Análise de Regras de Balanceamento da Carga em Sistemas Distribuídos: O Efeito do Retardo na Aquisição da Informação de Estado

Marta Emilia Barría Martínez

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Edmundo de Souza e Silva, Ph.D.
(presidente)

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Edil Severiano Tavares Fernandes, Ph.D.

Prof. Luis Felipe M. de Moraes, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

FEVEREIRO 1992

BARRÍA MARTÍNEZ, MARTA EMILIA

Análise de Regras de Balanceamento da Carga em Sistemas Distribuídos: O Efeito do Retardo na Aquisição de Informação de Estado [Rio de Janeiro] 1992
xiv, 140 p., 29.7 cm, (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1992)
TESE – Universidade Federal do Rio de Janeiro, COPPE
1 – Balanceamento da Carga
2 – Análise de Desempenho
3 – Ferramenta para Análise de Desempenho
I. COPPE/UFRJ II. Título(série).

A meu marido Reinaldo
e minhas filhas
Paula, Marisa, Ana Carla e Marcela

Agradecimentos

Ao profesor Edmundo de Souza e Silva, pela orientação e paciência no acompanhamento deste trabalho.

Ao meu marido, por seu apoio nos momentos difíceis e por suas úteis sugestões para a realização desta tese.

A minhas filhas, por terem aprendido a ser independentes, e com isto, terem feito deste período de nossas vidas uma ótima experiência para todos.

A CNPq e a FAPERJ, pelo apoio econômico.

Resumo da Tese apresentada à COPPE como parte dos requisitos necessários para à obtenção do grau de Mestre em Ciências (M. Sc.)

Análise de Regras de Balanceamento da Carga em Sistemas Distribuídos: O Efeito de Retardos na Aquisição de Informação de Estado

Marta Emilia Barría Martínez

Fevereiro de 1992

Orientador: Edmundo A. de Souza e Silva,
Programa: Engenharia de Sistemas e Computação

Foi analisado o problema de determinar a influência do retardo na aquisição de informação de estado nas medidas de desempenho tais como: vazão média do sistema, tempo médio de resposta, número médio de tarefas no sistema e percentual de tempo que uma fila permanece saturada. Para obter estas medidas, implementou-se uma ferramenta computacional, baseada nos modelos matemáticos apresentados em [7]. Foram analisadas diferentes regras de balanceamento de carga, e feitos estudos de sensibilidade das medidas consideradas com relação a parâmetros tais como taxa

de chegada de jobs ao sistema, tamanho do buffer das filas, taxa de serviço dos servidores, probabilidades de atribuição, etc. Este estudo permitiu obter algumas conclusões a respeito das regras mais adequadas para um determinado intervalo entre atualizações da informação de estado, e quais os parâmetros que influem sobre elas.

Abstract of Thesis presented to COPPE as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

Analysis of Load Balancing Policies on Distributed Systems: the Effects of the Delay on State Information Gathering

Marta Emilia Barría Martínez
Fevereiro, 1992

Thesis Supervisor: Edmundo A. de Souza e Silva,
Department: Engenharia de Sistemas e Computação

We studied the influence of the delay in obtaining state information in the performance of load balance policies for distributed system. The performance measures we consider are the throughput, the average reponse time, the average number of jobs in the system, and the percentage of time a given systems remain saturated. In order to obtain such measures we implemented a software tool based on the mathemathical models of [7]. Several load balance policies are analyzes. We also studied the sensitivity of the measures we consider with respect to parameters such as the arrival rate of the jobs to the system, buffer size to the system queues, the capacity of the servers. Among our conclusions, we are able to determine the best policy under given conditions, including the delay to update state information.

Índice

Lista de Figuras

Capítulo I

Introdução

O uso de sistemas distribuídos e redes locais tem aumentado consideravelmente nos últimos anos. Como estes tipos de sistemas constam de múltiplos recursos, a grande vantagem consiste em compartilhá-los entre os diferentes componentes de um sistema de computação. Tarefas submetidas a um sistema de computação podem usar estes recursos (CPUs, discos, etc) dependendo de suas necessidades. Já que o desempenho de tais sistemas não está determinado únicamente pela capacidade de processamento de seus servidores mas, também, pela maneira como seu uso é coordenado, o aprofundamento do estudo de balanceamento da carga de trabalho entre os componentes do sistema constitui um campo de grande interesse de pesquisa.

As propriedades estocásticas das tarefas, tempos de chegadas e tempos de execução, levam ao estabelecimento de filas fazendo com que existam nós que ficam sobrecarregados enquanto outros podem estar vazios ou com pouca carga, o que implica em um desempenho deficiente do sistema. A intuicão sugere que um sistema pode melhorar seu desempenho através do uso balanceado dos recursos disponíveis, evitando sobrecarregar alguns servidores e sub-utilizar outros. Isto é conseguido transferindo tarefas dos nós muito carregados a nós vazios ou pouco carregados. Portanto, o problema de balanceamento de carga em um sistema distribuído pode ser formulado como a distribução das tarefas no sistema de forma tal a otimizar o desempenho do sistema com relação a alguma medida, por exemplo o tempo médio de resposta.

O estudo de regras efetivas de alocação dos recursos em sistemas de multiservidores distribuídos tem sido intensamente pesquisado na última década, nas áreas de computação e comunicação. Nestas áreas, problemas de roteamento e o problema de balanceamento da carga, onde os recursos estão relacionados ao processo da informação e ao armazenamento, estão intimamente relacionados. O problema de roteamento é similar ao de balanceamento da carga, sendo que os recursos a serem compartilhados são os canais de comunicação e o objetivo final é encontrar os caminhos ótimos para os pacotes, de forma que alguma medida seja otimizada.

As decisões de balanceamento de carga podem ser feitas em forma

distribuída [14] ou *centralizada* [16]. Na forma centralizada, toda a informação do estado do sistema é coletada em um só ponto e a decisão também é feita em um ponto. Por exemplo, existe um gerente que é o centralizador da informação do sistema e é ele quem toma as decisões. Uma desvantagem de um gerente centralizado é o 'overhead' de coletar e processar a informação de estado do sistema. Como todo sistema centralizado, uma grande desvantagem é a existência de um ponto vulnerável: o gerente.

Na forma distribuída, cada nó possui sua própria informação de estado, e a decisão de balanceamento não se reduz a um nó. Dependendo desta informação cada nó decide se envia ou não sua tarefa para processamento remoto. Para isto, pergunta pelo estado de alguns dos processadores do sistema, e envia a tarefa para aquele que satisfaz o critério implementado na regra de balanceamento (por exemplo: fila menor, tempo de resposta menor, etc.) Com este tipo de deci,são se evita a coleta de informação em um só ponto.

A decisão de como balancear a carga de trabalho do sistema entre os nós pode ser *estática* ou *dinâmica*. O balanceamento estático [16,19,8] distribui carga de acordo com alguma regra previamente estabelecida, por exemplo, em forma probabilística, na qual as tarefas são atribuídas aos nós com uma probabilidade pré-calculada para otimizar alguma medida (vazão , tempo médio de resposta, etc); O balanceamento de carga estático não leva em conta o estado atual do sistema, mas o comportamento médio do sistema. Por outro lado, uma decisão dinâmica leva em conta o estado do sistema no momento da decisão. Elas reagem, rapidamente, às mudanças de estado do sistema. As regras dinâmicas são muito mais difíceis de analisar que as regras estáticas as quais não se adaptam por si mesmas às variações do sistema.

Uma regra de balanceamento pode ser iniciada na *fonte* ou no *servidor* [20], dependendo de qual nó inicia a transferência da tarefa. A transferência é iniciada na fonte por um nó sobrecarregado, na chegada de um novo tarefa , baseada em alguma estrategia de balanceamento. Por outro lado, a transferência é iniciada no servidor quando um nó pouco carregado consulta cada uma das possíveis fontes para que lhe enviem um tarefa. Os algoritmos de balanceamento de carga podem ser divididos em vários níveis dependendo da quantidade de informação que requeiram. Em [20], depois de analisar e comparar vários deste algoritmos, os autores concluíram que um algoritmo que coleta mais informação, geralmente, produz melhores resultados; e que, transferências iniciadas na fonte tem melhores resultados que as iniciadas no servidor, para o mesmo nível de informação e custo de transferência não significativo.

Uma regra de balanceamento é composta por uma *regra de transferência* [10] que determina quando um nó deve transferir uma tarefa , e por uma *regra de localização* que determina para onde uma tarefa deve ser transferida.

Neste trabalho se considera um problema de balanceamento da carga para o modelo de multiservidores mostrado na figura I.1. Este tipo de modelo para sistemas de multiprocessadores é muito usado na literatura para problemas de

roteamento e balanceamento da carga [5,19,16,4,17].

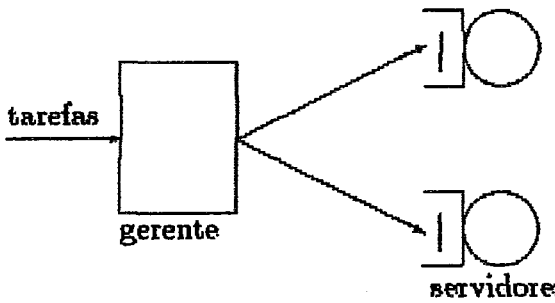

Figura I.1: Um modelo para balanceamento da carga

Considera-se também o modelo distribuído em que cada servidor se encarrega de transferir ou não as tarefas a ele submetidas.

No modelo usado neste trabalho, cada servidor possui sua própria fila de capacidade limitada. Os servidores são caracterizados por sua taxa de serviço, as quais podem ser iguais para todos os servidores (sistema homogêneo) ou diferentes para cada um dos processadores (sistema heterogêneo). A arquitetura do sistema consiste em um servidor central ('gerente'), quem é o encarregado de atribuir as tarefas que chegam ao sistema aos diferentes servidores. O gerente faz esta atribuição dependendo do estado do sistema que ele possui. A atualização da informação de estado do gerente é feita a cada T unidades de tempo. Isto faz com que o gerente trabalhe com informação desatualizada e as decisões tomadas podem não rerfletir o estado atual do sistema, podendo gerar desempenho do mesmo. O objetivo principal deste trabalho é analisar os efeitos desses retardos sobre o desempenho de diferentes algoritmos de balanceamento de carga.

Existe uma ferramenta implementada em [9], que utiliza programação orientada a objeto para a descrição e geração da Cadeia de Markov correspondente a um sistema descrito em alto nível. Esta ferramenta será usada para a descrição do sistema sob análise e para a geração da cadeia de Markov que representa a evolução do sistema no período de tempo entre atualizações do estado do gerente. A cadeia será gerada em forma simbólica, isto é, as probabilidades de atribuição serão paramétricas, permitindo com isto que sejam computadas posteriormente, pela ferramenta criada neste trabalho, e dependerão da regra de balanceamento de carga usada.

Uma vez que as atualizações de estado do sistema são feitas a intervalos de tempo fixos de tamanho T, precisa-se construir uma cadeia de Markov nos pontos embutidos kT, $k = 1, 2, \ldots$, para poder analisar o sistema. Como a complexidade de analisar o modelo matemático é grande, é necessário poder fazer a análise em forma automática. Para isto, foi implementada, neste trabalho, uma ferramenta computacional que permite obter para um determinado sistema, a partir da cadeia de Markov genérica (gerada pela ferramenta de [9]) e da regra de balanceamento específica, a cadeia de Markov nos pontos embutidos e diferentes medida

de desempenho, como vazão média, tempo de resposta médio, etc.

A solução analítica baseia-se nos resultados prévios obtidos para modelos de falha/reparo de [7], mas adaptados ao caso de modelos de balanceamento da carga. O método de solução precisa da avaliação de medidas transientes e estacionárias. Como parte da análise é usada a técnica de randomização [15,11], e a obtenção de medidas de desempenho para cadeias de Markov com recompensas.

A contribuição do presente trabalho consiste na adaptação do modelo apresentado em [7], para o problema de balanceamento da carga. Além do mais, foi construída uma ferramenta complexa para permitir a descrição e análise de diferentes modelos em forma automática.

No que diz respeito à implementação, vale salientar alguns itens:

- A ferramenta implementada em [9] não era suficientemente flexível para a avaliação do modelo, usando-se apenas para gerar uma cadeia de Markov genérica. Houve a necessidade de implementação de várias regras de balanceamento, em Prolog, bem como outros programas (nesta mesma linguagem) necessários para a obtenção de dados de entrada para a avaliação do modelo e suas medidas.

- Foi implementado um programa, em Pascal, que avalia a cadeia de Markov genérica gerando, através de um 'parser', a cadeia final.

- Implementou-se uma ferramenta que, tendo como uma das entradas, a cadeia de Markov numérica, randomiza-a, para permitir o cálculo da matriz de transição nos pontos embutidos.

  Esta ferramenta permite também calcular as medidas de desempenho do sistema.

  A ferramenta calcula as medidas de forma simultânea, para diferentes valores de T.

- Com esta ferramenta foi possível realizar, para diferentes regras de balanceamento da carga, diversas análises de sensibilidade a parâmetros tais como: tamanho do intervalo T, taxa de chegadas de tarefas ao sistema, tamanho do 'buffer', etc.

- A partir destes resultados foi possível fazer uma melhor avaliação do comportamento da regra de balanceamento e obter as conclusões correspondentes.

O resto deste trabalho está organizado da seginte forma. No capítulo II é revisado material prévio necessário para a análise e obtenção de medidas de desempenho. Também, é descrito o modelo e são calculadas algumas medidas de desempenho. No Capítulo III são revisados trabalhos anteriores publicados na literatura sobre balanceamento de carga. O Capítulo IV descreve a metodologia orientada a objeto, usada na implementação do modelo para balanceamento da carga. No Capítulo V é descrita a ferramenta computacional desenvolvida aqui para a análise

de diferentes regras de balanceamento de carga. O Capítulo VI contém os resultados numéricos obtidos a partit da análise das diferentes regras de balanceamento da carga. O Capítulo VII apresenta as conclusões obtidas.

No curso do texto optou-se por traduzir da grafia inglesa, termos tais como 'throughput' (vazão), 'job' (tarefa), 'scheduler' (gerente).

Capítulo II

O modelo e suas medidas

II.1 Material Prévio

A técnica de Randomização

Seja $X = \{ X(t) : t \geq 0 \}$ um processo de Markov de tempo contínuo com espaço de estado finito $S = \{a_i : i = 1 \ldots E\}$, e matriz geradora $\mathbf{Q} = \{ q_{ij} \}$.

Seja N(t) um processo de Poisson homogêneo, independente de X(t), com espaço de estado finito e, taxa Λ, tal que $\Lambda \geq max_i \{ \| q_{ii} \| \}$.

Seja **P** uma matriz de transição definida como **P**= **Q** Λ + **I**, sendo **I** a matriz identidade.

Seja $\mathbf{Z}_{N(t)}$ uma cadeia de Markov de tempo discreto com espaço de estado S e matriz de transição **P**. As transições de **Z** acontecem nos instantes dos eventos de N(t) (a cadeia de Markov **Z** está subordinada ao processo de Markov N(t))

Então, de [11,15] tem-se que: $X(t) = Z_{N(t)}$.

II.2 Descrição do modelo

Considera-se um sistema distribuído genérico que consta de vários processadores, conectados através de uma rede de comunicação. Para facilitar a discussão supõe-se que existe um gerente, que é o encarregado de distribuir as tarefas que chegam ao sistema para os processadores. Esta distribuição é feita de acordo com a regra de balanceamento implementada nele, a qual dependerá do estado do sistema.

O modelo inicial consiste de N servidores cada um com sua própria fila de espera de capacidade finita e um gerente, como se mostra na figura II.1

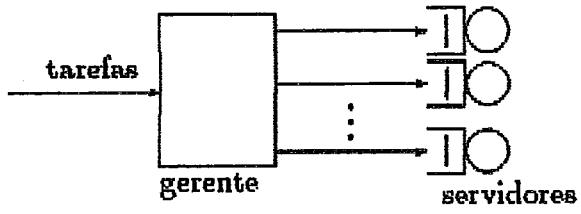

Figura II.1: O modelo

Este gerente decide, para cada tarefa que chega ao sistema, qual processador será utilizado. Esta decisão é feita dependendo da regra de balanceamento escolhida e do estado do sistema, que é atualizado a intervalos de tempo fixos de tamanho T. Isto implica que a regra de balanceamento é dinâmica, já que leva em consideração as mudanças no estado do sistema. Com isto o sistema sendo estudado e' modelado de uma forma mais próxima à real.

Supõe-se que os tarefas que chegam ao sistema o fazem de acordo com um processo de Poisson de taxa λ. Uma vez alocada a um servidor particular, um tarefa não pode ser re-atribuída e deve ser processada completamente pelo servidor.

O tempo de serviço é suposto ser exponencialmente distribuído com média μ.

O gerente atribui uma tarefa a um servidor dependendo de uma regra de balanceamento particular e do estado do sistema. A coleta de informação sobre o estado do sistema é feita a intervalos de tempo fixos de T unidades de tempo. Durante o tempo entre atualizações, o sistema evolui, isto é, tarefas chegam ao sistema e tarefas são processadas, assim ao final do intervalo o estado do sistema pode ser qualquer. Se este tempo entre atualizações de estado for muito grande, o gerente tomará decisões a partir de informações envelhecidas, influindo negativamente no desempenho do sistema.

II.3 Método de solução

A solução analítica baseia-se nos resultados prévios obtidos para modelos de falha/reparo de [7], principalmente, nos resultados obtidos para o Modelo 2 daquele artigo.

Analisa-se o sistema usando a seguinte metodologia. Asume-se que o gerente atualiza a informação de estado do sistema cada certo tempo T, como indicado na figura II.2. Como o comportamento do sistema durante os intervalos entre atualizações depende somente do estado do sistema no início deste intervalo, pode-se construir uma cadeia de Markov embutida nos pontos em que o gerente faz

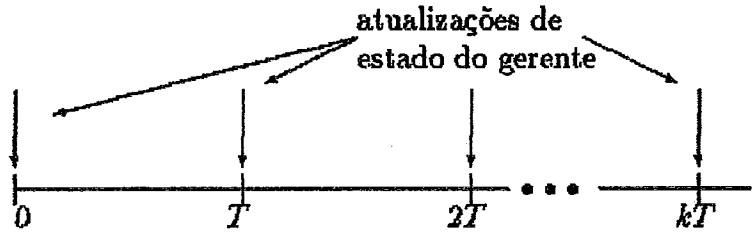

Figura II.2: Atualizações de estado do gerente

as atualizações de informação de estado do sistema.

Estas atualizações de estado do gerente formam uma seqüência de pontos no tempo os quais identificam uma cadeia de Markov embutida $Y = \{Y_k : k = 0, 1, \ldots\}$ nos pontos embutidos no tempo $\{\tau_k : k = 0, 1, \ldots\}$. Os pontos embutidos correspondem à atualização de informação de estado do gerente. Durante o intervalo de tempo entre atualizações, o sistema evolui como um processo de Markov X.

O método de solução determina primeiro a matriz de transições **D** da Cadeia de Markov embutida (Y). A i-ésima linha da matriz **D** ($\vec{d_i}$) representa a probabilidade de transição desde o estado i a todos os outros estados no intervalo T.

Cada linha da matriz **D** ($\vec{d_i}$) é calculada mediante randomização.

O vetor $\vec{d_i} = \langle d_{i1}, d_{i2}, \ldots d_{iE} \rangle$, pode ser obtido de:

$$\vec{d_i} = \sum_{n=0}^{\infty} e^{-\Lambda T} \frac{(\Lambda T)^n}{n!} \vec{\pi}(n) \qquad \text{(II.1)}$$

onde o vetor de probabilidades inicial é $\pi(0) = \vec{e_i}$ e $\vec{e_i} = \langle 0, \ldots, 1 \ldots, 0 \rangle$, o qual possue um 1 na i-ésima coluna, e $\pi(n) = \pi(n-1)\mathbf{P}$ é o vetor de probabilidades na n-ésima transição;

O vetor de probabilidades em estado estacionário da cadeia de Markov embutida é $\vec{\beta} = \langle \beta_1, \beta_2, \ldots, \beta_E \rangle$ e é obtido resolvendo $\vec{\beta} = \vec{\beta}\mathbf{D}$, onde β_i é o vetor de probabilidades em estado estacionário para o estado i.

