. Existem vários outros diagramas disponíveis nesta linguagem que utilizamos também no processo de documentação da implementação da ferramenta de simulação. Entre eles podemos citar diagramas de casos de uso, de transição de estados, de componentes, de pacotes, de atividades, de sequencia, etc.

Como o foco na dissertação não é uma documentação técnica específica de implemntação, disponibilizamos a seguir somente o diagrama de classes das principais classes implementadas na ferramenta de simulação proposta, o que já dá uma idéia geral da implementação:

com.cptool.Calendar

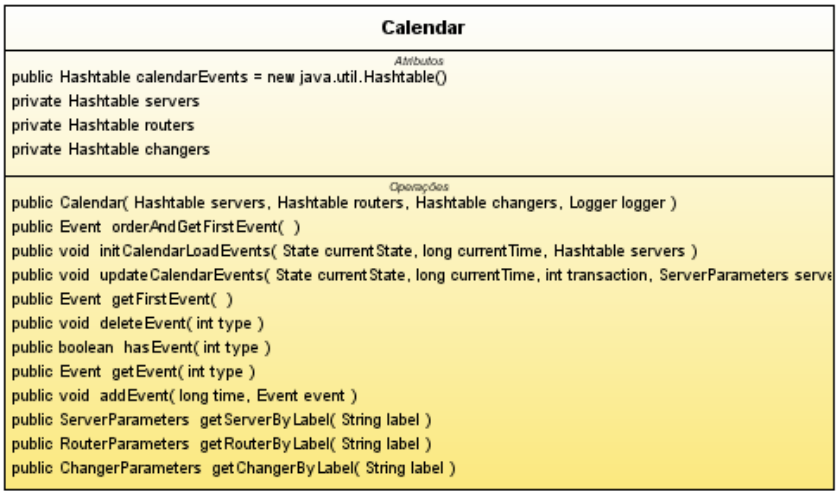

Figura 29 Diagrama de classe de com.cptool.Calendar

com.cptool.DistGenerator

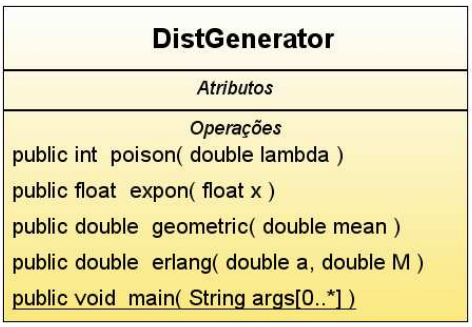

Figura 30: Diagrama de classe de com.cptool.DistGenerator

com.cptool.Event

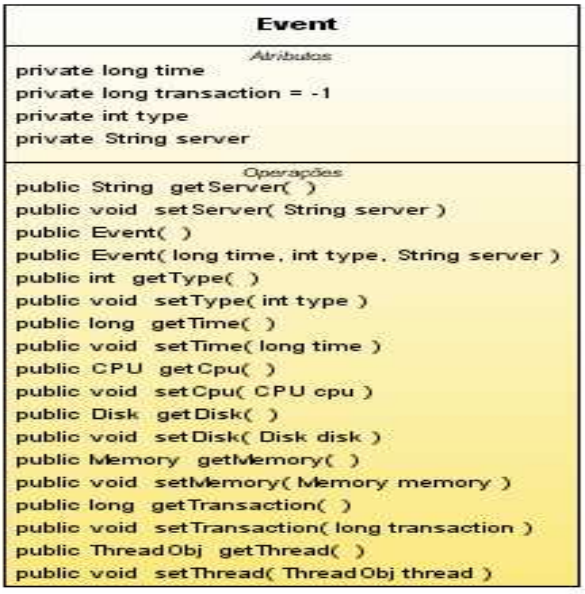

Figura 31: Diagrama de classe de com.cptool.Event

com.cptool.Logger

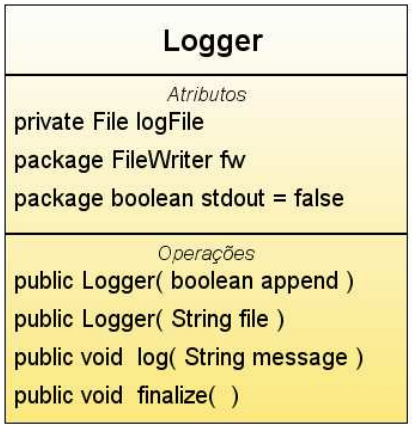

Figura 32: Diagrama de classe de com.cptool.Logger

com.cptool.Simulator

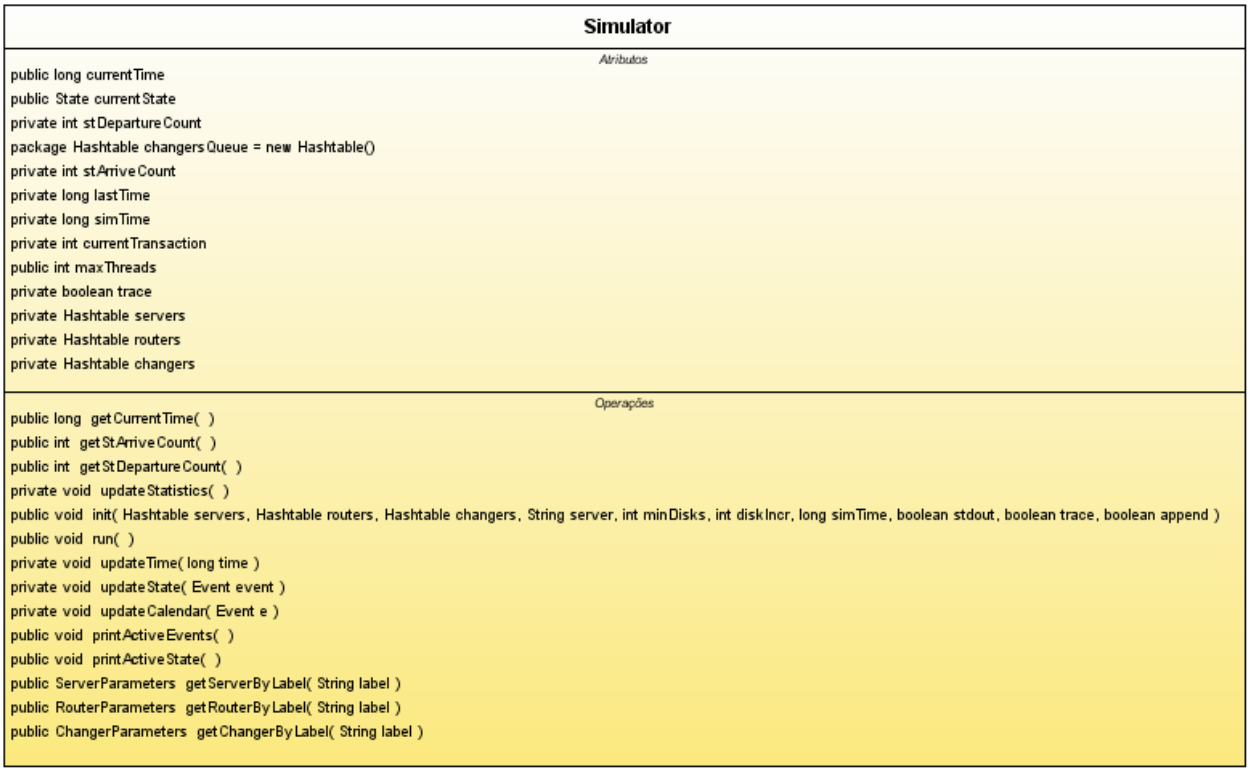

Figura 33: Diagrama de classe de com.cptool.Simulator

com.cptool.State

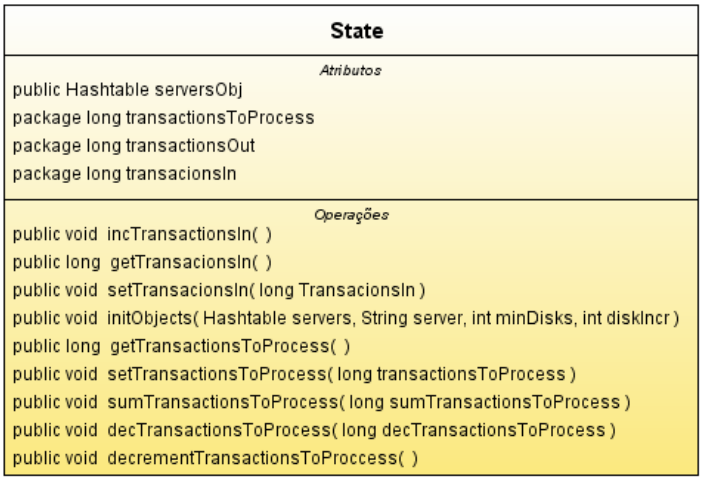

Figura 34: Diagrama de classe de com.cptool.State

com.cptool.ThreadObj

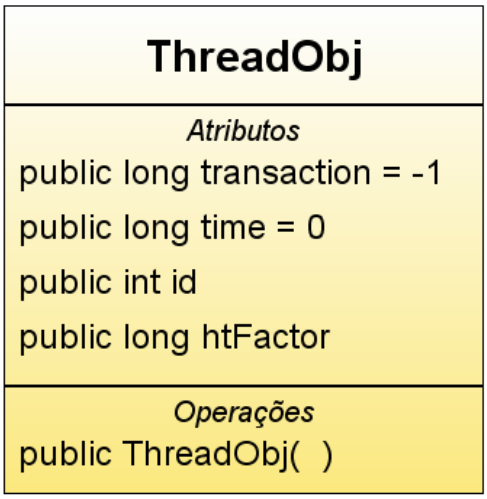