II.4 Medidas de desempenho

II.4.1 Cálculo da Vazão Média

Esta medida é calculada de forma similar ao tempo médio de visitas não programadas do Modelo 2 de [7].

Considera-se o número de tarefas processadas no intervalo (0,T). Dado que o estado inicial no tempo $\tau = 0$ é a_i, seja $Th_i(t)$ uma variável aleatoria que representa o número de tarefas processadas no intervalo de tempo (0,T). Deseja-se obter, o número médio esperado de tarefas processadas neste intervalo, $E[Th_i(T)]$, dado que o estado inicial é e_i. Para calcular esta medida usa-se randomização. Se um tarefa foi processada na cadeia de Markov randomizada, a transição de $r \rightarrow s$ é marcada [6].

Então:

$$E[Th_i(T)] = \sum_{n=0}^{\infty} e^{-\Lambda T} \frac{(\Lambda T)^n}{n!} \sum_{l=1}^{n} P(\text{l-ésima trans. seja marcada}) \qquad \text{(II.2)}$$

É preciso calcular a probabilidade de que uma transição específica esteja marcada. Define-se para isto:

$$t_{rs} \stackrel{def}{=} \begin{cases} 1 & \text{se } r \rightarrow s \text{ é marcada} \\ 0 & \text{outros casos} \end{cases}$$

onde t_{rs} é obtido a partir da matriz geradora **Q** do problema particular sendo estudado.

A matriz $\mathbf{Tr} = [p_{rs}t_{rs}]$ dá as probabilidades marcadas de um passo.

Define-se também o vetor $\Phi(l) = \pi(l-1)\ \mathbf{Tr}$; $\quad l = 1, 2, \ldots$. Então, a probabilidade que a l-ésima transição seja marcada é a soma das componentes de $\Phi(l)$ ($\| \Phi(l) \|$), isto é, a soma das componentes do vetor Φ na l-ésima transição.

Portanto,

$$E[Th_i(t)] = \sum_{n=0}^{\infty} e^{-\Lambda T} \frac{(\Lambda T)^n}{n!} \sum_{l=1}^{n} \| \Phi(l) \|$$

Para obter a vazão média em estado estacionário (TH) no intervalo (0,T), é necessário descondicionar para cada estado inicial pelas probabilidades em estado estacionário (β_i) e dividir pelo tamanho do intervalo (T), então :

$$TH = \sum_{i=1}^{E} \frac{\beta_i E[TH_i]}{T} \qquad \text{(II.3)}$$

II.4.2 Número Médio de Tarefas no Sistema

Considere-se o número de tarefas no intervalo (0,T). Dado que o estado inicial no tempo $\tau = 0$ é a_i, seja N_i uma variável aleatória que representa o número tarefas no intervalo de tempo (0,T). Deseja-se calcular $E[N_i(t)]$ o número médio esperado de tarefas processadas neste intervalo, dado que o estado inicial é a_i. Define-se η_j a população total do estado a_j, isto é, a soma do número total de tarefas representada

pelo estado a_j; e seja $\pi(k)_j$ a j-ésima componente do vetor de probabilidades na transição 'k'.

Então, para cada estado inicial, o número médio de tarefas no sistema é dado por:

$$E[N_i(t)/\text{n mudanças de estado em (0,T),}\ \text{e está na k-ésima mudança de estado}] = \sum_{j=1}^{E} \eta_j \pi(k)_j$$

Descondicionando para todas as transições que ocorrem no intervalo (0,T):

$$E[N_i(t)/\text{n trans.}] = \sum_{k=0}^{n} \left(\sum_{j=1}^{E} \eta_j \pi(k)_j \right) \frac{T}{n+1}$$

Manipulando os somatórios, tem-se:

$$E[N_i(t)/\text{n transc.}] = \frac{T}{n+1} \sum_{j=1}^{E} \eta_j \sum_{l=0}^{n} \pi(l)_j$$

Finalmente, descondicionando pela Poisson, que fornece o número de transições que ocorrem em um intervalo (0,T), tem-se que:

$$E[N_i(t)] = \sum_{n=0}^{\infty} e^{-\Lambda T} \frac{(\Lambda T)^n}{(n+1)!} \sum_{j=1}^{E} \eta_j \sum_{l=0}^{n} \pi(l)_j \qquad \text{(II.4)}$$

Seja N o número médio de tarefas no sistema em estado estacionário no intervalo (0,T). Então, descondicionando pelos vetores em estado estacionário da Cadeia de Markov Y, e normalizando pelo tamanho do intervalo, tem-se:

$$N = \sum_{i=1}^{E} \frac{\beta_i E[N_i]}{T} \qquad \text{(II.5)}$$

II.4.3 Tempo de resposta no sistema

Pela Lei de Little [15], calcula-se o tempo médio de resposta no sistema (TR) como:

$$TR = \frac{N}{TH} \qquad \text{(II.6)}$$

Usa-se em vez da taxa de chegada de tarefas ao sistema, a TH (vazão) devido a que o sistema tem filas com capacidade limitada.

II.4.4 Tempo médio que uma das filas permanece saturada

Seja T_{fjch_i}, a variável aleatória que indica o tempo que o servidor j permanece com sua fila saturada durante o intervalo $(0, T)$, dado que o estado inicial é a_i. Portanto, o valor esperado para esta v.a. está dado por:

$$E[T_{fjch_i}] = \frac{1}{\Lambda} \sum_{n=0}^{\infty} e^{-\Lambda T} \frac{(\Lambda T)^{n+1}}{(n+1)!} \sum_{k=0}^{n} \| \pi(k) \|_j \qquad \text{(II.7)}$$

Onde $\| \pi(k) \|_j$ representa a soma das componentes do vetor de probabilidades π, que tem a fila j saturada, na k-ésima transição.

Seja T_{fjch} o valor médio estacionário no intervalo $(0, T)$. Então a medida estacionária e' dada pela seguinte equação:

$$T_{fjch} = \frac{\sum_i \beta_i E[T_{fjch_i}]}{T} \qquad \text{(II.8)}$$

O capítulo seguinte dará uma visão do paradigma da programação orientada a objeto e da ferramenta implementada em [9], a qual será usada para a geração da cadeia de Markov, que representa a evolução do sistema entre atualizações de estado do gerente.

Capítulo III

Trabalhos anteriores em balanceamento da carga

Em [5] são analisadas três estratégias de balanceamento de carga baseadas na informação atual do estado de um sistema heterogêneo de multiprocessadores simples.

O modelo usa um gerente central o qual roteia a tarefa que chega a algum dos processadores, de acordo com alguma estratégia de roteamento (determinística e não determinística). O roteamento das tarefas é projetado para reduzir o tempo medio de resposta ao balancear a carga entre os processadores.

Modelos de redes de filas são usados para representar o sistema.

O roteamento não determinístico usa um sistema de multiprocessadores heterogêneos (isto é, cada um dos processadores do sistema pode ter diferentes taxas de serviço).

O sistema é modelado como uma fila aberta com m processadores independentes. Cada processador se assume completamente caracterizado pela sua taxa média de serviço μ_i. As tarefas chegam ao sistema com taxa média λ. Nesta regra, um tarefa chegando ao sistema é roteada ao processador i com probabilidade p_i, onde a decisão de roteamento baseia-se no resultado de tiradas individuais. Cada processador tem uma disciplina de atendimento FCFS. O comportamento do sistema é descrito por um processo markoviano de tempo contínuo e de espaço de estado discreto. Se p_i é fixo, o sistema pode ser decomposto por m filas M/M/1 independentes, onde o sistema i tem taxa de chegada λp_i e taxa de serviço μ_i

Para as regras de roteamento determinísticas o sistema é modelado como um sistema multiprocessador simples com um gerente, o qual distribui aos processadores as tarefas que chegam ao sistema, de acordo com um critério previamente estabelecido, usando para isto informação de estado do sistema. A estratégia de alocar uma tarefa é determinística, já que uma tarefa que chega é sempre roteada para algum servidor.

A primeira regra determinística é a Regra de Tempo Mínimo de Resposta (R). Ela roteia a tarefa que chega ao sistema a algum servidor que dará o menor tempo esperado de resposta de uma tarefa. O gerente transferirá a tarefa que chega, à fila i se $R_i(n_i + 1) = min_l\{R_l(n_l + 1\}$ onde $R_l(n_l) = n_l/\mu_l$, n_l é o comprimento da fila l e $/\mu_l$ é a taxa de serviço da fila l.

A segunda regra, Regra de Tempo Mínimo no Sistema (T) visa minimizar o tempo no sistema. Para isto, o gerente enviará uma tarefa chegando ao sistema à fila que minimize o tempo esperado, para completar serviço de todas as tarefas no sistema, assumindo que não existem chegadas posteriores. A análise demonstra que, embora esta regra melhore o desempenho do sistema, não é uma regra de roteamento ótima.

Na terceira regra, Regra de Vaz ao Maxima (TP), a tarefa chegando ao sistema será roteada à fila que dará a vaz ao esperada máxima para o sistema, durante o seguinte período entre chegadas. A vaz ao do sistema é a soma das vaz oes individuais de cada um dos processadores no sistema.

A solução é encontrada para um sistema com dois processadores heterogêneos. É usado, para tanto, técnicas numéricas recursivas, que usam equações de equilíbrio global.

Os resultados obtidos permitem que as seguintes conclusões sejam tiradas:

- a regra de TP dá o menor tempo para uma taxa total de chegadas (λ) escolhida.
- As regras T e R se degradam mais, quando λ cresce, que a regra TP. Isto se deve a que estas regras tendem a entregar tarefas a processadores mais rápidos, enquanto deixam os processadores mais lentos ociosos. Esta estratégia é eficaz para cargas baixas, mas para cargas altas a regra de TP é considerada melhor, pois a informação global do gerente está disponível.

O artigo [13] fornece três diferentes algorítmos de balanceamento de carga para um sistema distribuído com um meio de comunicação 'broadcast'. Neste sistema, não existe gerente e os algorítmos de balanceamento estão distribuídos entre os servidores.

O modelo descreve um sistema distribuído de N servidores. Cada nó é idêntico e está conectado através de um canal de comunicação. Cada nó possui um processador, um processador de comunicação e uma fila de espera.

O canal é um meio passivo 'broadcast' com um método de acesso CSMA/CD. O acesso ao canal e a transmissão de mensagens são controlados pelo processador de comunicação de acordo com um protocolo Ethernet.

As tarefas chegam, independentemente, a cada nó e entram na fila. A disciplina de atendimento em todos os nós é FIFO. O processo de chegada em cada nó assume-se ser Poisson com média λ. Supõe-se que o tempo de serviço é

distribuído exponencialmente com média μ. As tarefas deixam o sistema depois de serem atendidas. Supõe-se, também, que o sistema opera em condições de estado estacionário. O estado do sistema é descrito pela N-tupla do número de tarefas em cada nó que espera por serviço.

O propósito do canal é transferir tarefas de um nó a outro de forma tal a melhorar o tempo esperado de resposta. O fluxo no canal é controlado por um algorítmo distribuído de balanceamento de carga. Os autores apresentam três algorítmos de balanceamento, para um sistema distribuído 'broasdcast'.

No algorítmo de 'estado broadcast' (STB), cada vez que o estado do nó muda, o nó envia seu estado aos outros nós do sistema. Então, todos os nós mantém uma informação do estado do sistema aproximadamente atualizada (há diferenças devido ao retardo de transmissão de mensagens). Os nós tomam a decisão de transferir uma tarefa em forma distribuída, de acordo com critérios de transferência que devem ser cumpridos.

No algorítmo de 'broadcast ocioso' (BID) um nó envia uma mensagem de ociosidade, quando entra neste estado, aos outros nós do sistema, que ativarão seu sistema de transferência de tarefas, se for o caso. Os nós com mais carga têm maior chance de transferir uma tarefa a um nó ocioso.

No algorítmo 'polling quando ocioso' (PID), o nó começa um 'polling' quando entra em um estado ocioso, perguntando a um conjunto de nós se desejam transferir tarefas para ele.

Foi usada simulação como um meio de predizer o comportamento do sistema. Para esta simulação, foram usados três modelos, um para cada algorítmo. Em cada um dos modelos assumiu-se que os únicos retardos que existem, são devidos a retardos de comunicação. A comunicação é feita de acordo com um protocolo Ethernet e o efeito das colisões está incluído.

Os resultados da simulação mostram que cada um dos algorítmos é melhor sob certas condições. Foram obtidas três conclusões principais:

1. Uma multiplicidade de recursos muito grande não fornece, necessariamente, um bom resultado de tempo 'turnaround'. Cada algorítmo alcança um ponto onde um incremento no número de servidores faz decrescer o desempenho do sistema.

2. O processo de balanceamento tem uma alta atividade de comunicação.

3. A seleção das leis de controle e da política de informação deve depender dos retardos de transmissão esperados do sistema balanceado, devido ao 'dilema de transmissão' (se por um lado a mensagens transmitida melhoram a capacidade do algorítmo, por outro elevam o tempo médio de espera das mensagens, em conseqüência da maior utilização do canal).

Ni and Hwang [16] desenvolveram uma técnica de balanceamento de

carga probabilística ótima que melhora o tempo médio de resposta de uma tarefa. O sistema considerado é um sistema multiprocessador levemente acoplado, constituído de múltiplos processadores independentes. Cada processador possui suas próprias características (sist. heterogêneo). Os processadores têm sua própria memória local. Cada processador mantém sua própria fila de tarefas prontos para rodar. Neste sistema, um servidor central ('gerente') é o responsável pela alocação das tarefas entre os vários processadores. Uma tarefa que chega é atribuída a um dos processadores de acordo com a política de escalonamento e com as características das tarefas. Dependendo da capacidade do processador e da posibilidade de atribuição, as tarefas são agrupadas em classes diferentes. Diferentes classes de tarefas são atribuidas a diferentes grupos de processadores.

O modelo do sistema é descrito como um sistema multiprocessador com n processadores e m diferentes classes de tarefas. Cada processador é modelado por uma fila M/M/1. Conseqüentemente, o sistema é modelado por n independentes M/M/1. A i-ésima classe que chega tem chegada Poisson com média λ_i. O j-ésimo processador tem uma taxa de serviço distribuída exponencialmente com média μ_j. Cada um dos processadores tem uma regra de atendimento FCFS. O gerente tem uma regra probabilística que não depende do estado do sistema para a atribuição dos tarefas aos processadores. Uma tarefa de tipo i é atribuída para rodar em um processador (do conjunto de processadores atribuíveis) com probabilidade P_i.

A operação do sistema é dada por uma matriz de atribuição [A], que é uma matriz de $m \times n$, onde $a_{ij} = 1$ indica que a tarefa da i-ésima classe pode ser executado no j-ésimo processador ($a_{ij} = 0$ em outro caso) e por uma matriz de scheduling $S = [s_{ij}]$, onde s_{ij} é a probabilidade da tarefa da i-ésima classe ser atribuída ao j-ésimo processador; então s_{ij}=0 se $a_{ij} = 0$ e o somatorio de s_{ij} para um determinado processador deve ser 1.

Uma vez que a matriz de atribuição é determinada, o modelo pode ser decomposto em n filas independentes, onde a k-ésima fila tem uma taxa de chegada λ'_k, que é a soma das chegadas das diferentes clase atribuidas ao processador, e uma taxa de serviço μ_k.

O objetivo é encontrar uma matriz de alocação ótima que minimize o tempo médio de resposta das tarefas. Isto é formulado como um problema de programação não linear sujeito a restrições lineares. Sendo este problema bastante complexo, os autores adotaram, primeramente, uma abordagem simplificada de resolução, trabalhando com um sistema de duas classes de tarefas, onde a segunda classe tem taxa preatribuida. Para este sistema eles encontraram as probabilidades de roteamento ótimas. Para encontrar a solução final os autores atribuem taxas a alguns tarefas. Todas as classes que devem rodar em um processador são combinadas em uma classe com taxa pré-atribuída. As outras tarefas, de outras classes, podem rodar no conjunto de processadores restante.

A solução é encontrada através de um algorítmo recursivo estruturado, o qual gera a matriz de alocação. Este algorítmo fornece um tempo de resposta único e mínimo para todas as possíveis soluções de S. O algorítmo encontra ao me-

nos uma das possíveis soluções de S.

Em [20] são apresentados dez algorítmos de balanceamento de carga e sua avaliação de desempenho, no intuito de ajudar a escolha de uma estratégia de balanceamento.

O sistema considerado é um sistema computacional localmente distribuído. As tarefas entram no sistema via nós chamados fontes e são processados por nós chamados servidores. Um processador físico pode ser tanto uma fonte como um servidor.

As suposições feitas para o modelo são: as tarefas são individualmente executáveis, são logicamente independentes umas das outras e podem ser processadas por qualquer servidor. Uma tarefa, uma vez atribuída a um servidor é processada nele, não podendo ser reatribuída. Assume-se também, por simplicidade, que todos os servidores têm a mesma taxa de serviço. A disciplina de atendimento na fonte e no servidor é FCFS.

Uma estratégia de balanceamento pode ser iniciada *na fonte* quando a mesma determina onde rotear uma tarefa, ou *no servidor* quando o mesmo determina que tarefas em diferentes fontes ele pode servir. Usando-se o algorítmo de iniciativa na fonte ocorre uma tendência de formação de fila no servidor e a decisão de roteamento é tomada ao chegar uma tarefa. Com o algorítmo de iniciativa no servidor, a tendência de formação de fila ocorre na fonte e a decisão de roteamento é tomada ao partir uma tarefa.

Os dez algorítmos apresentados são clasificados quanto a 'dependência de informação'. Isto é, depende da quantidade de informação usada pelo nó para tomar a decisão de balanceamento e do tipo da estratégia de balanceamento (iniciada na fonte ou no servidor).

Estes algorítmos são avaliados para diferentes distribuições de chegadas (Poisson, Determinística, Batched) e serviço (exponencial, determinístico, hiperhexponencial, batched) em um sistema que possui N fontes e K servidores. Dependendo da combinação de suposições, os algorítmos são avaliados em forma exata ou por simulação.

Para homogeneizar a comparação da avaliação os autores desenvolveram uma métrica chamada 'fator de qualidade de compartilhamento da carga' ou 'fator-Q' que captura alguns aspectos de primeiro ordem do desempenho de regras de balanceamento de carga.

O 'fator-Q' ($Q_A(\rho)$) é definido como :

$$Q_A(\rho) = \frac{\text{tempo médio de resposta sobre todos as tarefas son FCFS}}{\frac{1}{K}\sup_{\sum_{i=1}^{N}\lambda_i=\rho} \max_i \{ \text{ tempo médio de resposta na i- ésima fonte sob o algorítmo A } \}}$$

onde o tempo de resposta de uma tarefa é definido como sendo o intervalo de tempo desde quando a tarefa chega ao sistema até sua partida do sistema. N é o número

de fontes, K é o número de servidores, μ é a taxa média de serviço, λ_i é a taxa de chegada no servidor i, e ρ é a utilização agregada do sistema.

Esta métrica avalia a eficiência e a equidade dos algorítmos. Fornece um valor, usualmente entre 0 e 1, o qual descreve quão próxima está uma regra de um a regra FCFS, para um sistema multiprocessador. Quanto maior for o 'fator-Q' maior é o desempenho da política. Um algorítmo que tem um 'fator-Q' maior que a unidade é o algorítmo 'shortest job first'.

Algumas da conclusões deste estudo são:

- Um desempenho muito variável do ponto de vista geral como particular, é obtido com diferentes estrategias.
- O algorítmo de balanceamento de carga é uma decisão de projeto crítica.
- Com o mesmo nível de informação disponível, algorítmos de iniciativa no servidor tem melhor resultado potencial em relação a algorítmos de iniciativa na fonte.
- O uso do 'fator-Q' fornece um indicador útil para comparar o desempenho das diferentes políticas.