Figura 35: Diagrama de classe de com.cptool.ThreadObj

com.cptool.simObjects.SimObject

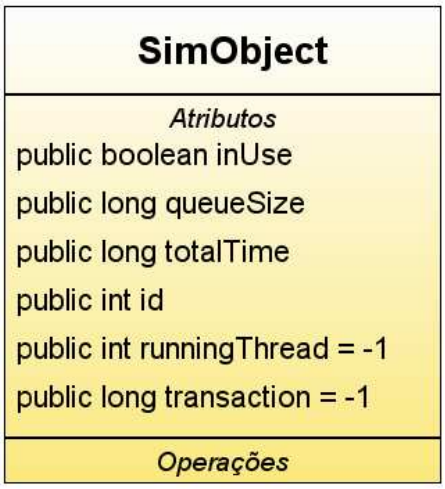

Figura 36: Diagrama de classe de com.cptool.simObjects.SimObject

com.cptool.simObjects.CPU

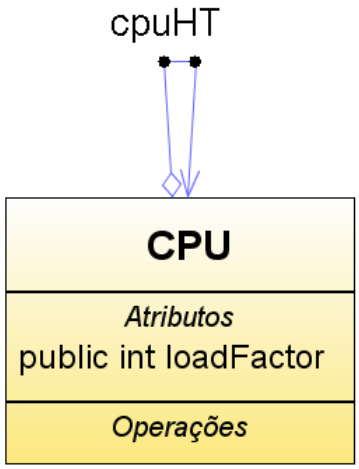

Figura 37: Diagrama de classe de com.cptool.simObjects.CPU

com.cptool.simObjects.Disk

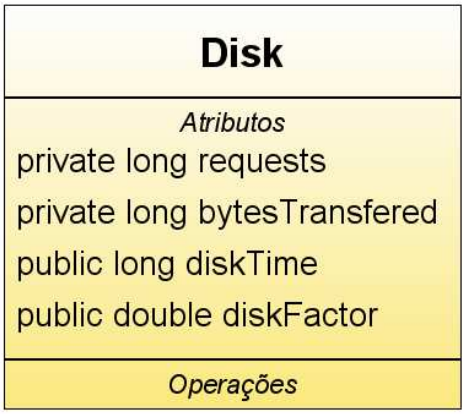

Figura 38: Diagrama de classe de com.cptool.simObjects.Disk

com.cptool.simObjects.Memory

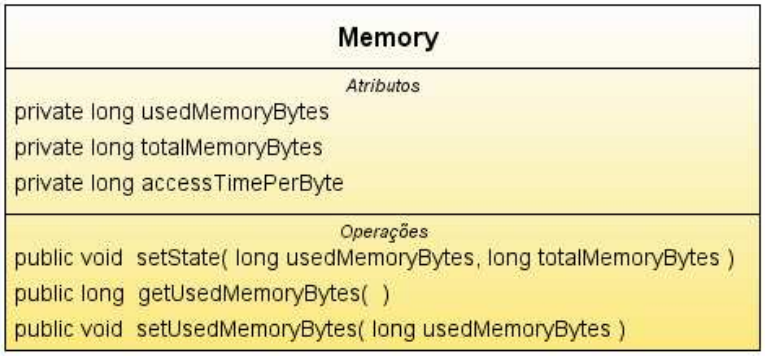

Figura 39: Diagrama de classe de com.cptool.simObjects.Memory

com.cptool.simObjects.Queue

Figura 40: Diagrama de classe de com.cptool.simObjects.Queue

com.cptool.simObjects.Server

Figura 41: Diagrama de classe de com.cptool.simObjects.Server

com.cptool.results.Queue

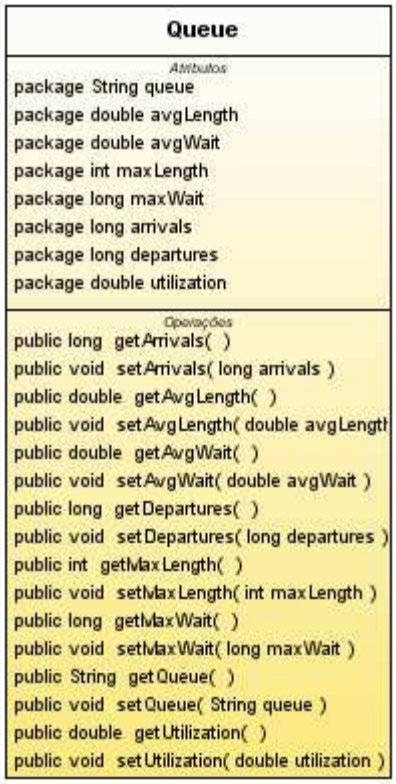

Figura 42: Diagrama de classe de com.cptool.results.Queue

com.cptool.results.Resource

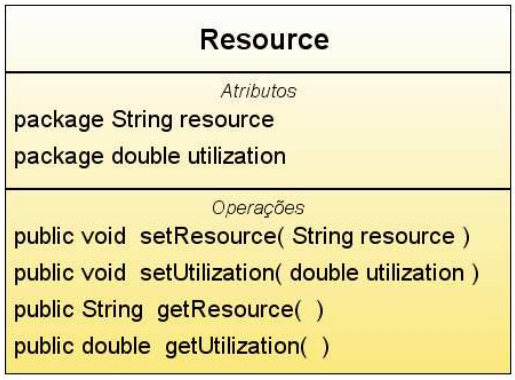

Figura 43: Diagrama de classe de com.cptool.results.Resource

com.cptool.results.Results

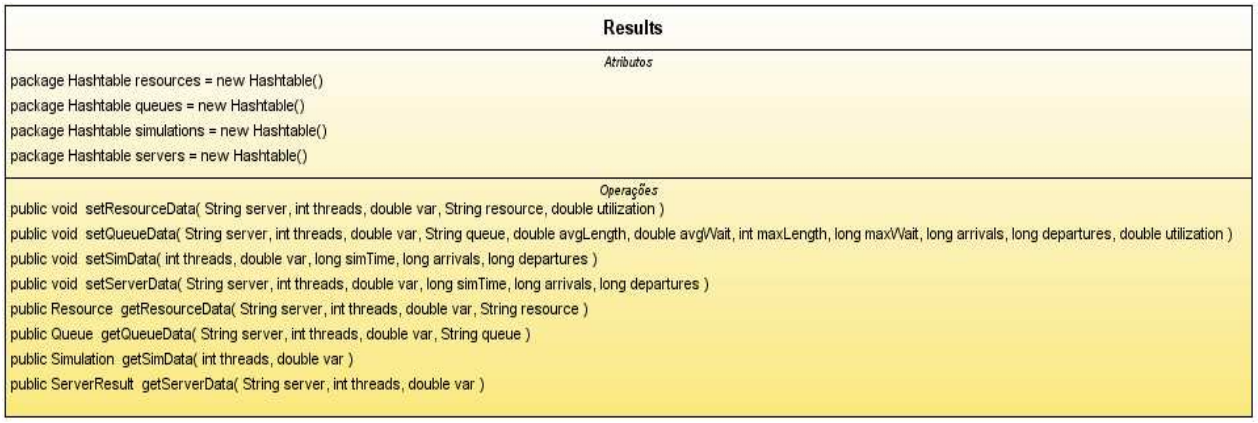

Figura 44: Diagrama de classe de com.cptool.results.Results

com.cptool.results.ServerResult

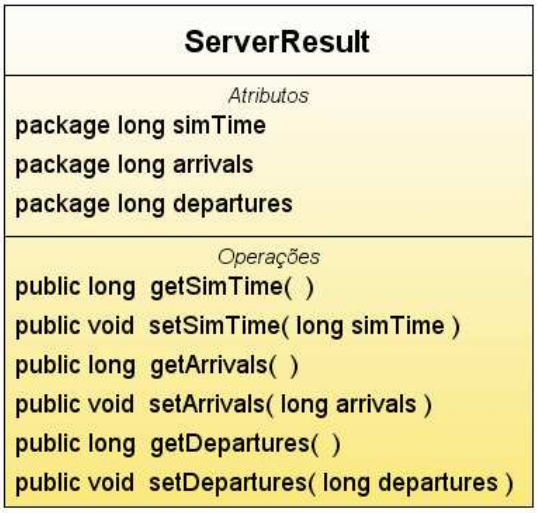

Figura 45: Diagrama de classe de com.cptool.results.ServerResult

com.cptool.results.Simulation

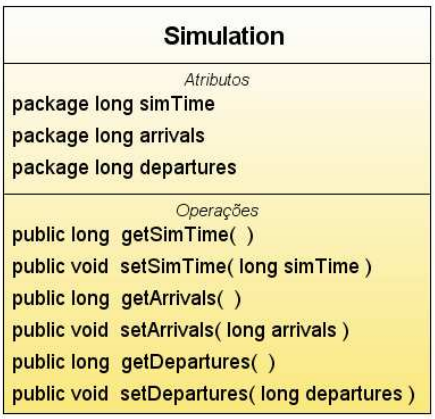

Figura 46: Diagrama de classe de com.cptool.results.Simulation

Apêndice II
Código Fonte do Benchmark

Classe StressTest

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.RandomAccessFile;
import java.io.UnsupportedEncodingException;
import java.util.StringTokenizer;

public class StressTest {



    public static void main(String args[]) {

     if (args.length==5) {

            String testType = args[0];
            int threadNumber = new Integer(args[1]).intValue();
            long loop = new Long(args[2]).longValue();
            long loopMulFactor = new Long(args[3]).longValue();
            int loopStep = new Integer(args[4]).intValue();