Em [19] considera-se um sistema de computação distribuído composto por nós heterogêneos e uma rede de comunicação. Cada nó representa computadores hospedeiros e a rede é o meio de comunicação entre os nós que podem estar conectados em uma forma arbitrária. Cada nó contém um ou mais recursos. Os nós podem ser heterogêneos, isto é, eles podem ter diferente número de recursos, diferente configuração, e características de velocidade. Porém, têm a mesma capacidade de processamento, isto é, as tarefas podem ser processados do começo ao fim em um nó. Os nós e a rede são modelados como uma rede de fila em forma de produto.

As tarefas chegam ao nó de acordo com um processo de Poisson, com taxa ϕ_i. A taxa total de chegadas de tarefas tem taxa Φ.

Quando as tarefas chegam a um nó, este pode decidir executá-las localmente ou transferí-las para serem executadas em um outro nó. Quando a tarefa é transferida incorre em um retardo de comunicação que se adiciona ao retardo na fila do hospedeiro e ao tempo de processamento. A decisão de transferir uma tarefa é estática, isto é, não depende do estado do sistema.

O tempo de resposta de uma tarefa no sistema consiste do retardo no nó que o processa (tempo na fila mais o tempo de processamento), mais possíveis retardos de comunicação devidos à transferência de tarefas. Neste artigo supõe-se que a função de retardo médio no nó depende da carga no nó. O objetivo é balancear a carga com vistas a minimizar o tempo médio de resposta.

O desempenho é otimizado ao determinar a carga em cada hospedeiro que minimize o tempo de resposta de uma tarefa. É usado um problema de programação não linear para obter o resultado.

Os autores dividem os centros em três tipos: fonte , sumidouro e neutros. O centro de tipo fonte envia tarefas, mas não recebe; o sumidouro só recebe tarefas de outros nós, mas não envia tarefas; e o centro de tipo neutro processa as proprias tarefas locais.

Os autores assumem que a rede de comunicação satisfaz à propriedade de desigualdade do triângulo, sendo esta uma condição suficiente para encontrar uma solução ótima no caso de manter nós de tipo fontes, sumidouros ou neutros. Também supõem que o retardo do nó i ao nó j, é independente do par fonte-destino e é uma função do tráfego na rede.

Com estas suposições se constrói a função objetivo não linear sujeita a restrições lineares.

A solução ótima do problema satisfaz um conjunto de relações sujeitas à restrição de fluxo total (método dos multiplicadores de Langrange). Para obter estas relações são calculadas as derivadas parciais do número médio de tarefas no nó i com relação à taxa de processamento de tarefas no nó i (ou carga no nó i) o que fornece o retardo incremental no nó. Similarmente, a derivada parcial do número médio de tarefas na rede de comunicação com relação ao tráfego total na rede fornece o retardo de comunicação incremental.

Um algoritmo eficiénte, chamado de 'algoritmo de estudo paramétrico', gera a solução ótima em função do tempo de comunicação. Este algorítmo é apropriado para o estudo do efeito da velocidade da rede de comunicação na solução ótima. O segundo algorítmo é um 'algorítmo de ponto simples' e produz a solução ótima para um determinado conjunto de parâmetros.

Em [10] os autores não propõem nenhuma regra específica de balanceamento de carga para implementação, mas respondem à pergunta de qual deve ser o nível apropriado de complexidade, isto é, qual a quantidade de informação de estado que precisa ser coletada, para se obter bons resultados de uma regra dinâmica.

Regras dinâmicas que reagem com presteza às variações de estado do sistema requerem a coleta de muita informação de estado. Isto pode levar às seguintes considerações: em primeiro lugar o efeito do 'overhead' de administração de uma regra. Um 'overhead' excessivo pode negar os benefícios de uma regra de balanceamento. Uma segunda consideração é que inevitavelmente serão tomadas decisões inadequadas, devido à coleta de informação, o que empobrecerá o desempenho do sistema. Um último ponto é a instabilidade. Como regras complexas podem reagir rápidamente à mudanças de estado no sistema, é possível atingir o caso extremo em que os processadores gastariam todo seu tempo somente transferindo tarefas ('processor-thrashing'). Regras menos complexas tendem a reagir mais lentamente e portanto estão menos sujeitas ao problema de instabilidade.

O objetivo dos autores é comparar o desempenho de três regras de balanceamento (Random, Threshold, e Shortest), entre elas e com casos limites: sem balanceamento (representado por K independentes M/M/1) e, perfeitamente balanceada com custo zero (representada por uma M/M/K). A medida de desempenho é o tempo médio de resposta como uma função da carga do sistema.

O sistema analisado é um sistema distribuído com nós não idênticos, cada um deles composto de um processador simples. Os nós são conectados por uma rede 'broadcast' (Ethernet).

São analisadas três regras de localização (Random, Threshold e Shortest) as quais têm a mesma regra de transferência (Threshold).

A decisão de transferir uma tarefa de um nó a outro é baseada somente em informação local de um nó. A regra de transferência consiste em transferir uma tarefa de um nó , se o comprimento da fila local (incluindo a tarefa sendo processado) é maior que um certo limiar T.

As três regras de localização são definidas como segue:

Random: Com esta regra um nó destino é selecionado de forma aleatória e a tarefa é transferida a este nó, sem que exista troca de informação entre eles. Como as tarefas que chegam a um nó são tratadas como novas chegadas podem ser retransferidas se o comprimento da fila for maior que o limiar T, o que poderia causar instabilidade. Para evitar isto é colocado um limite no número de retransferências da tarefa.

Threshold: Um nó selecionado em forma aleatória é testado para determinar se sua fila esta abaixo do limiar T. Sendo assim a tarefa é enviada para ele. Senão outros nós são testados até localizar o nó adequado. Existe um número limite de testes a serem usados. Se este limite é ultrapassado então a tarefa é processada no nó de origem.

Shortest: Vários nós são escolhidos aleatoriamente e, a cada um deles é perguntado pelo tamanho da sua fila. A tarefa é transferida ao nó com fila menor. Se a filas têm tamanho maior que o limiar T, então o nó destino é quem deve processar a tarefa. O nó destino deve processar a tarefa transferida independente de seu estado no momento da chegada.

As três regras tem o mesmo modelo analítico, que trata o nó como um centro de filas. Os tarefas chegam ao centro com taxa λ. O tempo médio de serviço das tarefas é S. O fator de carga $\rho = \lambda \times S$, pode ser bastante maior que a utilização média dos nós por causa do custo de transferência das tarefas. O custo de transferência de tarefas de um nó é representado pelo custo de processamento de enviar a tarefa. O custo de receber uma tarefa está incluido no tempo de serviço da tarefa. Supõe-se o custo de testar ('probed') um nó desprezível. Ao transferir tarefas os nós dão prioridade de preempção sobre o processamento das tarefas. A

disciplina para atendimento das tarefas pode ser qualquer uma que escolha as tarefas independente de seu tempo de serviço.

Para simplificar a análise supõe-se que cada nó é estocásticamente independente do estado de qualquer outro nó. Por conseguinte, o modelo é desenvolvido para sistema homogêneo, portanto cada nó pode ser analisado isoladamente. O efeito do resto da rede sobre um nó individual é representado por um processo de chegada de tarefas transferidas. Como a rede é homogênea, as medidas de desempenho do sistema podem ser obtidas ao analisar um nó individual.

Todas as quantidades necessárias para determinar as taxas de transição de estado são parâmetros de entrada, exceto a descrição do processo de chegada das tarefas transferidas. Este processo de chegadas depende da regra de balanceamento de carga utilizada. A suposição de homogeneidade torna possível a determinação da taxa de tarefas transferidas: a taxa de chegada total deve ser igual a taxa total com que os nós transferem tarefas a outros nós e as probabilidades de estado estacionárias para possíveis nós destinos são as mesmas que para o próprio nó. Estas quantidades são saídas do modelo.

Para desenvolver as equações que modelam o sistema se consideram os nós nos seguintes estados:

- Processando: quando a fila do nó é menor ou igual ao valor do limiar T. Nesta fase um nó está ocioso ou processando tarefas.
- Transferindo: quando o tamanho da fila é maior que o valor do limiar T. Nesta fase o nó está transferindo tarefas ou processando tarefas que não podem ser transferidas.

Durante a fase de processamento o nó está ocioso ou processando tarefas. Nesta fase o modelo é uma cadeia de Markov de nascimento e morte. Em cada estado a taxa de chegadas é a soma da taxa de chegada de novas tarefas (λ) e a taxa de chegadas das tarefas transferidas a este nó pelo resto do sistema ($\lambda_t(n)$), este último termo depende do comprimento 'n' da fila no nó e da regra de balanceamento sendo modelada. Esta cadeia de Markov pode ser resolvida pelos métodos tradicionais.

Durante a fase de transfêrencia, o nó está ocupado seja transferindo tarefas ou processando tarefas que não podem ser transferidas, de acordo com a regra de localização. Esta fase é modelada como um período ocupado de duas classes, HOL M/M/1 com prioridade preemptiva, onde as classes são tarefas que estão sendo processadas e tarefas que estão sendo transferidas. A taxa total de chegadas ao nó é $(\lambda + \lambda_t(T^+))$ onde $\lambda_t(T^+)$ denota a taxa de chegada de tarefas transferidas ao nó condicionada a que o nó está na fase de transfêrencia. A proporção desta taxa total de chegadas consistindo de tarefas que serão transferidas e a proporção de tarefas que serão processadas depende da probabilidade de uma tarefa não ser transferida por causa de uma restrição na regra de localização. As análises correspondentes às fases de processamento e transfêrencia obtem probabilidades condicionais e medidas

de desempenho. Estas são combinadas usando pesos que representam a proporção do tempo que o sistema gasta em cada uma das fases, para determinar o desempenho total.

As medidas de desempenho obtidas incluem: tempo médio de resposta, utilização, comprimento da fila, taxa de transferência, e taxa de testes ('probes').

É feita uma simulação dirigida a evento, que entrega uma validação da decomposição utilizada nos modelos analíticos. A simulação usa o mesmo adotado para o modelo analítico, mas não faz a decomposição.

Os resultados sugerem que regras de balanceamento de carga muito simples, que usam pouca informação de estado do sistema, são de considerável valor prático, já que seu comportamento é muito melhor do que quando não existe balanceamento de carga; e é próximo ao das políticas mais complexas que usam mais informação. Os resultados também indicam que as políticas de Threshold são passíveis de implementação.

Em [14] estudam-se as características de desempenho de algorítmos simples de balanceamento de carga para sistemas distribuídos.

O processamento e a transmisão de mensagens para atualizar estado ('probed') e tarefas poderiam gerar um 'overhead' consideravél. Diferentes arquiteturas de sistemas podem impor custos muito diferentes para estes overheads. Neste artigo, é assumido que a arquitetura dos nós possui uma poderosa unidade de interface a qual é usada para processar muitos dos 'overheads' gerados pela transferência de tarefas e 'probed'. Apesar disso, no entanto, ocorrem retardos durante este procesamento.

Considera-se que os retardos para processar transferência de tarefas são muito maiores que os produzidos no processamento de 'probed', já que uma tarefa pode conter dezenas de pacotes e um 'probed' provavelmente precisa de somente um. Supõe que atrasos não desprezíveis são encontrados quando ocorrem transferências de tarefas.

Este artigo analisa o efeito deste retardo sobre o desempenho de três algorítmos (Forward, Reverse e Symmetric).

Cada um dos algorítmos de balanceamento de carga que são descritos a seguir possui um limiar T.

Forward O algorítmo é ativado cada vez que uma tarefa local chega ao nó. Se o número de tarefas neste nó (incluindo a tarefa atualmente sendo processada) é maior que $T+1$, é feito uma tentativa de transferir a tarefa recém chegada a outro nó. Um número finito de nós é verificado em forma aleatória para determinar uma localização para a tarefa. Um nó perguntado responde positivamente se o número de tarefas que ele possui é menor que $T+1$ e se não está esperando por alguma outra transferência de tarefas. Se mais de um nó

responde positivamente, o nó transmissor envia a tarefa ao nó escolhido de forma aleatória. Se nenhum dos nós perguntados responde positivamente, o nó espera outra chegada local antes de tentar outra vez.

Reverse: Este algorítmo é ativado cada vez que uma tarefa se completa no nó e o número total de tarefas nele é menor que $T + 1$ e o nó não está esperando transferência de uma tarefa. Sendo assim o nó pergunta um subconjunto de nós para adquirir uma tarefa. Somente os nós que possuem mais de $T + 1$ tarefas (incluindo a atualmente executada) podem responder positivamente. Se mais de um nó responde, o nó que pergunta escolhe aleatoriamente um deles para enviar a tarefa.

Symmetric: Este algorítmo combina Forward e Reverse. Então, se a fila de um nó ultrapassa o liminar mais um ao chegar uma tarefa, ele tenta transferir tarefas, e se cai abaixo de $T + 1$ após processar uma tarefa, tenta adquirir uma tarefa de outros nós.

Para a análise matemática é formulado um modelo teórico de filas, para cada um dos algorítmos.

Supõe-se que o processo de chegadas de tarefas a cada nó é um processo de Poisson (λ). Os tempos do serviço e transferência de tarefas , também são supostos exponenciais com médias $1/\nu$ e $1/\gamma$ respectivamente. O tempo de transferência de tarefas inclui o tempo desde o início da transferência de um nó até a recepção no nó destino. Supõe-se também que o tempo de 'probed' é zero, baseado no fato de que o tamanho de uma tarefa é muito maior do que uma mensagem de 'probed'. Portanto, supõe-se também que o único retardo que ocorrerá na rede será devido à transferência de tarefas. Outra suposição feita é que os nós são homogêneos, isto é, eles têm a mesma taxa de chegadas e mesmo serviço. As tarefas serão atendidas segundo a política de atendimento FCFS em cada nó.

O modelo markoviano da rede completa possui um grande espaço de estados, o que o faz intratável computacionalmente. Para poder analisá-la emprega-se a técnica de decomposição. Sendo assim cada nóé analisado independentemente dos outros. A interação entre os nós devido às transferência de tarefas é modelado como uma modificação de chegadas ou partidas em cada nó.

A análise dos algorítmos é realizada usando a técnica de solução Geométrica de Matrizes, a qual fornece uma solução exata do modelo para cada nó, isto é as probabilidades de estado estacionário são calculadas exatamente. Supõe-se que os valores dos parâmetros são conhecidos e deriva-se com estes as equações necessárias para calcular os parâmetros desconhecidos. No caso de Symmetric são: a probabilidade de não encontrar um lugar para a tarefa, em resposta a um conjunto de 'forward probes'; a probabilidade de não encontrar uma tarefa remota para um conjunto de 'reverses probes'; a taxa à qual o nó envia tarefas ao nó que perguntou; a taxa de receber 'forward probes'. Através de um procedimento iterativo e usando a suposicão de homogeneidade e simetria do algorítmo, são determinados estes valores. O número de iterações para conseguir isto é relativamente pequeno, em torno de 10 a 30 iterações, independentemente do valor inicial dos parâmetros.

Foi feita uma simulação para validar os resultados do modelo analítico. O modelo de simulação consistiu de 10 nós em todos os casos exceto quando $\rho = 0.9$, onde o modelo consta de 20 nós. O intervalo de confiança usado foi de 95 % .

Alguns dos resultados mais relevantes foram os seguintes:

- O algorítmo Symmetric tem melhor desempenho para retardos moderados, isto é, retardos menores ou iguais ao tempo médio de serviço. Quando os retardos decrescem o desempenho é quase o mesmo para as três políticas.
- Para retardos grandes, isto é, maiores ou iguais ao tempo médio de serviço, o tempo de resposta não é melhor que para o caso não balanceado, nas situações de baixa carga. Para carga alta ($\rho \geq 0.9$) existem benefícios.
- Os benefícios de balancear a carga são maiores para cargas altas.
- O limiar ótimo parece ser função da carga e do tempo de transferência das tarefas. Para baixa carga o limiar ótimo foi 0 para todas as cargas consideradas. Porém, quando os retardos eram maiores, o limiar otimo aumentou correspondentemente.
- A política de Forward é melhor que Reverse em quase todas as cargas exceto para alta carga. A política Symmetric é a melhor das tres políticas testadas, porém tem o potencial de produzir grande 'overhead' de 'probes'.

Em [18] propõe-se um método descentralizado de balanceamento da carga para um sistema de tempo real distribuído. Em um sistema de tempo real a probabilidade de que uma tarefa seja perdida deve ser a menor possível, uma vez que uma tarefa perdida pode levar a consequências desastrosas. O artigo propõe um método de balanceamento da carga no qual cada nó mantém a informação de estado de um pequeno conjunto de nós (por exemplo, os vizinhos ao nó), chamado um 'buddy set' A coleta de informação para atualizar o estado do 'buddy set' é feita em forma 'broadcast'.

São definidos 3 limiares para definir o estado de carga de um nó.

TH_u : limiar de pouco carregado;

TH_f : limiar de carregado;

TH_v : limiar de sobrecarregado.

Um nó está

pouco carregado se QL (comprimento da fila) $\leq TH_u$
medianamente carregado se $TH_u <$ QL $\leq TH_f$
completamente carregado se $TH_f <$ QL $\leq TH_v$
sobrecarregado se QL > THv

Quando um nó chega ao estado de 'completamente carregado' ou 'pouco carregado', comunica o fato em forma 'broadcast' a seus vizinhos, de forma tal que os nós que o mantém no 'buddy set' atualizarão seu estado. Não existe um retardo significativo ao mudar o estado em forma broadcasting; desde que o 'buddy set' de cada nó seja formado por aqueles nós na sua proximidade física, o retardo em informar uma mudança de estado e/ou transferir tarefas não deverá ser muito grande. Cada nó que recebe esta informação atualizará sua informação de estado, ao eliminar o nó 'completamente carregado' ou ao incluir o nó 'pouco carregado' na lista ordenada ('preferred list') de servidores disponíveis. Um nó sobrecarregado pode selecionar o primeiro nó da sua lista ordenada e transferir tarefas para ele. O propósito de construir uma lista ordenada de preferidos para cada um dos nós é evitar problemas de retardo de 'probing' e o problema de 'dumping', isto é, que o mesmo nó seja selecionado, por mais de um nó sobrecarregado, para transferir suas tarefas. A probabilidade de isto ocorrer é pequena, já que isto aconteceria quando esse nó fosse o primeiro da lista de preferidos dos nós sobrecarregados. O custo de tranferir tarefas é uma função crescente com a distância física entre o nó transmissor e o nó receptor. Para reduzir este custo, o nó receptor deveria estar localizado tão perto do nó fonte como possível. A lista de preferidos de cada um dos nós é então estruturada sobre a base do número de pulos entre o nó fonte e o nó destino. Quando existe mais de um nó com igual número de pulos, então estes nós devem ordenar-se para minimizar o problema de 'dumping'.

No sistema proposto, os nós são conectados por uma rede arbitrária, e cada nó e equipado com um processador de rede, que manеja as comunicações usuais e a transferência de tarefas entre os nós. Um nó é composto de um servidor simples. Existem duas fontes de chegadas de tarefas ao nó, as tarefas externas e as transferidas. As tarefas que chegam ao nó podem ser executadas localmente ou remotamente em qualquer outro nó do sistema.

Uma cadeia de Markov embutida no tempo é usada para modelar o desempenho do método de balanceamento da carga proposto. A saída de uma tarefa do sistema especifica um ponto embutido. Os autores começam desenvolvendo um modelo exato, a partir do qual acham uma solução aproximada e um modelo aproximado chamado 'upper bound model'. A solução exata é muito próxima da solução aproximada e é limitada superiormente pela solução do modelo de 'upper bound'.