            String threadsLabel = "";
            for (int i=0;i<threadNumber;i++) {
              threadsLabel += "Loop "+i+";Time "+i+";";
            }
            System.out.println("Test Type;Threads;Total Loop;Loop Mult.
Factor;Total Time;"+ResourcesStatisticsThread.getLabel()+threadsLabel);
            for (int i=1*loopStep;i<=loop;i=i+loopStep) {
              for (int j=1;j<=threadNumber;j++) {
                if (j<=i) {
                   test(testType,j,i,loopMulFactor);
                } else {
                   break;
                }
              }
            }

        } else {
            System.out.println("Digite java StressTest <testType>
<numMaxThreads> <loopMax> <loopMulFactor> <loopStep>");
        }

    }

    public static void test(String testType, int threadNumber, long
totalloop,long loopMulFactor) {
```

```
        StressThread [] threads = new StressThread[threadNumber];
        long start = System.currentTimeMillis();
        long loop = totalloop/threadNumber;
        long rest = totalloop % threadNumber;


        if (loop>0) {
         long aux = totalloop;
         int i=0;
         while (aux>0) {
            int sum=0;
            if (rest>0) {
               sum=1;
               rest--;
            } else {
               sum=0;
            }

            if (aux>=loop) {
                 threads[i] = new
StressThread(testType,loop+sum,loopMulFactor);
                 threads[i].start();
               aux -= loop+sum;
            } else {
               threads[i] = new StressThread(testType,aux+sum,loopMulFactor);
                 threads[i].start();
               aux =0;
            }
            i++;

         }


        } else {

            int sum=0;

         for (int i=0;i<rest;i++) {
            threads[i] = new StressThread(testType,1,loopMulFactor);
            threads[i].start();

         }

        }



        ResourcesStatisticsThread rst = new ResourcesStatisticsThread(1);
        rst.start();

        for (int i=0;i<threads.length;i++) {
            try {
              if (threads[i] != null)
                 threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
```

```
            }
        }
        long end = System.currentTimeMillis();
        String threadsInfo = "";
        for (int i=0;i<threadNumber;i++) {
          if (threads[i] != null)
             threadsInfo += threads[i].loop +";"+threads[i].time+";";
        }
        String
result=testType+";"+threadNumber+";"+totalloop+";"+loopMulFactor+";"+(end-
start)+";"+rst.getInfo()+threadsInfo;
        System.out.println(substDotComma(result));
        rst.stop();

    }

    public static String substDotComma(String dot) {
     String comma=dot.replaceAll("\\.",",");;

     return comma;
    }

}
```

Classe StressThread

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class StressThread extends Thread{

    String testType;
    long loop;
    long time;
    long loopMulFactor=1;

    public StressThread(String testType,long loop,long loopMulFactor) {
        super();
        this.testType = testType;
        this.loop = loop;
        this.loopMulFactor=loopMulFactor;
    }

    public void run() {
        double a=0;
        File file;
        try {
          long start = System.currentTimeMillis();
             for(int i=0;i<loop*loopMulFactor;i++) {
               if (testType.equals("CPU")) {
                   int count = 0;
                   for (int j=1;j<=500000000;j++){
                    boolean ehPrimo = true;
                    for (int k=2;k < j;k++){
```

```
                        if (j % k == 0){
                            ehPrimo = false;
                        }
                    }
                    if (ehPrimo){
                        count++;
                    }
//                  System.out.println(j);
                    if (count == 1000) break;
            }



              } else if (testType.equals("IO")) {
                file = new File("Test_T"+this.getName());
                BufferedWriter bw = new BufferedWriter(new FileWriter(file));
                bw.write("TESTANDO");
               bw.close();

              } else if (testType.equals("CPUIO")) {
                file = new File("Test_T"+this.getName());
                BufferedWriter bw = new BufferedWriter(new FileWriter(file));
                a = Math.random();
                  a = a*4+3;
                  a = Math.sqrt(a);
                  a = Math.pow(a,3);
                    bw.write("TESTANDO");
                 bw.close();

              }
          }
            long end = System.currentTimeMillis();
          time = end-start;
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }

}
```