O modelo de 'upper bound' é gerado sob a suposição que cada nó pode transferir sempre somente uma das tarefas que sobrepassam o limiar a outro nó e o resto destas tarefas ficam enfileiradas no nó. Com esta suposição as transições entre os estados simplificam-se muito e pode ser calculada uma solução para este modelo simplificado. Esta solução será uma fronteira superior para a solução exata. A solução do modelo aproximado é calculada por meio de uma aproximação em dois passos, uma vez que as equações tem 2 parâmetros de cálculo muito complexos. No primeiro passo o modelo é resolvido, calculando a distribuição de probabilidades do comprimento das filas (q_i) e as taxas de transferência de tarefas (τ), para quando a probabilidade de ter exatamente zero nós disponíveis para transferir as tarefas dentro do buddy set é igual a zero ($\epsilon = 0$). O segundo passo, é calcular os ϵ baseados

nos q_is e τs calculados no primeiro passo. Posteriormente, em forma iterativa, o procedimento se repete até que os q_is e τs convirjam para valores fixos. Os autores demostram que o método iterativo converge em um número finito de passos.

O desempenho do método de balanceamento de carga proposto é avaliado com o modelo 'upper bound', a solução aproximada e simulação. Os dois primeiros são usados para derivar a distribuição do comprimento da fila em cada um dos nós e a probabilidade de perder a tarefa ('meeting deadlines'), e analisar os efeitos do tamanho do 'buddy set', a freqüência da mudança de estado e o tempo médio de cada tarefa no sistema. Por outro lado a simulação é usada para verificar os resultados analíticos.

Os autores, em [4], tentam ampliar e melhorar o método dado por Ni e Hwang em [16] para as probabilidades de atribuição. Procura-se uma solução adaptativa que trabalhe enfrentando parâmetros de carga desconhecidos e mudáveis.

Considera-se um sistema multiprocesador com um gerente central. Cada servidor possui uma fila . A sua carga de trabalho tem uma componente local ou dedicada e uma componente genérica. As tarefas locais da classe i podem somente ser servidas no servidor i, enquanto que tarefas genéricas podem ser encaminhadas a qualquer servidor. Todas as tarefas tem requerimentos de serviço identicamente distribuídos. Os servidores podem ter diferentes velocidades de serviço. O gerente central atribui tarefas genéricas a diferentes servidores.

O modelo consiste de N servidores de filas com um espaço ilimitado de espera. Assume-se que as chegadas são processos de Poisson. A i-ésima fila recebe um fluxo de tarefas dedicados com taxa λ_i. As tarefas genéricas chegam ao sistema com taxa λ, e podem ser processadas por qualquer servidor. Uma vez alocada a um servidor uma tarefa não pode ser reatribuída e deverá ser processada completamente por este servidor. O tempo médio de serviço num servidor é exponencialmente distribuído. As tarefas genéricas chegam ao gerente, que atribui cada tarefa a um servidor em forma probabilística, por exemplo, para o servidor i se atribui com probabilidade u_i.

Supõe-se que o tempo de processamento no gerente central e os retardos de comunicação entre o gerente e os servidores é desprezível.

O objetivo é procurar as probabilidades de atribuição do gerente assim como minimizar o tempo médio de resposta de uma tarefa no sistema.

Este problema é formulado como um problema de programação não linear sujeito a certas restrições lineares. Sendo este problema geral e difícil de resolver, os autores primeiro obtem um problema equivalente ao usar as propriedades de 'processor sharing' dos servidores, supondo também que as velocidades dos servidores é a mesma e as tarefas tem tempos de serviço idênticos e geralmente distribuídos. Este problema equivalente usa os mínimos quadrados das distâncias, para medir o desvio da distribuição da carga a partir de uma condição perfeitamente balanceada, produzindo uma solução ótima para o problema de minimização do tempo de resposta. Para um sistema homogêneo, isto é, todos os servidores tem igual tempo de

serviço (distribuído exponencialmente), o problema de atribuição de probabilidades ótimas para a divisão do fluxo de chegadas genérico, é equivalente a que os tempos ponderados de servidores ociosos são balanceados no sentido dos mínimos quadrados ponderados, onde os pesos estão relacionados à raíz quadrada da velocidade dos servidores.

São feitas medidas do tempo ocioso dos servidores. Cada um dos servidores é amostrado em forma independente a uma taxa de v vezes por segundo. Com estas medidas o gerente ajusta periódicamenteAs probabilidade de atribuição do gerente são mantidas fixas durante este intervalo de medição e são atualizadas somente quando chega a nova medida ao gerente.

A partir das medidas de tempo ocioso de servidor é obtido um novo conjunto de medidas. As probabilidades de divisão ótimas são simplesmente a projeção da raíz desta função en um simplex não-negativo em R^N. Propõe-se um algorítmo de aproximação estocástica para resolver o problema de encontrar a raíz. Uma versão com ganho constante faz o algorítmo sensivel às mudanças da carga oferecida.

Simulação é usada para demonstrar o desempenho destes algorítmos no problema de filas original. Os resultados da simulação mostram que a convergência para um tempo de resposta ótimo é extremamente rápida, embora a aproximação das probabilidades de atribuição seja mais lenta.

O algorítmo de aproximação estocástica tem uma convergência mais rápida. O algorítmo dos mínimos quadrados modificados converge rapidamente, enquanto que o algorítmo de gradiente estocástico é um pouco mais lento.

de Souza e Silva e Gerla em [8] fornecem um algorítmo eficiente para a solução de uma ampla variedade de problemas de balanceamento estático.

O sistema distribuído é modelado como uma coleção de modelos de servidores centrais com terminais, um para cada hospedeiro, interconectados por um modelo de redes de filas de comunicação.

As tarefas são divididas em sub-tarefas, que podem ser executadas localmente, se o hospedeiro possui os recursos necessários ou pode ser transferida a um dos outros hospedeiros do sistema.

Supõe-se que são permitidas multiplas classes de tarefas. Tarefas que representam a carga local são restritas a rodar no hospedeiro onde são geradas, isto é, não estão sujeitas a balanceamento e são representadas por cadeias fechadas. Todas as outras tarefas no modelo podem ser executadas em um ou mais hospedeiros e são representadas por cadeias abertas. Uma vez que uma tarefa é gerada, ela muda de classe ao mover-se pelos hospedeiros. As tarefas são transferidas aos hospedeiros através de uma rede de comunicação que possui uma solução em forma de produto.

O problema de balanceamento de carga é definido como um problema de programação não linear, onde a função objetivo a ser minimizada é o retardo

global médio do sistema (D(N)), com relação aos fluxos de tarefas cada um dos centros de serviço. O retardo global é definido coma a soma ponderada do retardo de todas as cadeias. Os fatores de ponderação são parâmetros que permitem ao projetista evitar ineqüidade para as diferentes classes.

O método de solução usado é a técnica de 'downhill' baseada no método de 'Flow Deviation'. O desenvolvimento está restrito a uma cadeia fechada simples por hospedeiro, mas pode ser estendida a multiplas cadeias fechadas por hospedeiro.

Para computar a direção do passo de descida para a técnica 'downhill' é necessário obter as derivadas parciais da função de retardo global com relaç ao a cada um dos fluxos das cadeias fechadas em cada um dos centros.

Os autores também consideram tarefas que podem gerar ('spawn') outras tarefas consistindo de uma ou mais sub-tarefas , (por exemplo, carga 'background' gerada por usuários interativos). Estas tarefas são executadas em seqüência e não requerem sincronização em nenhum ponto durante a execução. Supõe-se que as tarefas interativas podem rodar somente em seu hospedeiro local. As tarefas 'spawned' podem ser atribuídas a um conjunto de 'sites' na rede. Sup oe-se, por simplicidade, que as tarefas interativas dentro de um hospedeiro' são representadas por uma cadeia fechada simples e as tarefas 'spawned' são compostas de tarefas simples e são modeladas, aproximadamente, como cadeias abertas com taxas relacionadas por cadeias fechadas.

O objetivo, como no caso anterior, é minimizar o retardo global distribuindo o tráfego das cadeias abertas entre os diferentes 'sites'.

Os autores também fornecem um algorítmo para Balanceamento da Carga (FDLB) em uma rede de computadores distribuída, como a considerada neste artigo.

O algorítmo FDLB encontra o mínimo global para D(N), uma vez que esta função é a soma de funções estritamente convexas sobre o fluxo de cadeias abertas, isto é, o retardo de cada um dos 'sites' é uma função convexa crescente sobre os fluxos das cadeias abertas e portanto, a soma ponderada desses retardos também é convexa. A convergência é garantida pelo fato de que cada iteração produz uma melhora na função objetivo.

O artigo também fornece várias aplicações e resultados numéricos que demostram a eficácia de seu uso.

Em [17] considera-se um sistema de computação distribuído que consiste de M computadores hospedeiros heterogêneos, conectados por uma rede de comunicação. Uma tarefa gerada por um hospedeiro, pode ser genérica e neste caso pode ser processada por qualquer dos hospedeiros do sistema, ou pode ser dedicada, e então ela deve ser processada no hospedeiro no qual foi gerada. Supõe-se que cada hospedeiro pode gerar vários tipos de tarefas dedicadas, onde cada tipo impõe restrições no tempo médio de tempo de reposta. O trabalho de [17] possui dois

objetivos: o primeiro é redistribuir as tarefas genéricas entre os M computadores hospedeiros a fim de minimizar o tempo médio de resposta (problema de balanceamento da carga); o segundo, projetar um controle para cada computador hospedeiro, para que as tarefas dedicadas e genéricas sejam escalonadas de forma tal a cumprir as restrições de tempo de resposta para cada um dos tipos de tráfego dedicado.

Os autores explicam que o modelo proposto é mais versátil que o proposto por [19], devido a que não somente consideram o problema de redistribuir o tráfego genérico na presença de retardo de comunicação, para minimizar o tempo de resposta, mas também acrescentam a característica de que cada um dos computadores hospedeiros escalona o tráfego dedicado e o tráfego local associado.

A complicação adicional no modelo de balanceamento de carga é a dependência do tempo de resposta das tarefas genéricas nas regras de escalonamento.

O balanceamento de carga proposto é estático, mas o escalonamento local é dinâmico.

Os autores supõem que as tarefas genéricas são geradas em cada hospedeiro i de acordo com um processo de Poisson independente com taxa ϕ_i. Também, que as tarefas dedicadas são geradas nos diferentes hospedeiros de acordo com um processo de Poisson de taxa λ_{ij}. Supõem que os tempos de serviço seguem uma distribuição geral. Assume-se que em cada hospedeiro a capacidade do buffer é infinita e que a política de atendimento é não-preemptiva. Portanto, modelam cada hospedeiro como um sistema de filas não-preemptiva M/G/1.

O problema de balanceamento é definido como um problema de programação não linear, onde a função objetivo a minimizar é o tempo médio de resposta das tarefas genéricas. Este tempo é a soma de dois componentes: o primeiro termo é o retardo médio ponderado (em razão da permanência em fila e em estado de processamento) das tarefas genéricas; o segundo termo é o retardo de comunicação esperado das tarefas genéricas.

O problema de alocação das tarefas genéricas em cada hospedeiro pode ser resolvido como um problema de otimização polimatróide. A estrutura polimatróide leva a um algorítmo eficiente para determinar o retardo médio das tarefas em cada hospedeiro. Devido também a esta estrutura a solução ótima é derivada diretamente, isto é, não é necessario resolver um problema de programação linear.

A forma de solução é a seguinte: primeiro, se determina para cada hospedeiro e de forma descentralizada, os retardos produzidos pelas tarefas dedicadas. Uma vez que todas estas funções de retardo tenham sido determinadas localmente, são transmitidas a um hospedeiro central, onde o algorítmo de [19] é rodado para encontrar a alocação ótima do tráfego genérico. Como uma forma de simplificar os autores omitem o segundo termo da função objetivo, que é o termo do retardo de comunicação. A alocação ótima das tarefas genéricas, deve ser transmitida a cada um dos hospedeiros, para que as regras de escalonamento local, isto é, retardos médios do tráfego genérico alocado e o grupo de maior prioridade o qual inclui

o tráfego genérico, sejam colocadas em forma concordante com o calculado. Esta computação também pode ser feita de forma descentralizada, isto é , cada um dos hospedeiros pode realizar seu próprio cálculo baseado na alocação ótima das tarefas genéricas e os outros dados conhecidos do hospedeiro.

O modelo de balanceamento dinâmico, usado nesta tese, considera retardos na atualização da informação de estado do sistema. Dos poucos autores que levam em consideração o efeito de algum tipo de retardo no balanceamento da carga, podemos mencionar [19] que usa um modelo de balanceamento estático centralizado, o qual considera retardo de comunicação na transferência das tarefas. Este retardo tem duas componentes: o retardo para enviar a tarefa de seu nó origem ao nó onde será processado e o retardo de enviar a resposta ao nó de origem. [17] usa o mesmo modelo de [19], considerando inicialmente o retardo de comunicação na função objetivo, mas, posteriormente, para simplificar a solução, o retardo não foi considerado. Em [18], os autores consideram retardo na transferência das tarefas e na atualização de estado para um sistema dinâmico de tempo real, e [14] assumem que o retardo completo acontece durante a transferência das tarefas, em um modelo dinâmico distribuído.

Capítulo IV

O paradigma da metodologia orientada a objeto

IV.1 A Metodologia Orientada a Objeto

A seguir é dada uma descrição da metodologia orientada a objeto para a especificação de modelos markovianos [3].

Um sistema modelado segundo esta metodologia é composto de um conjunto de componentes chamados de objetos que interagem entre si através da troca de mensagens. Cada objeto é uma entidade que possui um estado interno que evolui com o tempo. O estado interno de um objeto muda devido a um evento gerado por ele mesmo ou ao receber uma mensagem de um outro objeto. O estado de um objeto determina os eventos que ele pode gerar e as taxas com que estes eventos ocorrem. Um evento pode mudar o estado interno do objeto que o gerou, sem afetar nenhum outro, mas em geral causará alguma reação em outros objetos. Esta reação é modelada pela troca de mensagens. A especificação de um objeto inclui a definição de como o objeto reage com o recebimento de mensagens, a especificação dos eventos que ele pode gerar e as ações tomadas quando há ocorrência de um evento.

O estado global do sistema é formado pelo conjunto de estados internos dos objetos e a lista de mensagens não entregues. As mensagem são entregues em tempo zero e é nulo o tempo de reação do objeto a uma mensagem recebida. Isto da origem a alguns estados transitórios (nos quais existem uma ou mais mensagens não entregues). Nestes estados o tempo de permanência é zero, e para propósitos de análise, não são considerados, já que somente tem interesse os estados tangíveis (tempo de permanência no estado é diferente de zero). As ações que podem acontecer em resposta a um evento ou mensagem geram um conjunto de estados transitórios que começam e terminam num estado tangível.

Em [9] foi implementada uma ferramenta para a descrição de sistemas em alto nível e a partir dessa descrição gerar a cadeia de Markov correspondente à

evolução do sistema.

IV.1.1 Descrição Formal do Modelo

A descrição formal encontra-se em [3], e se repete aqui para o entendimento deste trabalho.

Um objeto $\mathcal{O}$ é definido formalmente da seguinte forma :

$$\mathcal{O} \triangleq (I, S, \mathcal{S}_0, \varepsilon, M^r, M^s, P, \delta', \delta)$$

Onde

I = Nome do objeto (deve ser único).

S = Conjunto de possíveis estados do objeto.

$\mathcal{S}_0 \in S$ = Estado inicial do objeto.

ε = Conjunto de eventos que o objeto pode gerar.

M^r = Conjunto de mensagens que o objeto pode receber.

M^s = Conjunto de mensagens que o objeto pode enviar.

R = Função taxa do objeto :

$$R : S \times \varepsilon \rightarrow \Re^+$$

dado um estado $s \in S$ e un evento 'e', esta função entrega a taxa na qual o evento ocorre.

δ' = Função de eventos do objeto :

$$\delta' : S \times \varepsilon \rightarrow \{(0,1] \times S \times [M^s]\}$$

$$[M^s] \in M^s \text{ é uma lista ordenada de mensagens}$$

δ = Função de mensagens do objeto.

$$\delta : S \times M^r \rightarrow \{(0,1] \times S \times [M^s]\}$$

Na definição formal do objeto dada acima, δ' é uma função tal que, dado um estado e um evento, retorna o conjunto de possíveis respostas ao evento, onde cada resposta consiste de um novo estado, da lista ordenada de mensagens a serem entregues e da probabilidade desta resposta ocorrer. Associada a cada evento existe a função de taxa do objeto, que retorna a probabilidade do evento ocorrer.

A função δ dado um estado s e uma mensagem m, retorna o conjunto de possíveis respostas, onde cada uma consiste de um novo estado, de uma lista ordenada de mensagens a serem estregues e da probabilidade desta resposta ocorrer.

Um sistema χ modelado com esta metodologia, consiste do conjunto de estados $\mathcal{S}$, um estado inicial S_0 e uma função de probabilidade γ.

$$\chi \triangleq \{\mathcal{S}, S_0, \gamma\}$$

Onde o conjunto de estados $\mathcal{S}$ do sistema χ é definido por :

$$\mathcal{S} = \{< S_1, S_2, \ldots, S_N >, [m_1, \ldots, m_n]\}$$

Onde N é o número de objetos do modelo, $s_i \in S, e \quad 1 \leq i \leq N \qquad m_i \in M$ é o conjunto de todas as mensagens do sistema.

O estado inicial S_0 é definido por :

$$S_0 =< S_{1_0}, S_{2_0}, \ldots, S_{N_0} >$$

A função de transição γ é definida em termos das funções δ e δ' como segue :

$$\gamma : (< S_1, \ldots, S_i, \ldots, S_N >, [m_1, \ldots, m_k])$$

$$\rightarrow (< S_1, \ldots, S_i', \ldots, S_N >, [m_2, \ldots, m_k, m_{k+1}, \ldots, m_{k+m}])$$

se a mensagem m_1 tem como destinatário o objeto $\mathcal{O}_i$, onde $m_{k+1}, \ldots, m_{k+n}$ são mensagens geradas pelo objeto $\mathcal{O}_i$ como resposta da mensagem m_1 ou seja :

$$(p, s_i', [m_{k+1}, \ldots, m_{k+n}]) \in \delta_i(s_i, m_1), p > 0$$

$$\gamma : (< S_1, \ldots, S_i, \ldots, S_N >, [])$$
$$\rightarrow (< S_1, \ldots, S_i', \ldots, S_N >, [m_1, \ldots, m_k])$$

se e é um evento que o objeto $\mathcal{O}_i$ gerou, e :

$$(p, S_i', [m_1, \ldots, m_k]) \in \delta_i'(S_i, e)$$

Um estado é alcançável se existe uma seqüência válida do estado inicial S_0 a ele. Um estado alcançável sem mensagens a serem entregues ($< s_1, \ldots, s_N >$, []), é chamado tangível.

IV.1.2 Ferramenta de Análise do Modelo Orientado a Objeto

Baseada nesta descrição formal da metodologia orientada a objeto para a especificação e geração de modelos Markovianos, existe uma ferramenta de software [9], implementada em *PROLOG*, linguagem qur oferece muitas vantagens sobre as linguagens convencionais [3,9], pois não exige a especificação dos tipos de dados utilizados, permitindo inteira liberdade na descrição dos estados dos objetos, e fornecendo unificação, uma forma poderosa de iniciação de variáveis e casamento de padrões (usado na verificação das pré-condições das regras para decidir a ação). Prolog, ainda, implementa o 'backtracking, permitindo que mais de uma regra tenha suas condições satisfeitas ao mesmo tempo (todas elas são testadas para achar todos os estados alcançáveis). Uma desvantagem desta linguagem é a baixa eficiência na compilação e execução.

Esta ferramenta é organizada em quatro níveis (figura IV.1, descritos a seguir :

1. O núcleo aceita a descrição do sistema em termos de objetos, eventos e mensagens. A partir desta descrição e de um estado inicial, o núcleo gera a matriz de transição de estados do sistema.

2. Objetos podem ter o mesmo comportamento, diferindo apenas no valor dos parâmetros especificados para cada objeto. No nível de definição do tipo de objeto, diferentes tipos de objetos são especificados. Um determinado modelo pode conter mais de uma ocorrência do mesmo tipo de objeto. Objetos diferentes mas do mesmo tipo, poderão ter parâmetros diferentes. Neste nível pode existir uma biblioteca de tipos de objetos.

3. No nível de aplicação, o usuário define um modelo criando objetos utilizando-se da biblioteca de tipos de objetos definidos no nível inferior e especificando os parâmetros necessários a cada objeto.

4. O nível de interface permite que o usuário final se abstraia dos níveis anteriores, oferecendo uma linguagem de alto nível para a especificação do modelo.

Esta ferramenta é usada como parte da ferramenta que avalia o desempenho de políticas de balanceamento de carga.

IV.1.3 Programa Núcleo

O programa Núcleo é o encarregado da exploração da árvore de alcançabilidade do sistema especificado. Este programa aceita, como entrada, a descrição do sistema em termos de objetos, eventos e mensagens. A partir desta especificação e de um estado inicial, o programa Núcleo explora os estados alcançáveis do sistema. Esta exploração é feita em forma *'breadth-first'*.

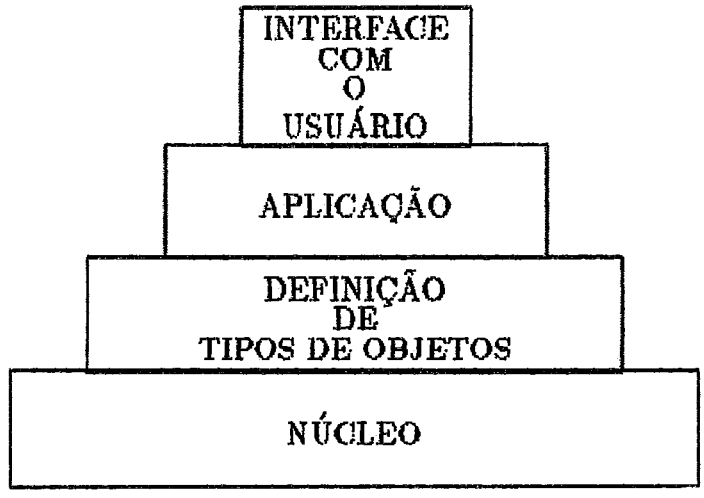

Figura IV.1: Níveis da ferramenta

O programa Núcleo foi implementado em Prolog [9]. A descrição de objetos e do estado inicial do sistema é feita com cláusulas Prolog. Por exemplo, a especificação do estado inicial (feita no nível de alplicação) é feita com uma lista, cujos componentes são os estados iniciais de cada objeto, como no exemplo a seguir :

```
INITIAL(F(CPU1,S(1)).F(CPU2,S(2)).
        F(SCHEDULER,S(Q(CPU1,1).Q(CPU2,2).NIL)).NIL).
```

Este é um *fato* Prolog cujo argumento é a lista contendo o estado inicial do sistema. Neste exemplo *'CPU1'*, *'CPU2'*, *'SCHEDULER'* representam os nomes dos diferentes objetos e *'S(1)'*, *'S(2)'*, *'S(Q(CPU1,1). Q(CPU2,2). NIL)'* o estado inicial de cada objeto.

Abaixo se descreve brevemente como o Núcleo gera os estados alcançáveis de um sistema a partir de cláusulas Prolog que descrevem os objetos. É importante ressaltar que um usuário da ferramenta implementada em [9] não especifica estas cláusulas da maneira indicada abaixo. As cláusulas são automaticamente geradas para o Núcleo, a partir de uma definição de mais alto nível.

A descrição de um objeto é feita em função do estado local de cada objeto, manipulando-o, enviando mensagens ou recebendo mensagens. Isto é feito por meio de cláusulas Prolog. Por exemplo, se o estado é formado como no exemplo acima, uma transição espontânea que modifica o estado interno do objeto *'*COMP'* restando um a este estado, indicando com isto o processamento de um tarefa, será descrita da seguinte forma :

```
REACT(F(*COMP,S(*NUM)),*NEW_RATE,*STATE,*NEXT_STATE) ←
    TYPE(*COMP,TYPE) &
    RATE(*COMP,*RATE) &
    ROUTE(*COMP,*DEST,*PROB) &
    MULTIPLY(*PROB,*RATE,*NEW_RATE) &
    DIFF(*NUM,1,*NEW_NUM) &
    SUBST(F(*COMP,S(*NUM)),*STATE,F(*COMP,S(*NEW_NUM)),
    *NEW_STATE) &
    ⋮
```

Nesta especificação indica-se o tipo do objeto a que pertece a componente (TYPE(...)), a taxa de disparo desta transição (RATE(...)). A cláusula ROUTE(...) unifica as variáveis *DEST e *PROB com os valores correspondentes especificados no nível de aplicação, A cláusula MULTIPLY multiplica a taxa de disparo com a probabilidade de ir ao destino para formar a nova taxa de disparo deste estado *NEW_RATE. O estado interno do objeto *COMP é modificado ao restar 1 ao número de tarefas (*NEW_NUM), para logo calcular o novo estado e formar o novo estado global '*NEW_STATE', através da claúsula SUBST(...). O programa núcleo executa estas cláusulas para gerar os estados alcanáveis dados pela variável *NEW_STATE.

A especificação do envio e recebimento de mensagens é feita da mesma forma descrita anteriormente, ou seja, o envio de uma mensagem corresponde ao chamado de uma cláusula Prolog que ao executar acha o novo estado do sistema, por exemplo :

```
REACT(F(*COMP,S(*NUM)),*RATE,*STATE,*NEXT_STATE) ←
    ⋮
    MESSAGE(MENS(ARRIVAL),*DEST,*,*NEW_STATE,*NEXT_STATE) &
    ⋮
```

Na especificação do objeto destinatário da mensagem 'ARRIVAL' a reação é descrita gerando o novo estado global, por exemplo :

```
MESSAGE(MENS(ARRIVAL),*COMP,1,*STATE,*NEW_STATE) ←
    ⋮
    Reação do objeto a mensagem 'ARRIVAL' &
    ⋮
```

O programa Núcleo está implementado em VMS/Prolog do IBM [1].

O programa gera como saída a matríz de transição do sistema ('STATES PROLOG') e a descrição dos estados alcançados ('RATES PROLOG') .

A cláusula principal do programa Núcleo é a cláusula 'TRACE(...)', a que tem por função encontrar para cada objeto (*COMP), todas as transições disparáveis e o estado atingido com cada transição. Em seguida, compara se o estado atingido já foi alcançado antes, se não, o numera e armazena numa estrutura de dados '*VISITED', mantida com este propósito. Esta estrutura está organizada segundo uma *hash table* para otimizar o tempo de procura. Cada posição da Hash Table é composta por uma lista de estados.

Quando todos os objetos do sistema já foram explorados (todas as transições de saída foram achadas) um novo estado é explorado. Para se fazer isto, é mantida uma estrutura de dados (*FRONT-*BACK) (esta é uma lista diferencial para agilizar o seu manejo [12]), onde se armazenam todos os estados não explorados.

Quando a fila *FRONT-*BACK está vazia, todos os estados já foram explorados e o algoritmo termina. Caso contrário a busca continua, a menos que uma outra condição de parada seja satisfeita.

IV.1.4 Exemplos

A seguir, será indicado como especificar modelos com a metodología orientada a objeto explicada anteriormente. Serão dados dois exemplos. Ambos sistemas consistem de dois servidores e um gerente (ver figura IV.2). No primeiro exemplo o gerente atualiza sua informação com um tempo distribuído exponencialmente, e no segundo exemplo o gerente atribui probabilidades em forma paramétrica.

Exemplo 1

Este modelo é de um sistema simples de balanceamento de carga como mostrado na figura IV.2. O sistema consiste de um gerente central, o encarregado de encaminhar as tarefas ao servidor correspondente, e de dois servidores (CPUs), encarregados de processá-los. O gerente possui um estado que está dado pela 2-tupla (n_1, n_2), onde n_i é o número de tarefas na fila da CPU_i. Inicialmente, o gerente possui um estado qualquer, que vai mudando com o tempo, já que trata de manter uma visão atualizada ao obter informação de estado dos servidores em intervalos de tempo exponencialmente distribuidos com média $1/\nu$.

Quando ocorre a chegada de uma tarefa, o gerente envia-a ao servidor suposto de ter a fila de menor tamanho. Se as filas são de igual tamanho, ele escolhe um servidor ou outro com probabilidade $1/2$

Neste primeiro exemplo, o gerente possue a seguinte política de balanceamento:

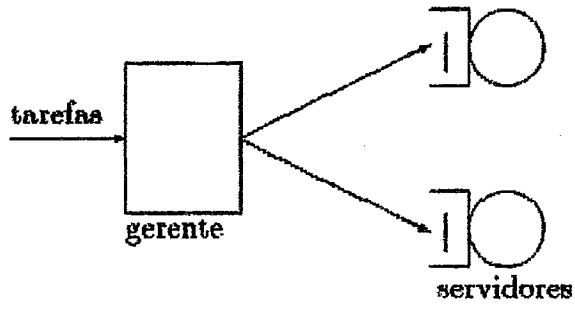

Figura IV.2: Um modelo para balanceamento da carga

1. $n_1 = n_2$ o gerente escolhe uma cpu ou outra com igual probabilidade.
2. $n_1 > n_2$ ou $n_1 < n_2$ escolhe a cpu com fila de menor tamanho e envia a tarefa para ela.

O único evento gerado pelo gerente é a atualização do seu estado, ao obter a informação de estado dos servidores. Quando recebe a mensagem de chegada de uma tarefa, compara o tamanho das filas (da descrição de estado que ele possui), e decide com este estado, para qual das cpus enviar a tarefa. A seguir envia uma mensagem de chegada ('arrival') à CPU que ele acredita ter o menor número em fila.

TIPO: schedule;
NOME:nome_objeto;
ESTADO:n1,n2;
EVENTO :
AÇÃO:atualiza_estado(cpu1,*state,n1)
atualiza_estado(cpu2,*state,n2)
RATE: ν
MENSAGEM "arrival" : (n1,n2) → (n1,n2);
CONDIÇÃO: n1 = n2
AÇÃO: message(mens(arrival), CPU1);
PROBABILIDADE:0.5
MENSAGEM "arrival" : (n1,n2) → (n1,n2);
CONDIÇÃO: n1 = n2
AÇÃO: message(mens(arrival), CPU2);
PROBABILIDADE:0.5
MENSAGEM "arrival" : (n1,n2) → (n1,n2);
CONDIÇÃO: n1 > n2
AÇÃO: message(mens(arrival), CPU2);
PROBABILIDADE:1
MENSAGEM "arrival" : (n1,n2) → (n1,n2);
CONDIÇÃO: n1 < n2

AÇÃO: message(mens(arrival), CPU1);
PROBABILIDADE:1

Exemplo 2

O sistema consiste de dois centros de serviço e um gerente, como se mostra na figura IV.2. A diferença para o exemplo anterior está na definição do gerente. Neste exemplo o gerente atribui as tarefas a cada servidor com probabilidades paramêtricas. Isto gerará uma cadeia de Markov genérica que será processada por uma ferramenta de software para poder avaliar as probabilidades de atribuição. Esta probabilidades adquirirão seu valor dependendo da política de balanceamento usada.

A descrição é feita em uma linguagem de alto nível, no nível de aplicação da ferramenta para a geração de cadeias markovianas. A cada chegada de uma tarefa o gerente somente gera mensagens de chegada ('arrival') para o servidor correspondente. Esta atribuição é feita dependendo da definição de estado que ele possua no momento da chegada da tarefa.

TIPO: schedule1;
NOME:nome_objeto;
ESTADO:n1,n2;
MENSAGEM "arrival" : (n1,n2) → (n1,n2);
AVALIAÇÃO: Member(F(*dest,s(*num)), *state);
Route(nome_objeto, *dest, *prob1);
AÇÃO: message(mens(arrival),*dest);
PROBABILIDADE:*prob1;

Cada um dos objetos que compõe o sistema deve ser descrito com base em regras e fatos Prolog, se eles não se encontrarem numa biblioteca de objetos.

Descrição do sistema no nível de aplicação

No nível de aplicação, a descrição do sistema formado por uma fonte que gera as chegadas, um gerente com uma regra de balanceamento como a do exemplo 2, e dois servidores, seria da seguinte forma.

TYPE(FONTE,*FONTE*).
(SCHEDULER,*SCHEDULE*).
TYPE(CPU1,*SERVIDOR*).
TYPE(CPU2,*SERVIDOR*).

```
INITIAL(F(CPU1,S(0)).F(CPU2,S(0)).F(FONTE,S(6)).
        F(SCHEDULE,S(Q(CPU1,0).Q(CPU2,0).NIL)).NIL).

RATE(CPU1,U).
RATE(CPU2,U).
RATE(FONTE,L).

CAPACIDADE(CPU1,3).
CAPACIDADE(CPU2,3).

ROUTE(CPU1,FONTE,1).
ROUTE(CPU2,FONTE,1).
ROUTE(FONTE,SCHEDULE,1).
ROUTE(SCHEDULE, CPU1, P1).
ROUTE(SCHEDULE, CPU2, P2).

< – RECONSULT(FONTE).
< – RECONSULT(SCHEDULE).
< – RECONSULT(SERVIDOR).
```

O fato INITIAL define que as filas dos servidores CPU1 e CPU2 estão inicialmente vazias, enquanto a fila da FONTE tem 6 tarefas na fila. Também especifica que o estado interno do gerente corresponde ao caso no qual os servidores possuen filas vazias.

O fato TYPE indica a que tipo pertence o objeto. A descrição feita anteriormente serve para qualquer sistema de dois centros e um gerente central, o que vai fazer mudar as características do sistema são as definições de cada objeto. Por exemplo, neste caso o tipo do objeto Scheduler está especificado através do fato TYPE(SCHEDULER, SCHEDULE1). Este fato indica que o objeto Scheduler será do tipo definido no segundo exemplo. Para mudar de tipo de objeto dever-se-ia descrever o objeto gerente de outro tipo, por exemplo, como descrito no exemplo 1. Para isto somente chegaría trocar o tipo, neste caso TYPE(SCHEDULER, SCHEDULE), acrescentar o fato RATE(SCHEDULER, NU) que indica a taxa de atualização do gerente, e indicar ao compilador carregar as cláusulas que o descrevem, através de RECONSULT(SCHEDULE).

Capítulo V

Implementação

A seguir são descritos os diferentes programas implementados para a obtenção de medidas de desempenho de modelos de balanceamento de carga, e sua interrelação.

A ferramenta computacional consiste de dois ambientes: o primeiro é o ambiente dos programas que estão na linguagem Prolog e o segundo é o ambiente dos programas em Pascal. Os dois ambientes estão relacionados entre si por arquivos, como é mostrado na figura V.1.

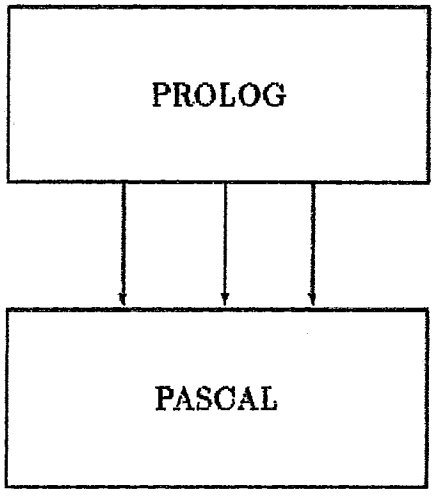

Figura V.1: Relação entre os ambientes

A seguir será descrita, de forma geral, o ambiente e depois de forma particular, cada um dos arquivos.

Na figura V.2 é mostrado de forma mais detalhada estes dois ambientes e os programas que os compõem. Os programas são desenhados com linha cheia, enquanto que os arquivos o são com linha pontilhada. As flechas indicam o sentido da comunicação de dados.

V.1 Ambiente Prolog

No ambiente Prolog do IBM (VM/Prolog), é usado o programa Núcleo descrito em capítulo IV, para a geração simbólica da cadeia de Markov. Neste ambiente encontram-se também a descrição do sistema a ser estudado, através de cláusulas e fatos Prolog e outros programas de propósito específico, como:

- SOMANUM
- LIST_PAR
- BALANC

Todos os que serão explicados mais detalhadamente a seguir.

Para simplificar a explicação não serão colocadas as extensões dos arquivos, .PAS para programas Pascal e PROLOG para os arquivos do ambiente Prolog.

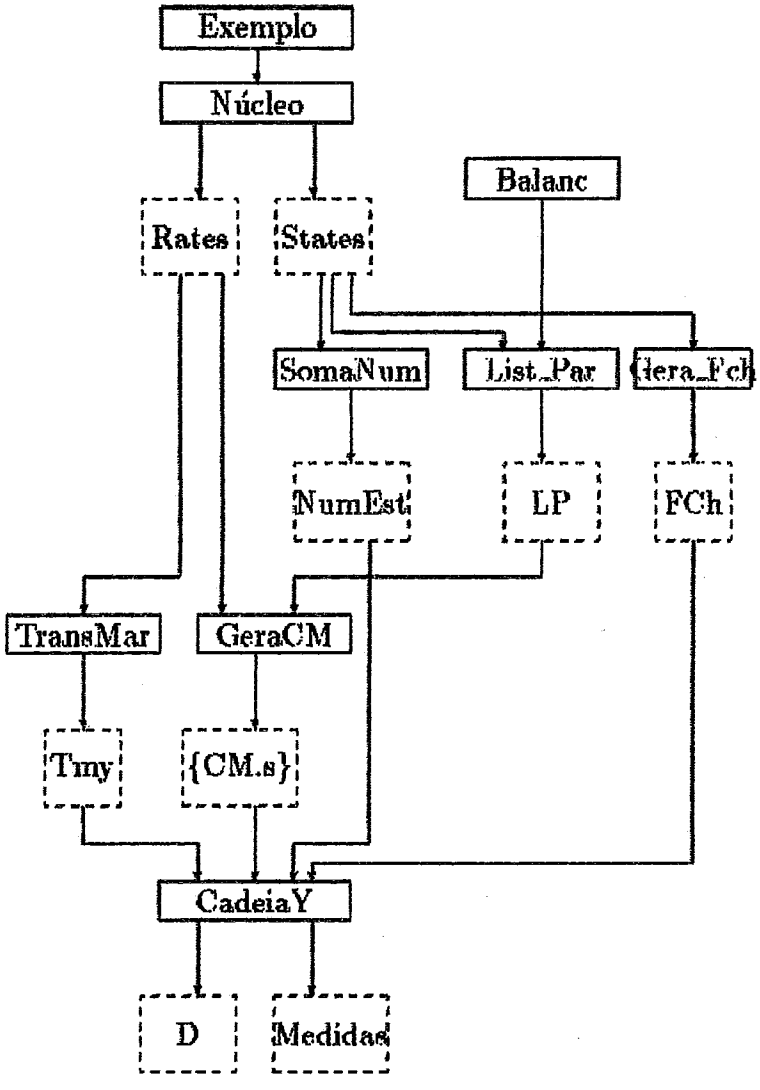

Figura V.2: Esquema de implementação do sistema para solução do problema de balanceamento

1. **EXEMPLO:** O sistema que será estudado é descrito com base em *fatos* Prolog, no nível de implementação da ferramenta computacional para Cadeias Markovianas (explicado no capítulo anterior).