Classe ResourcesStatisticsThread

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.util.Hashtable;
import java.util.StringTokenizer;


public class ResourcesStatisticsThread extends Thread{
  private int interval;
```

```
  final static String[][] monitors = new String [][]{
       {"vmstat",
         "Running","1",
         "Blocked","2",
         "SI","7",
         "SO","8",
         "BI","9",
         "BO","10",
         "IN","11",
         "CS","12",
         "CPU_ID","15"}
       };
  double [][] info = new double[monitors.length][];

  public ResourcesStatisticsThread(int interval) {
    this.interval = interval;
  }

  public String getLastInfo(String fileStr,String colStr) {
    String lastLine = getLastLine2(fileStr);
    StringTokenizer st = new StringTokenizer(lastLine," ");
      String token="";
      int col = new Integer(colStr).intValue();
    for (int i=0;i<col;i++) {
      token = st.nextToken();
      }
    return token;


  }

  public String getLastLine(String fileStr) {
    File file=new File(fileStr);
    BufferedReader reader;
    try {

      reader = new BufferedReader(new InputStreamReader(new
BackwardFileInputStream(file),Charset.defaultCharset()));
       String line;
       line = reader.readLine();
       System.out.println("String antes de acerto: "+line);
         // Reverse the order of the characters in the string
         char[] chars = line.toCharArray();
         for (int j = 0, k = chars.length - 1; j < k ; j++, k--)
         {
             char temp = chars[j];
             chars[j] = chars[k];
             chars[k]= temp;

         }
         return new String(chars);
    } catch (IOException e) {
      e.printStackTrace();
    }
    return "";
```

```
  }

  Hashtable lastLines = new Hashtable();
  boolean valid=false;

  private String getLastLine2(String fileStr) {
    String tmp="";
    String linha="";

    if (valid==false) {
      try {

        BufferedReader reader = new BufferedReader(new InputStreamReader(new
FileInputStream(new File(fileStr)),Charset.defaultCharset()));

        while ((tmp = reader.readLine()) != null)
          linha = tmp;

            reader.close();
         }catch (IOException e) {
            //do nothing
         }
         lastLines.put(fileStr, linha);
         valid = true;
    } else {
      linha=(String)lastLines.get(fileStr);

    }
       return linha;
   }


  public void run() {
    int count=0;
    for (int i=0;i<monitors.length;i++) {
      info [i] = new double[(monitors[i].length-1)/2];
    }
    while (true) {
      count++;
      for (int i=0;i<monitors.length;i++) {
        valid=false;
        int infoNumber=0;
        for (int j=1;j<monitors[i].length;j=j+2) {
          if (count>1) {
            info[i][infoNumber] = info[i][infoNumber]*((double)(count-
1)/count) +new Double(getLastInfo(monitors[i][0]+".log",
monitors[i][j+1])).doubleValue()*((double)1/count);
          } else {
            info[i][infoNumber] = new
Double(getLastInfo(monitors[i][0]+".log", monitors[i][j+1])).doubleValue();
          }

          infoNumber++;
        }
      }
```

```
        try {
          Thread.sleep(1000*interval);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }

    public static String getLabel() {
      String ret = "";
      for (int i=0;i<monitors.length;i++) {
        for (int j=1;j<monitors[i].length;j=j+2) {
          ret += monitors[i][j]+";";
        }
      }
      return ret;
    }

    public String getInfo() {
      String ret = "";
      for (int i=0;i<info.length;i++) {
        if (info[i]!=null) {
          for (int j=0;j<info[i].length;j++) {
            ret+=info[i][j]+";";
          }
        }
      }
      return ret;
    }
}
```