   O arquivo que contém a descrição de um sistema que consiste de dois servidores, um gerente que possui uma regra de atribuição genérica (como a mostrada no exemplo 2 do IV), e uma fonte que simula a chegada das tarefas, possui os seguintes *fatos*:

```
TYPE(FONTE,FONTE).
TYPE(SCHEDULER,SCHEDULE).
TYPE(CPU1,SERVIDOR).
TYPE(CPU2,SERVIDOR).

INITIAL(F(CPU1,S(0)).F(CPU2,S(0)).F(FONTE,S(6)).
        F(SCHEDULE,S(Q(CPU1.0).Q(CPU2,0).NIL)).NIL).

RATE(CPU1,U).
RATE(CPU2,U).
RATE(FONTE,L).

CAPACIDADE(CPU1,4).
CAPACIDADE(CPU2,4).

ROUTE(CPU1,FONTE,1).
ROUTE(CPU2,FONTE,1).
ROUTE(FONTE,SCHEDULE,1).
ROUTE(SCHEDULE, CPU1, P1).
ROUTE(SCHEDULE, CPU2, P2).

< – RECONSULT(FONTE).
< – RECONSULT(SERVIDOR).
< – RECONSULT(SCHEDULE).
```

   A seguir são descritos os diferentes *fatos* usados neste exemplo:

   - TYPE indica o tipo a que pertence o objeto, assim TYPE(CPU1, SERVIDOR) indica que o objeto CPU1 é do tipo SERVIDOR.
   - RATE indica a taxa de serviço de cada objeto.
   - CAPACIDADE é a capacidade da fila dos servidores. Por exemplo, o servidor CPU1 tem uma capacidade de 2 tarefas na fila (incluindo a que está em serviço).
   - ROUTE é a cláusula que indica o caminho que seguem as tarefas após serem servidas, e a probabilidade de tomar esse caminho. Por exemplo, ROUTE(SCHEDULE, CPU1, P1) indica que o objeto de tipo SCHEDULE roteia a tarefa para a CPU1 com probabilidade (paramètrica) P1

(posteriormente, esta probabilidade será avaliada pela regra de balanceamento específica).

- INITIAL define o sistema e o estado inicial de cada objeto que representa este sistema.
- Os predicados RECONSULT(nome_arquivo) carregam na memória as claúsulas contidas no arquivo 'nome_arquivo'. Isto é, RECONSULT(SERVIDOR) colocará na memória do computador as claúsulas para o objeto SERVIDOR, contidas no arquivo SERVIDOR, que serão usadas por todos os objetos de tipo SERVIDOR.

Cada sistema diferente deve ter uma descrição feita desta maneira.

2. **NÚCLEO:** Este programa [9] gera a Cadeia de Markov (estados e matriz de transições) para um sistema, a partir de um estado inicial e da definição do sistema em termos de objetos, eventos e mensagens. A saída deste programa é a cadeia de Markov mostrada na figura V.3, a qual está descrita nos arquivos 'STATES' e 'RATES'.

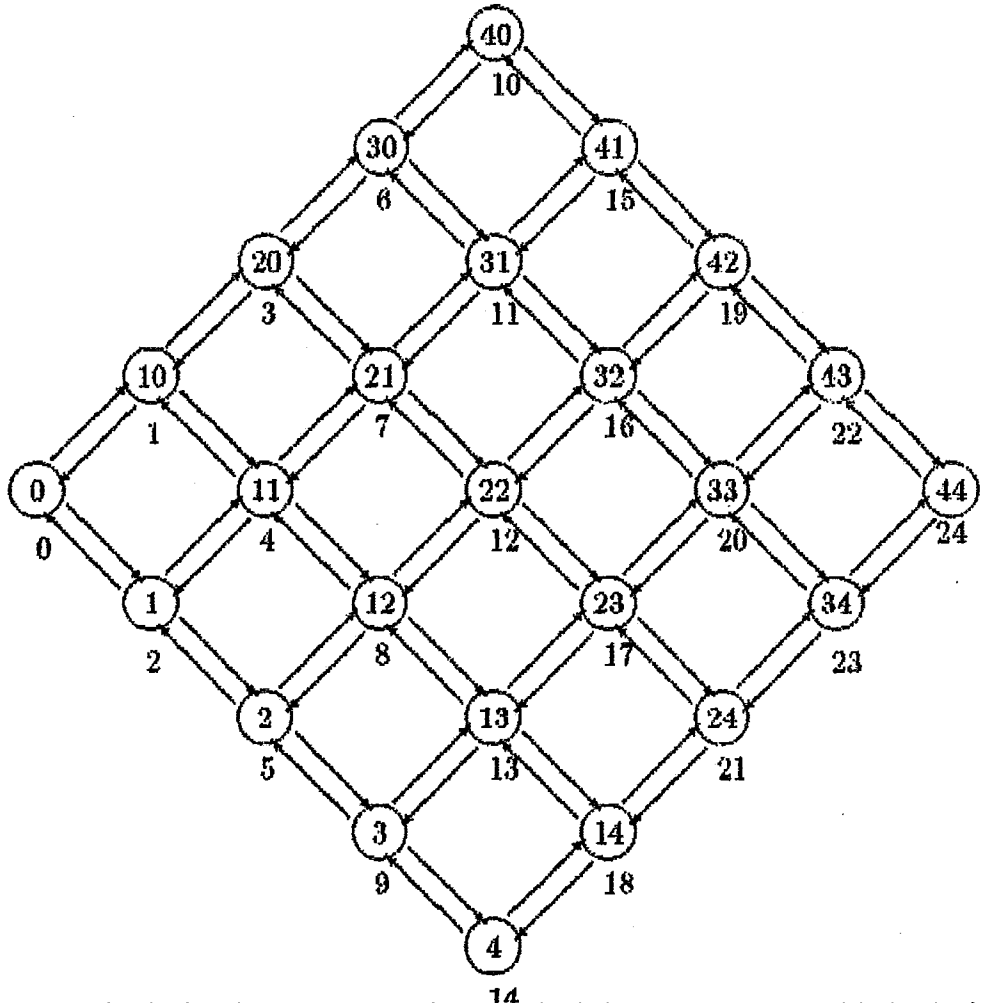

Figura V.3: Cadeia de Markov para um sistema de dois centros e capacidade de fila de três tarefas

O arquivo STATES contém a descrição simbólica dos estados da cadeia de Markov. Para o exemplo, o arquivo STATES é:

```
(F(CPU1,S(0)) . F(CPU2,S(0)) . F(FONTE,S(100)) . NIL)
(F(CPU1,S(1)) . F(CPU2,S(0)) . F(FONTE,S(99)) . NIL)
(F(CPU1,S(0)) . F(CPU2,S(1)) . F(FONTE,S(99)) . NIL)
(F(CPU1,S(2)) . F(CPU2,S(0)) . F(FONTE,S(98)) . NIL)
(F(CPU1,S(1)) . F(CPU2,S(1)) . F(FONTE,S(98)) . NIL)
                ⋮
```

Neste arquivo, a numeração dos estados da cadeia de Markov está dada em forma relativa. É assim, que a primeira descrição de estado corresponde à do estado zero, a segunda à do estado 1, etc.

O arquivo RATES contém a matriz de transições de estado. Estas taxas são geradas em forma paramêtrica, e posteriormente adquirirão valor ao serem computadas, pela ferramenta desenvolvida neste trabalho, levando em consideração a regra de balanceamento específica. Para o exemplo sendo mostrado, uma parte do arquivo 'RATES' seria:

estado inicial	estado final	transição
0	1	1 * P1 * L
0	2	1 * P2 * L
1	0	U
1	3	1 * P1 * L
1	4	1 * P2 * L
2	0	U
2	4	1 * P1 * L
2	5	1 * P2 * L
3	1	U
⋮		

A primeira coluna (denotada como 'estado inicial') indica o estado de início da transicão; a segunda coluna ('estado final') indica o estado em que o sistema fica após a transição e a terceira coluna ('transição') indica a taxa com a qual o sistema faz uma transicão. Neste exemplo a transição está dada em forma paramêtrica. Onde L é a taxa de chegada de tarefas ao sistema e U é a taxa de serviço nos servidores, estes dois valores são dados de entrada definidos pelo usuário na especificação do modelo. P1 é a probabilidade de uma tarefa ser atribuida ao servidor CPU1 pelo gerente, e P2 é a probabilidade de uma tarefa ser atribuida, pelo gerente ao servidor CPU2. O valor destas probabilidades será determinado de acordo com as regras de balanceamento para cada caso específico.

3. **BALANC:** Este arquivo contém as regras de balanceamento de uma determinada regra que serão aplicadas ao sistema. São estas regras as que avaliam, para cada estado inicial diferente, as probabilidades de atribuição paramêtricas, dadas pelo objeto Scheduler ao gerar a cadeia de Markov genérica.

   Por exemplo, para o caso de um sistema de dois centros e um gerente cuja regra de balanceamento é a de sempre atribuir os tarefas que chegam ao sistema ao servidor que tem a fila de menor tamanho; as regras de balanceamento e suas respectivas probabilidades são:

```
/* Se fila da CPU1 é maior que a fila da CPU2, então envia
a tarefa que chega para a fila da CPU2 com probabilidade 1.
REGRA_BALANC(F(CPU1,S(*NUM1)).F(CPU2,S(*NUM2)).*TAIL,
PROB1,*PROB2,1) < -
        GT(*NUM1,*NUM2) &
        *PROB1 = 1 &
        *PROB2 = 0 .
        ⋮

/* Se fila da CPU1 é menor que a fila da CPU2, então envia
a tarefa que chega para a fila da CPU1 com probabilidade 1.
REGRA_BALANC(F(CPU1,S(*NUM1)).F(CPU2,S(*NUM2)).*TAIL,
PROB1,*PROB2,2) < -
        LT(*NUM1,*NUM2) &
        *PROB1 = 0 &
        *PROB2 = 1 .
        ⋮
/* Se fila da CPU1 é igual que a fila da CPU2, então envia
a tarefa que chega para a fila da CPU1 e para a fila da CPU2
com probabilidade 0.5
REGRA_BALANC(F(CPU1,S(*NUM1)).F(CPU2,S(*NUM2)).*TAIL,
PROB1,*PROB2,3) < -
        EQ(*NUM1,*NUM2) &
        *PROB1 = 0.5 &
        *PROB2 = 0.5
        ⋮
```

   Para cada regra de balanceamento diferente deve ser criado um novo arquivo que contenha as regras de balanceamento apropriadas ao caso.

4. **SOMANUM:** Esta rotina soma o número de tarefas que estão nas filas dos servidores para cada estado da Cadeia de Markov. Faz isto para cada estado do arquivo de entrada 'STATES' e gera um arquivo de saida 'NUMEST' que possui o número total de tarefas nos servidores, para cada estado. Este arquivo é usado, posteriormente, pelo programa CMY.PAS para o cálculo do número médio de tarefas no sistema num intervalo de tamanho T.

5. **LIST_PAR:** Como a cadeia de Markov é gerada pelo NÚCLEO, em forma simbólica, se faz necessário determinar as probabilidades de atribuição (P1, P2, etc), para cada estado da cadeia gerada. Este programa gera a lista de parâmetros necessários para avaliar as taxas de transição da matriz de transições de estado genérica gerada pelo NUCLEO. Precisa, como entrada, do arquivo das regras de balanceamento (ex: 'BALANC') para uma determinada regra e o arquivo de estados ('STATES'). Com estes dois arquivos calcula os valores das probabilidades de atribuição aos servidores, para cada um dos estados iniciais. O nome do parâmetro que identifica a probabilidade e o seu valor são escritos em um arquivo de saída (para este exemplo 'LP2 PROLOG'), como é mostrado a seguir:

```
P1      0.5
P2      0.5

P1      0
P2      1

P1      1
P2      0

P1      0
P2      1

P1      0.5
P2      0.5
 ⋮
```

   Neste arquivo, cada estado está separado do seguinte por uma linha em branco.

6. **GERA_FCH:** Este programa procura os estados que estão com o buffer de um centro em sua capacidade máxima. Isto é, percorre o arquivo 'STATES' comparando o número de tarefas que possui um centro com o valor da capacidade máxima da fila para esse centro. Se estes dois valores são iguais, então o escreve em uma estrutura de dados. Faz isto para cada um dos centros do sistema. A estrutura de dados é escrita em um arquivo de saída chamado 'FCH'. O arquivo 'Fch' é usado pelo programa 'CMY.PAS' para o calcular a medida do tempo que um centro permanece com a fila cheia.

V.2 Ambiente Pascal

Neste ambiente encontra-se a ferramenta computacional desenvolvida neste trabalho, que permite calcular a Cadeia de Markov embutida (D), assim como medidas transientes e estacionárias, simultaneamente para vários tamanhos do intervalo T.

Como os pontos embutidos no tempo representam o instante no qual o gerente atualiza sua informação de estado, isto significa que o sistema evolui durante o intervalo T como um processo de Markov correspondente ao estado no qual o gerente se encontra no começo do intervalo (logo após a atualização). Os programas estão implementados para avaliar as cadeias de Markov diferentes e com elas calcular a matriz de transição nos pontos embutidos e a medidas de desempenho tais como Vazão média do sistema, Tempo médio de resposta, Número médio de tarefas no sistema. Cabe ressalvar, que estes programas, estão feitos para analisar sistemas que sejam representados por cadeias de Markov com qualquer número de estados.

A seguir os programas deste ambiente são explicados com maiores detalhes.

1. **GERACM:** Este programa avalia as Cadeias de Markov simbólicas diferentes, que correspondem à matriz geradora Q da cadeia de Markov que evolui entre cada atualização de estado do gerente. Estas geradoras são usadas para o cálculo da matriz de transições nos pontos embutidos no tempo. Em outras palavras, este programa transforma o arquivo de transições simbólico em um arquivo de transições numérico. Como entrada para este programa são necessários os seguintes arquivos:

   - **'Rates':** arquivo gerado pelo Núcleo como 'RATES PROLOG' e transportado para o ambiente Pascal com o nome antes dito. Possui as taxas de transições de estado da cadeia de Markov genérica, em forma paramêtricas (ver explicação de RATES PROLOG).
   - **'LP':** gerado pelo programa LIST_PAR no ambiente prolog, e transportado para o ambiente Pascal. Contém o nome e os valores das probabilidades de atribuição (parâmetros), para cada estado.

   Gera como saída o conjunto das cadeias de Markov correspondente a cada regra de balanceamento diferente, para cada um dos estados gerados pelo Núcleo. Estes arquivos tem por nome CM.s , onde 's' é o número da regra de balanceamento.

2. **TRANSMAR:** gera uma matriz com as indicacão de quais são as transições que indicam uma tarefa processada. Devido à forma como os estados são gerados pelo programa Núcleo, quando ocorre uma transição entre um estado de maior numeração a um estado de menor numeração, indica que houve um tarefa processada. Se a tarefa é processada indica-se isto colocando um '1' na transição correspondente. Se a tarefa não foi processada, então é colocado um '0' (zero) na transição. O arquivo de entrada necessário é 'RATES.DAT' (o antigo RATES PROLOG). Como saida gera o arquivo TMY.MTX. Como exemplo temos o seguinte pedaço do arquivo que corresponde ao exemplo dado no começo deste capítulo.

25 Rows
25 Columns

From	To	Rate
1	2	0
1	3	0
2	1	1
2	4	0
2	5	0
3	1	1
	⋮	

3. **CMY:** gera a matriz de transições nos pontos embutidos no tempo D. Para isto, são randomizados os processos de Markov (CM.s), correspondentes a cada estado inicial. Posteriormente, com estas cadeias de Markov calculam-se cada uma das linha da matriz de transições nos pontos embutidos ($\vec{d}_i$) mediante as equações apresentadas no capítulo II.

   Também são calculadas algumas medidas de interesse, como: tempo médio de tarefas no sistema, tempo médio de resposta, vazão média do sistema, etc. Tanto a matriz D como as medidas são calculadas simultaneamente para varios tamanhos do intervalo entre atualizações.

   Para obter uma maior eficiência do programa, foi necessário expressar as equações para o cálculo das linhas da matriz D, assim como as medidas de desempenho, na forma de equações recorrentes.

   Os arquivos de entrada necessários para este programa são:

   - **CM.s:** os arquivos correspondentes a cada processo de Markov diferente, gerado pelo uso de uma regra de balanceamento (de uma regra em particular) e por um estado inicial particular.
   - **TMY:** contém as transições marcadas, isto é, se existe uma transição de um estado numerado em forma superior a um estado com numeração menor, isto indica que houve um tarefa processada e a transição receberá um '1', em outro caso '0'. Este arquivo serve para calcular a vazão média no sistema.
   - **NUMEST:** contém a população global de cada estado. É gerado no ambiente prolog pelo programa SOMANUM, e transportado para o ambiente Pascal. É usado para o cálculo do número médio de tarefas no sistema em estado estacionário.
   - **FCH:** especifica os estados que têm os centros com o buffer cheio. É gerado pelo programa GERA_FCH no ambiente Prolog. Usa-se para o cálculo do tempo médio que um centro permanece com seu buffer cheio.

Os arquivos de saída gerados pelo programa são:

- **DY.t:** matriz de transições para a cadeia de Markov nos pontos embutidos no tempo, correspondente a um intervalo de tamanho 't'.
- **N.MTX:** contém o número médio de tarefas no sistema. Se as medidas para vários tamanhos de intervalo foram calculadas em forma simultânea, os valores do número médio de tarefas no sistema para cada intervalo está neste arquivo.
- **T.MTX:** Arquivo que contém os valores da vazão média do sistema para cada intervalo calculado simultaneamente.
- **TRESP.MTX:** Arquivo com os valores do tempo médio de resposta no sistema, para todos os intervalos calculados em forma simultânea.

Capítulo VI

Resultados Numéricos

VI.1 Modelos e Regras de Balanceamento Analisadas

Para avaliar a importância do retardo na aquisição da informação de estado do sistema, serão obtidas medidas de desempenho tais coma vazão média, tempo de resposta médio, número médio de tarefas no sistema, percentual do tempo em que as filas permanecem saturadas, etc, para diferentes regras de balanceamento de carga.

A metodologia descrita nos capítulos anteriores para a obtenção das medidas e a implementação estão feitas para analisar um sistema com qualquer número de servidores. Entretanto, neste capítulo, ser ao analisados sistemas com apenas dois e três centros, devido a problemas de limitação de memória na execução dos programas. As conclus oes, entretanto, não são afetadas por esta limitação.

O modelo pode ser também de gerente descentralizado, ou seja, existe um gerente em cada um dos centros, encarregado de distribuir as tarefas que chegam a esse centro dependendo da regra de balanceamento nele implementada.

A seguir são descritas as regras de balanceamento analisadas e os sistemas usados.

Modelo 1 Um sistema distribuído homogêneo de dois servidores e um gerente. O gerente é o encarregado de distribuir as tarefas que chegam ao sistema (com taxa de chegada λ), aos servidores, de acordo com uma probabilidade de atribuição dada por uma regra de balanceamento de carga. Os servidores possuem fila de tamanho limitado (buffer). A taxa de serviço (μ) de ambos servidores é a mesma.

Modelo 2 Um sistema distribuído heterogêneo de dois servidores e um gerente. O gerente é o encarregado de distribuir as tarefas que chegam ao sistema aos

servidores, de acordo com uma probabilidade de atribuição dada por uma regra de balanceamento de carga. Os servidores possuem fila de tamanho limitado. A taxa de serviço dos servidores é diferente.

Modelo 3 Um sistema distribuído homogêneo de três servidores e um gerente. O gerente é o encarregado de distribuir as tarefas que chegam ao sistema aos servidores, de acordo com uma probabilidade de atribuição dada por uma regra de balanceamento de carga. Os servidores possuem fila de tamanho limitado.

A descrição de estado dos diferentes modelos apresentados anteriormente, é a k-tupla $(n_1, \ldots, n_k)$; onde n_1 representa o número de tarefas que se encontram na fila da CPU1, n_2 é o número de tarefas na fila da CPU2 e n_k é o número de tarefas na fila da CPUk.

A seguir são descritas as regras de balanceamento analisadas.

Regra 1

Esta regra pode ser representada pelas seguintes condições:

- Se $n_1 = n_2$, então as tarefas são enviadas para a CPU1 ou para a CPU2 com probabilidade $p_1 = p_2 = 0.5$.
- Se $n_1 < n_2$ envia para a CPU1 com probabilidade p_1 e para a CPU2 com probabilidade $p_2 = 1 - p_1$.
- Se $n_1 > n_2$ envia para a CPU2 com probabilidade p_2 e para a CPU1 com probabilidade $p_1 = 1 - p_2$.

Por exemplo:

- Se $n_1 > n_2$, então as tarefas são enviadas para a fila da CPU2 com probabilidade $p_2 = 0.9$ e para a CPU1 com probabilidade $(1 - p_2) = 0.1$.
- Se $n_1 < n_2$, então as tarefas são enviadas para a fila da CPU1 com probabilidade $p_1 = 0.9$ e para a CPU2 com probabilidade $(1 - p_1) = 0.1$.
- Se $n_1 = n_2$, então as tarefas são enviadas para a CPU1 ou para a CPU2 com probabilidade $p_1 = p_2 = 0.5$.

Serão usadas probabilidades p_1 que podem variar entre 0.5 e 1.

Regra 2

Nesta regra o gerente atribui as probabilidades em forma proporcional ao número de tarefas que possuem as filas dos servidores.

- A probabilidade de atribuição à fila da CPU1 é igual a $n_1/n_1 + n_2$; e para a fila da CPU2 a probabilidade de atribuição vem dada por $n_2/n_1 + n_2$;
- Se $n_1 = 0$ e $n_2 = 0$, então as tarefas são atribuídas com probabilidade 0.5 para ambos servidores.

Regra 3

O seguinte algorítmo serve para ilustrar o caso do sistema não ter um gerente central. O sistema é distribuído e o gerente neste caso somente é um artificio de implementação.

Uma observação importante a ser feita é a de que uma tarefa não chega ao sistema, e sim a uma ou a outra CPU. Neste caso, um tarefa chega a um determinado servidor que pode escolher processá-la ou enviá-la ao outro servidor.

Para cada centro o algorítmo de balanceamento é o seguinte:

Existe um limiar mínimo (Nmin) para a fila do servidor, abaixo do qual ele pode receber tarefas de outros servidores. Há também um limiar máximo (Nmax), acima do qual um servidor não pode receber mais tarefas de outros servidores. Este servidor deverá enviar a tarefa para ser processada em outra CPU e, se isto não é possível, deve processá-la.

Se uma tarefa chega na CPU1, o algorítmo de balanceamento, expressado em pseudo-Pascal é:

Se fila1 $<$ Nmax então
    a CPU1 deve processar a tarefa.

Se fila1 $\geq$ Nmax então
    Se fila2 $<$ Nmin
        então manda a tarefa para a fila da CPU2 com prob. 1
        de outra maneira a tarefa é processada na CPU1;

O algorítmo equivalente deve ser usado na CPU2.

Regra para três centros

Para o sistema de três centros, a regra de balanceamento usada é a seguinte:

- se $n_1 = n_2 = n_3$ então $p_i = 1/3$
- se $n_1 = n_2 < n_3$ então $p_1 = p_2 = 1/2$ e $p_3 = 0$
- se $n_1 = n_3 < n_2$ então $p_1 = p_3 = 1/2$ e $p_2 = 0$
- se $n_2 = n_3 < n_1$ então $p_2 = p_3 = 1/2$ e $p_1 = 0$
- se $n_1 < n_2$ e $n_1 < n_3$ então $p_1 = 1$ e $p_2 = p_3 = 0$
- se $n_2 < n_1$ e $n_2 < n_3$ então $p_2 = 1$ e $p_1 = p_3 = 0$
- se $n_3 < n_1$ e $n_3 < n_2$ então $p_3 = 1$ e $p_1 = p_2 = 0$

A seguir são mostrados os experimentos feitos e as medidas obtidas. Em cada caso são especificados os parâmetros usados.

VI.2 Vazão média do sistema

Os parâmetros de entrada são:

- Modelo 1
- Regra 1, $p_1 = 1, p_2 = 0$
- $\lambda = 0.9$ [tarefas/seg]
- $\mu = 1$ [tarefas/seg]
- Buffer = 3 tarefas

O gráfico da figura VI.1 corresponde à vazão média do sistema versus T, para os parâmetros especificados.

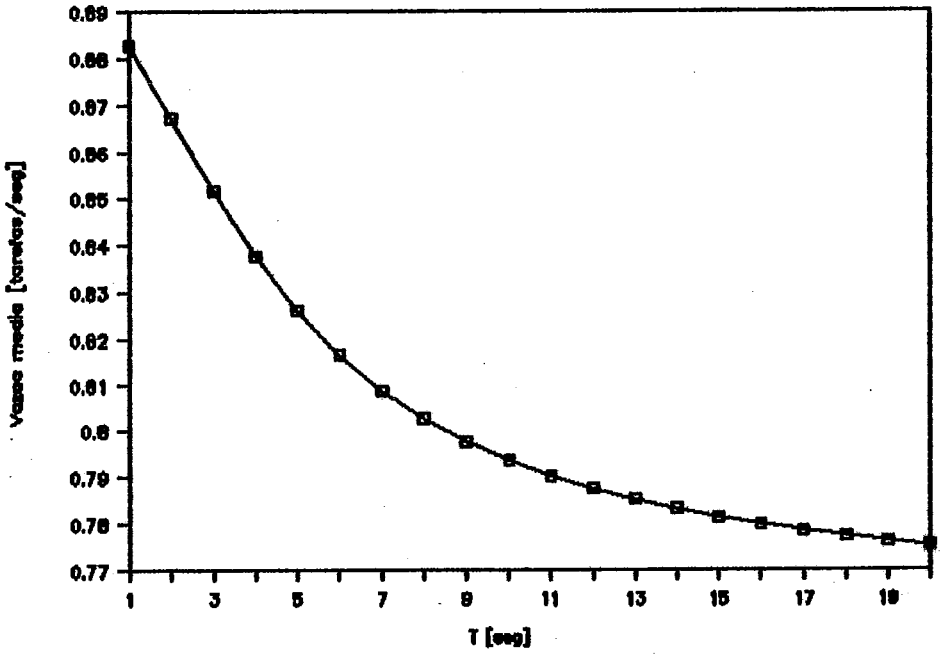

Figura VI.1: Vazão Média do Sistema vs T

No gráfico da figura VI.1 vemos que a vazão média é maior quando $T \rightarrow 0$, isto devido a que quando o tamanho do intervalo entre atualizações de estado do gerente é muito pequeno (tendendo a zero), o gerente sabe com maior exatidão o comportamento do sistema, e pode atribuir as tarefas aos centros com menor carga. Por outro lado, quando o tamanho do intervalo é muito grande, ou seja, a atualização do gerente é feita a intervalos grandes de tempo, o gerente trabalha com informação desatualizada tomando neste casos decisões erradas e atribuindo as tarefas, que chegam, a filas já sobrecarregadas, este fato faz com que a vazão do sistema vá diminuindo com o aumento de T.

VI.3 Número médio de tarefas no sistema

Os parâmetros de entrada são:

- Modelo 1
- Regra 1. $p_1 = 1, p_2 = 0$
- $\lambda = 0.9$ [tarefas/seg]
- $\mu = 1$ [tarefas/seg]
- Buffer = 3 tarefas

A figura VI.2 mostra o número médio de tarefas no sistema versus o tamanho do intervalo entre atualizações de estado do gerente (T).

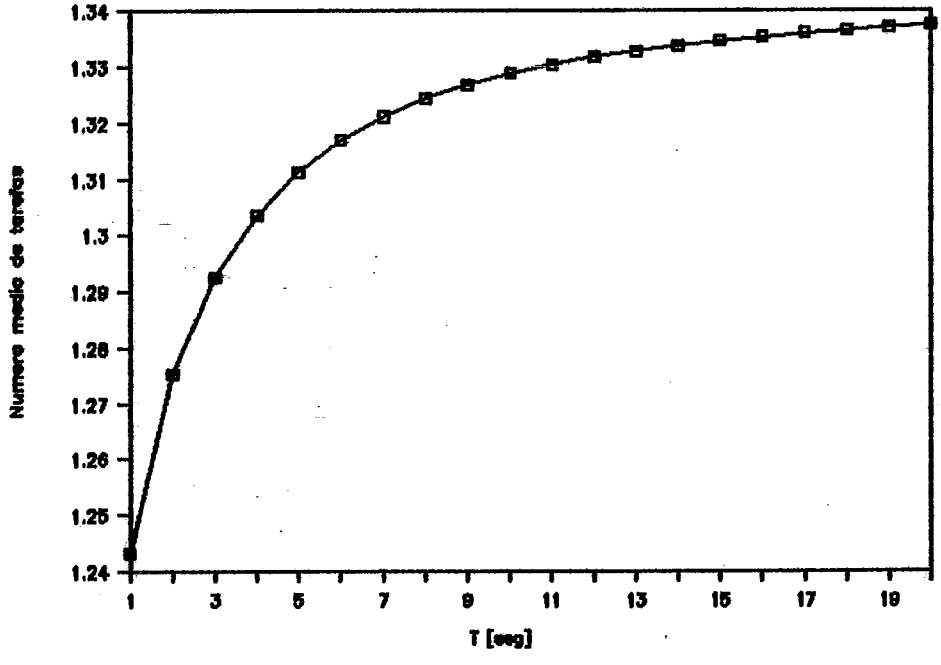

Figura VI.2: Número Médio de Tarefas no Sistema vs T

Vemos no gráfico da figura VI.2 que o número médio de tarefas no sistema aumenta com o aumento do tamanho do intervalo. Isto de deve ao fato de que, para intervalos entre atualizações pequenos, uma tarefa que chega encontrará, com alta probabilidade, o servidor desocupado, fazendo com que seja atendida com maior rapidez. Já para intervalos maiores, tarefas serão enviadas para servidores já saturados, incrementando o número de usuários.

VI.4 Tempo Médio de Resposta

Os parâmetros de entrada são:

- Modelo 1
- Regra 1, $p_1 = 1, p_2 = 0$
- $\lambda = 0.9$ [tarefas/seg]
- $\mu = 1$ [tarefas/seg]
- Buffer = 3 tarefas

A figura VI.3 mostra o tempo médio de resposta versus o tamanho do intervalo T. Pode-se ver na figura que o tempo de resposta é menor quando T=1 seg., pois o gerente sabe com maior precisão para onde enviar as tarefas de maneira tal que cheguem a um servidor desocupado ou com poucas tarefas na fila e, desta forma, sejam processadas em menor tempo. Quando T é grande o gerente tem informação desatualizada, por isso envia as tarefas para filas sobrecarregadas fazendo-as esperar mais.

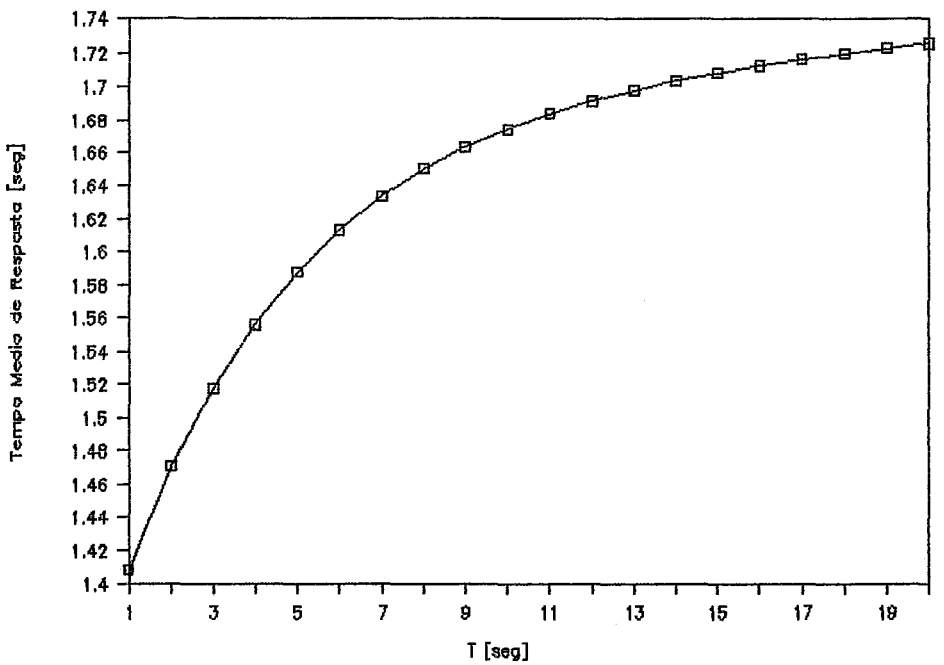

Figura VI.3: Tempo médio de resposta vs T

A seguir, fornecem-se observações importantes, com relação ao mecanismo de chegadas de tarefas aos centros. Estas observações, permitirão explicar, posteriormente, alguns fenômenos encontrados nos experimentos realizados.

Para um sistema com 2 centros e utilizando-se da regra 1 com $p_1 = 1$, existem duas situações que acontecem simultâneamente em um mesmo intervalo de tamanho T: a primeira que poderia ser chamada de não chegadas ao servidor 1, e outra de chegadas ao servidor 2, (ou viceversa). A situação de não chegada a um determinado servidor, acontece quando, no início do ciclo, o tamanho da fila deste servidor é maior que o tamanho da fila do outro servidor. Neste caso a regra de balanceamento estabelece que o gerente não envie tarefas para o mencionado centro. Por exemplo, a figura VI.4 representa um possível cenário para a situação de chegadas ao centro 1 (e não chegadas ao centro 2), no caso em que o estado do sistema ao início do ciclo é $(n1, n2)$, com $n1 < n2$.

Outro fenômeno interessante de destacar é que no caso em que ambos servidores são iguais, devido à simetria da regra de balanceamento, a taxa média de chegada a ambos os servidores é $\lambda/2$. Note-se, porém, que a distribuição dos tempos entre chegadas não é exponencial, já que este tempo depende do estado em que se encontra o sistema. Por exemplo, durante os ciclos de chegada de um determinado centro, os tempos entre chegadas das tarefas correspondem a uma v.a. exponencial com taxa média λ, porém nos ciclos de não chegadas a taxa média de chegadas de tarefas a este centro é 0. A única situação em que os tempos entre chegadas de usuários a um determinado centro correspondem a v.a's exponenciais, acontece no caso em que $T \rightarrow \infty$ e o estado inicial é tal que $n1 <> n2$, já que neste caso a regra de balanceamento estabelece que todos as tarefas são enviadas a um único centro.

$n_1 < n_2$

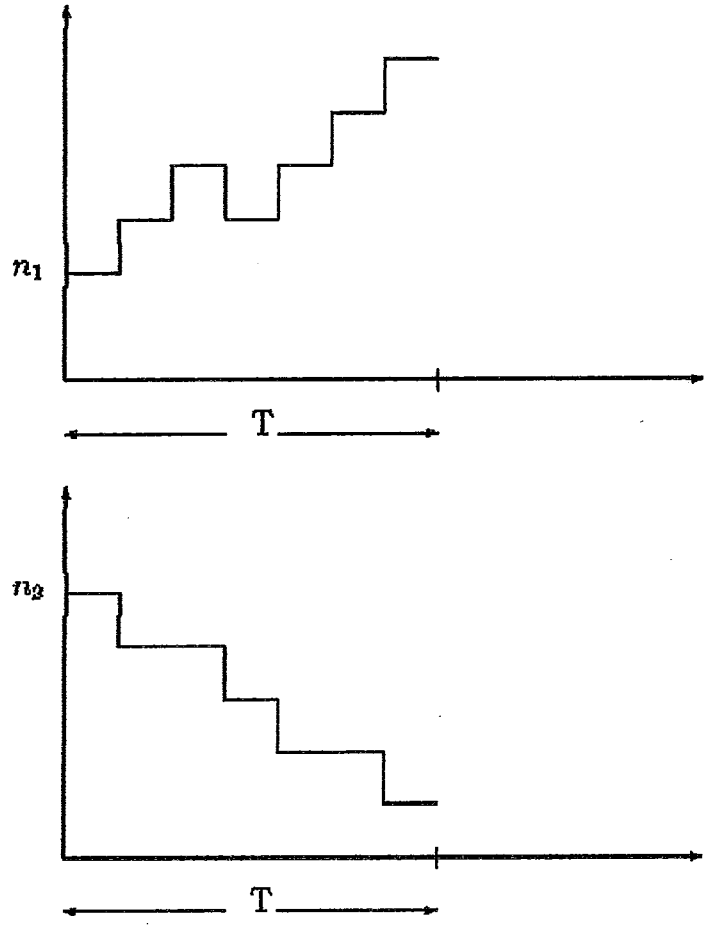

Figura VI.4: Situações de não chegadas e chegadas para T

VI.5 Sensibilidade das medidas com relação à taxa de chegada de tarefas no sistema

Parâmetros usados

- Modelo 1
- Regra 1, $p_1 = 1, p_2 = 0$
- buffer dos servidores = 5 [tarefas]
- taxa de serviço (μ) = 1 [tarefas / seg]
- taxa de chegada ao sistema (λ) variando entre 0.5,..,0.9, 1, 1.25, 1.5 e 2 [tarefas / seg].

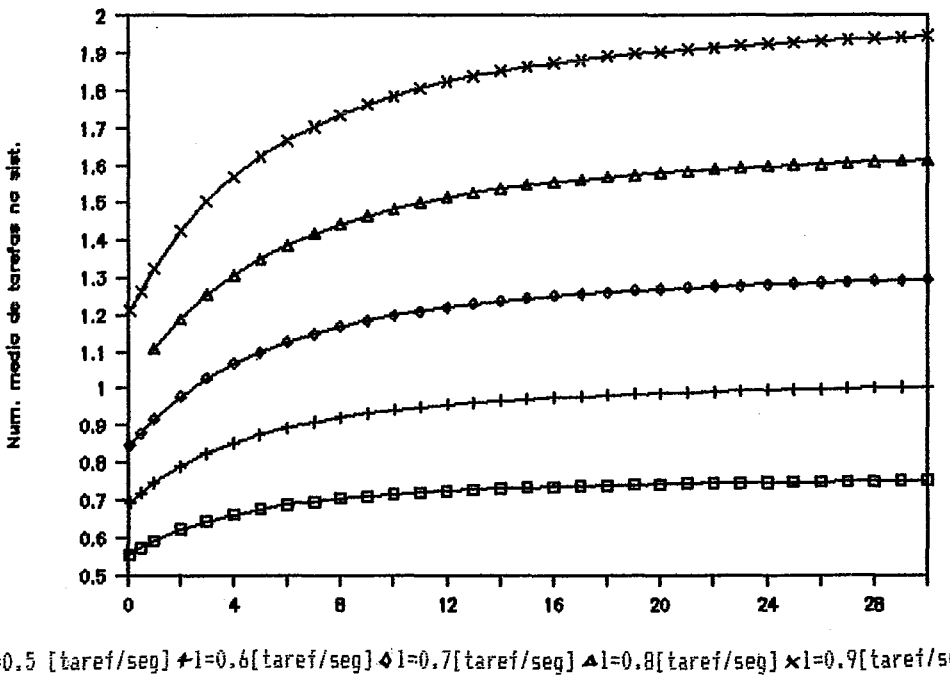

Figura VI.5: Número médio de tarefas no sistema vs T para diferentes taxas de chegada

Os gráficos das figuras VI.5 e VI.6 mostram o número médio de tarefas no sistema versus T para diferentes taxas de chegadas de tarefas ao sistema.

Para explicar de forma intuitiva os resultados das figuras VI.6 e VI.8 será usado um modelo aproximado do sistema estudado.

Para analisar o tempo de resposta de um determinado centro, supõe-se que o sistema está saturado e portanto $R_c = n_c/\mu_c$, onde μ_c é a taxa média de

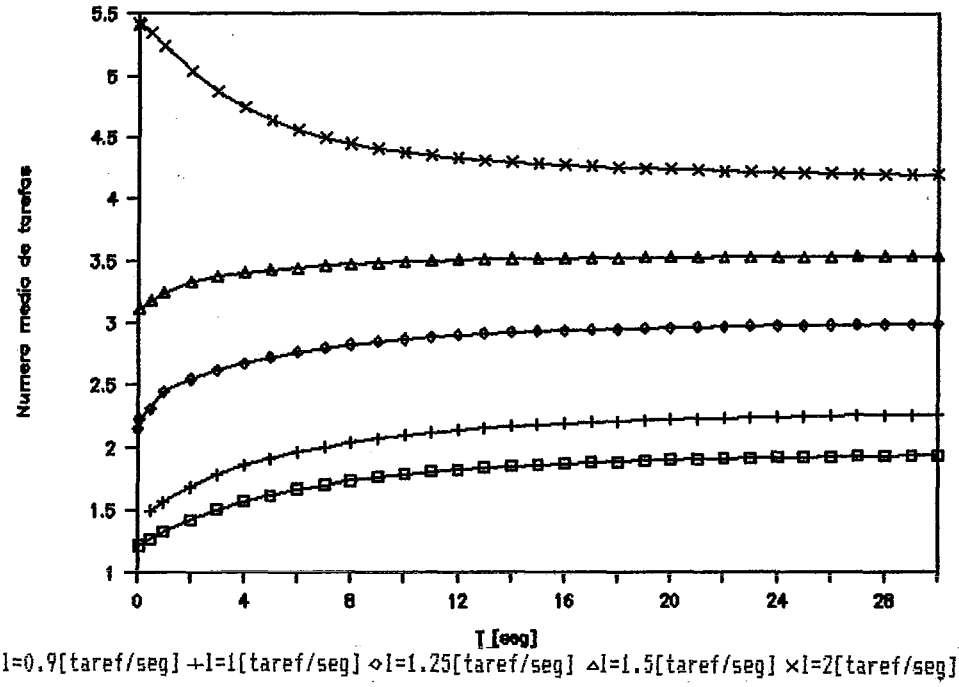

Figura VI.6: Número médio de tarefas no sistema vs T para diferentes taxas de chegada

serviço do centro e n_c é o número médio de tarefas neste centro. Adicionalmente se usará o fato que, para uma fila M/M/1/K , o n_c está dado pela seguinte equação (ver equação 5.2.41 de [2]):

$$n_c = \begin{cases} \frac{\rho_c}{1-\rho_c} - \frac{(K+1)\rho_c^{K+1}}{1-\rho_c^{K+1}} & , \rho_c \neq 1 \\ \frac{K}{2} & , \rho_c = 1 \end{cases} \qquad \text{(VI.1)}$$

Onde $\rho_c = \lambda_c/\mu_c$, e λ_c corresponde à taxa média de chegada de tarefas ao centro 'c'.

A seguir a análise da figura VI.8 será dividido em 4 casos:

- caso: T=0

  Neste caso tem-se que $\lambda_c = \lambda/2 = 1$, e $\rho = 1$. Tal como foi discutido na metodologia de análise, em primeiro lugar se supõe que os tempos entre chegadas são v.a's exponenciais com parâmetro $\lambda_c = 1$. Neste caso a equação estabelece que:

  $$n_c \approx \frac{K}{2} = 2.5$$

  Portanto, o tempo de resposta do centro é :

  $$R_c \approx \frac{2.5}{1} = 2.5$$

Do ponto de vista do sistema, tem-se que 'N', o número médio de usuários, devido à simetria do sistema, corresponde a 2 vezes n_c, portanto $N = 5$. O tempo de resposta do sistema R corresponde a R_c, e portanto $R \approx R_c \approx 2.5$. A diferença obtida com o caso exato (ver figura VI.8) $R_c = 3$, pode ser entendida da seguinte forma. No caso em que T=0, o gerente tende a usar de melhor forma os buffers, uma vez que tende a balancear a carga da melhor maneira possível. Portanto, tarefas que no modelo aproximado seriam descartadas do sistema, quando chegam a um centro com buffer cheio, no modelo exato encontram espaço no buffer e ficam em fila. Este fenômeno faz com que o número médio de tarefas em um centro, para o caso exato, seja maior que para o modelo aproximado (ver figura VI.6). Isto faz com que o tempo de resposta seja maior, e portanto explica a diferença entre valores observados e calculados em forma aproximada.

- Caso 2: $T \rightarrow \infty$

  Neste caso, desprezando o transiente inicial, o sistema se comporta como consistindo somente de um centro com $\mu = 1$ [tarefas/seg], $\lambda = 2$ [tarefas/seg] (portanto, $\rho = 2$) e buffer=5 [tarefas]. Da equação VI.1 se tem que:

$$N = n_c = \frac{2}{1-2} - \frac{6 * 2^6}{1-2^6} \approx 4.1$$

  Na figura VI.6 pode ser visto que o número médio tende a este valor.

  Portanto, $R = N/\mu \approx 4.1$.

- Caso 3: $0 < T < n_c/\mu$

  Neste caso, tem-se que n_c corresponde ao valor de n_c para T=0 [seg], modificado pelas situações de chegada e não chegada pelas quais passa o centro. Em primeiro, lugar deve ser notado que devido à simetria entre os dois centros, a probabilidade que um centro esteja em uma situação de chegada é igual a que ele esteja numa situação de não chegada.

  No caso em que o centro está na situação de não chegada, as tarefas que estejam na fila são atendidos com taxa $\mu = 1$ [tarefas / seg] e, portanto o tamanho da fila diminue com taxa média $\mu = 1$ [tarefas /seg]. Note-se que o valor de T para o caso sob análise (T=3 [seg]) foi escolhido para que com alta probabilidade o buffer não consiga esvaziar-se completamente nesta fase. Por outro lado, quando o centro está na situação de chegada, durante este ciclo, tem-se que $\lambda = 2$ [tarefas /seg], $\mu_c = 1$ [tarefas /seg] e, portanto, neste período tarefas tendem a se acumular na fila do centro com taxa média igual a 1 (que corresponde a $\lambda - \mu$ [tarefas /seg]). Devido à limitação de buffer o número máximo de tarefas nesta fila será o tamanho do buffer.

  As situações explicadas, anteriormente, fazem com que o número médio de tarefas na fila do centro sem chegadas diminua progressivamente a partir de $n_c(0)$ (número de tarefas com T=0), quando se aumenta T, ou seja o centro terá um menor número de tarefas em relação ao n'umero de tarefas com tamanho de intervalo zero. Portanto, o tempo de resposta diminui desde $R_c(T = 0)$ até $R_c(T = n_c(0)/\mu)$.

Para a fila com chegadas, devido à limitação de buffer, existirá nela no máximo o número de tarefas indicado por este tamanho. Como conseqüência destes dois fatos o número de tarefas no sistema diminue com T, em relação a T=0 [seg].

Por último, é interessante fazer notar que nem sempre em todos os ciclos de operação do sistema, os centros estão nas situações de chegadas e não chegadas explicadas anteriormente, já que nos casos em que o ciclo é iniciado em um estado balanceado ($n_1 = n_2$) [tarefas], o gerente envia tarefas com taxa média $\lambda/2$ a ambos centros. Portanto, a influência do fenômeno de 'chegadas-não chegadas' na diminuição de n_c (com relação a $n_c(0)$) depende da probabilidade que o sistema comece num estado balanceado e, como em geral, esta probabilidade não é muito grande, a diminuição de n_c (e portanto de R_c) não é muito significativa.

- Caso 4: $T > n_c/\mu$

  Neste caso, podemos identificar duas fases no ciclo. A primeira corresponde à situação em que o centro que está em situação de não chegadas ainda tem tarefas na sua fila. Note-se que em valor médio nesta fase corresponde aproximadamente ao período n_c/μ_c. A segunda fase corresponde à situação em que o centro ao qual não entram tarefas permanece vazio. Com relação ao valor de R_c, se tem que a primeira fase influencia o valor desta medida da forma que foi explicado no ponto acima. Porém, na segunda fase o valor de R_c é influenciado somente pelo centro que está na situação de chegadas, já que o outro centro está vazio e portanto, não existem tarefas que devam ser consideradas para o cálculo de R.

  A situação explicada anteriormente faz com que o valor de R vá variando desde $R(\frac{n_c}{\mu_c})$ até $R(\infty)$ a medida que aumenta T, tal como mostrado na figura VI.8.

  Por último, tal como foi dito no caso anterior, como nem todos os ciclos correspondem a situações de 'chegadas-não chegadas' nos centros, o efeito da situação explicada acima é tal que a velocidade em que R tende a $R(\infty)$ não é muito grande, como pode ser visto na figura VI.8.

  Com a análise feita acima tentou-se explicar o fenômeno não intuitivo da diminuição de R com o aumento de T, utilizando um raciocinio aproximado sobre o comportamento do sistema. Entretanto, só o modelo exato pode comprovar o fenômeno.

Em resumo, a aparente anomalia ocorre pelo fato dos "buffers" serem limitados e que as tarefas rejeitadas por falta de espaço não são consideradas para o cálculo do tempo médio de resposta.

Os gráficos das figuras VI.7 e VI.8 mostram o tempo médio de resposta versus o tempo entre atualização da informação de estado (T), para diferentes taxas de chegada de tarefas ao sistema. Pode ser visto no gráfico que a medida que a taxa de chegada aumenta, também aumenta o tempo de resposta das tarefas no sistema, o qual é um resultado obvio. É possível notar também, que para quase todas as taxas de chegada de tarefas ao sistema, o tempo de resposta é menor para

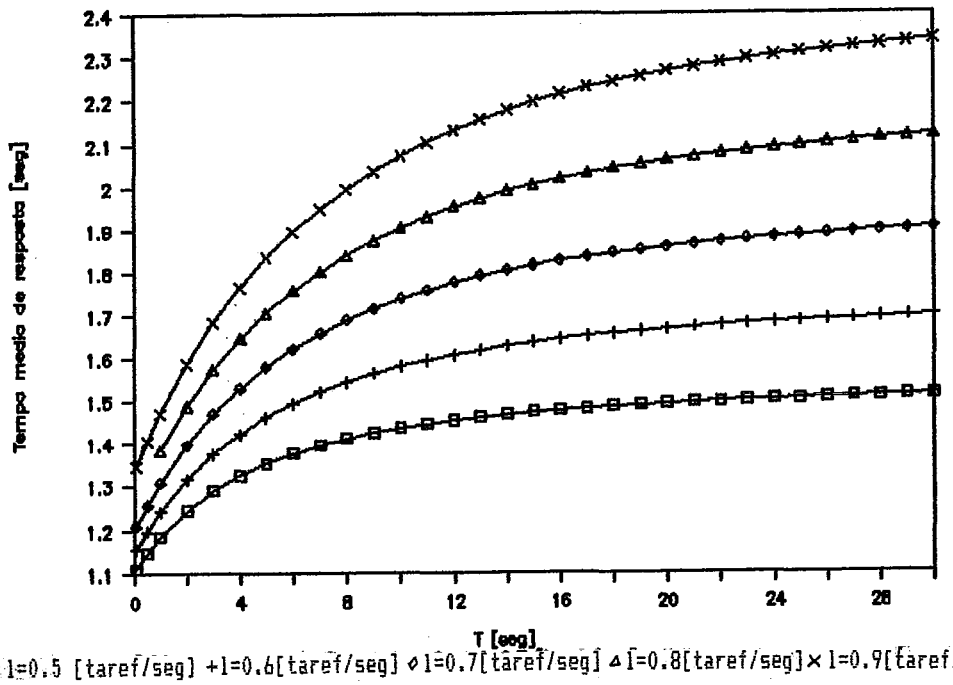

Figura VI.7: Tempo médio de resposta vs T para taxas de chegada entre 0.5 e 0.9

T=0 seg (exceto para $\lambda = 2$ [tarefas /seg], que tem o mínimo em T=3 [seg]) e cresce para tamanhos de intervalo grandes devido ao fenômeno explicado acima.

Em geral, acontece que o tempo de resposta aumenta ao aumentar o T, uma vez que os intervalos maiores fazem que o gerente trabalhe com informação envelhecida e envie tarefas para filas que estão carregadas fazendo que elas fiquem sobrecarregadas. Quando o tamanho de T tende a zero as filas tendem a estar mais vazias porque o gerente tem uma informação mais recente do estado do sistema e pode escalonar as tarefas para o servidor ocioso e não sobrecarregar a fila que já está cheia.

Alguns exemplos: se a taxa de chegada varia de $\lambda = 0.9$ [tarefas /seg] a $\lambda = 1$ [tarefas /seg], o tempo de resposta aumenta aproximadamente 6.8% para T=1 [seg], e este aumento é crescente para T maiores. É assim que para T=30 [seg] o aumento no tempo de resposta é aproximadamente 9.2% Uma variação da taxa de chegada entre $\lambda = 0.9$ [tarefas /seg] a $\lambda = 2$ [tarefas /seg], faz aumentar o tempo de resposta em 102% aproximadamente. Porém, para este caso o tempo de resposta não aumenta tanto como no caso anterior para T grandes como no caso anterior, o aumenta fica em em aprox. 65% .

As figuras VI.9 e VI.10 mostram a vazão média do sistema versus T para as diferentes taxas de chegadas de tarefas ao sistema.

Para todas as taxas de chegada é possível verificar que a vazão decai a medida que T aumenta.

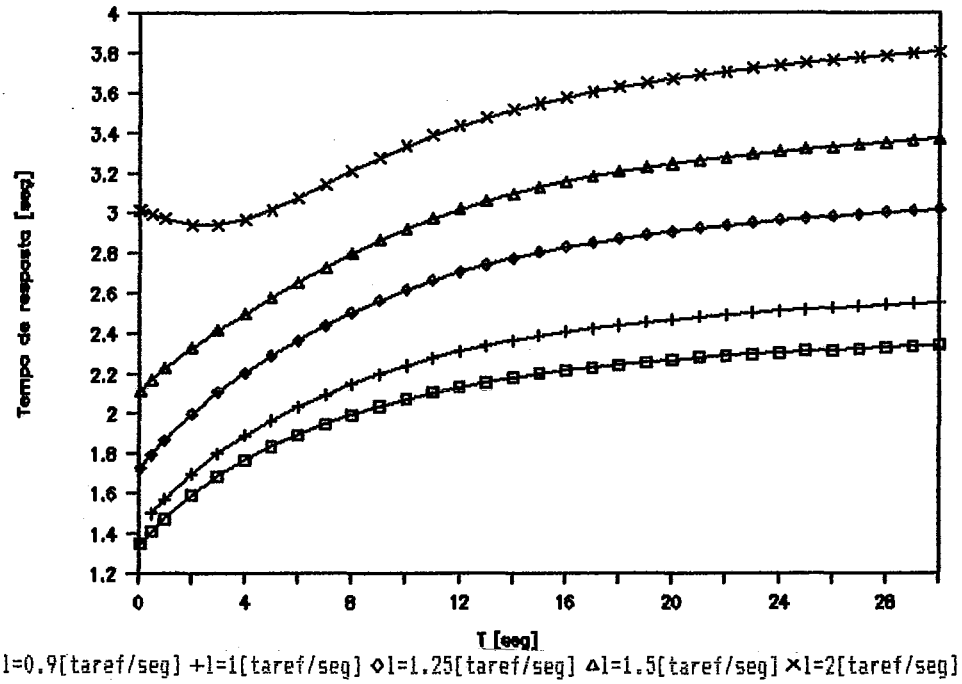

Figura VI.8: Tempo médio de resposta vs T para taxas de chegadas entre 0.9 e 2

A vazão alcança seu valor máximo para T tendendo a zero. Por exemplo, para T=0 seg. e taxas de chegadas $\lambda = 0.5 \ldots 1$ [tarefas /seg], a vazão máxima é praticamente igual à taxa de chegada. O comportamento é aproximadamente igual a um sistema sem limitação de buffer. Já para T=2 [seg] a vazão começa a diminuir segundo o fenômeno descrito na seção de vazão média.

Para taxas de chegadas maiores (1.25, 1.5, 2 [tarefas /seg]) a vazão alcança seu máximo valor para T=0 [seg], embora não seja igual à taxa de chegada, pois existe limitação de buffer.

Para T grande a vazão diminui devido a que o gerente trabalha com informação envelhecida, e manda tarefas para filas que estão com buffer saturado, fazendo com isto que as tarefas se percam e a vazão diminua. Esta situação se acentua para taxas de chegada grandes, por exemplo $\lambda = 2$ [tarefas /seg] se produz a maior diminuição da vazão, que é de aproximadamente 62.5% entre T=1 e T=30 [seg]. A figura VI.12 mostra que para T=3 [seg] e $\lambda = 2$ [tarefas /seg], a fila permanece saturada 24.5 % do tempo entre atualizações.

Pode-se ver na figura VI.10 que, quando a taxa de chegada diminui, o valor de T que fornece uma diminuiç ao de 10% ou mais no valor da vaz ao para T=0, aumenta. Por exemplo, para $\lambda = 2$[tarefas/ seg] são os $T > 3$[seg]. para $\lambda = 1.5$ [tarefas/seg] está dado pelos $T > 6$ [seg], para $\lambda = 1$[tarefas /seg] são os $T > 16$ [seg]. Para valores menores da taxa de chegada a diminuiç ao não é tão significativa. Por isto, os valores de T são muito grandes (ver figura VI.9).

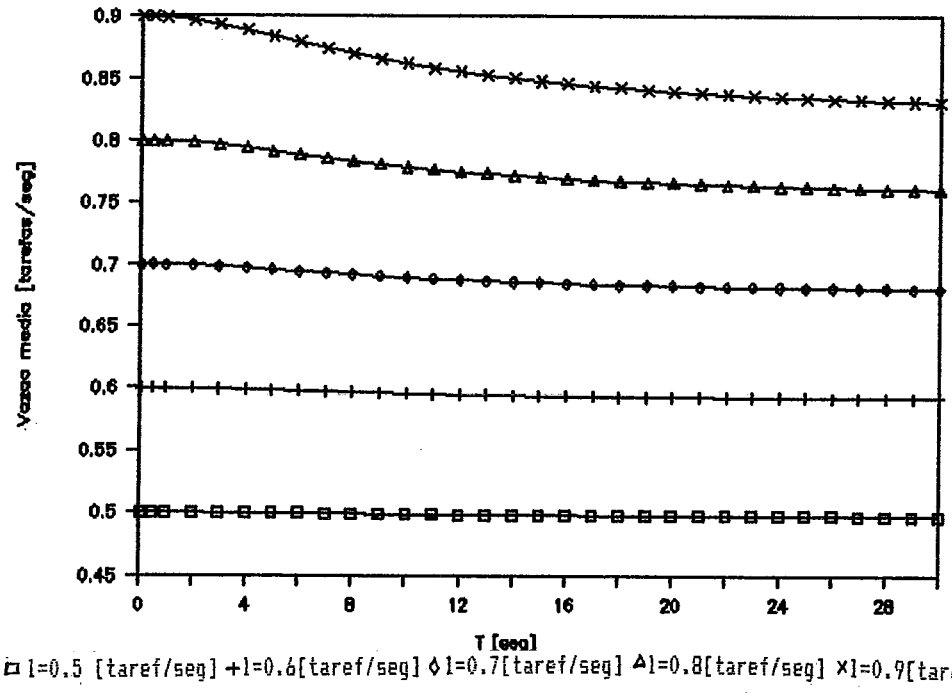

Figura VI.9: Vazão média no sistema vs T para taxas de chegada entre 0.5 e 0.9

As figuras VI.11 e VI.12 mostram o percentual do tempo que a fila permanece saturada no intervalo entre atualizações da informação de estado. Note que o percentual de tempo que a fila permanece saturada no intervalo T não é o mesmo que a probabilidade de bloqueio. Neste caso se está medindo quanto tempo uma fila ou a outra permanecem com seu buffer cheio neste intervalo. Por ser o sistema simétrico são mostrados somente os gráficos correspondentes à fila de um dos servidores.

Pode ser visto na figura VI.11 que o mesmo tipo de fenômeno que ocorre para o tempo de resposta ocorre com o percentual de fila cheia. Isto acontece devido a motivos semelhantes aos explicados para R.

Para justificar, qualitativamente, este tipo de gráfico se usará a seguinte metodología de análise. Através desta metodología tenta-se explicar o fenômeno não intuitivo da diminuição de percentual de fila cheia para $\lambda = 2$ [tarefas /seg].

Para T=0 [seg] vamos supor que:

$$\lambda_c^* = \frac{\lambda \mu_c}{\sum_i \mu_i} \qquad \text{(VI.2)}$$

Mesmo λ_c^* não sendo o valor exato, é uma boa aproximação da taxa de chegada de jobs ao centro. O valor de λ_c^* nos permitirá calcular P_c que está dado pela equação 5.2.8 de [2], que estabelece que para uma fila, a probabilidade de bloqueio P_c está dada por:

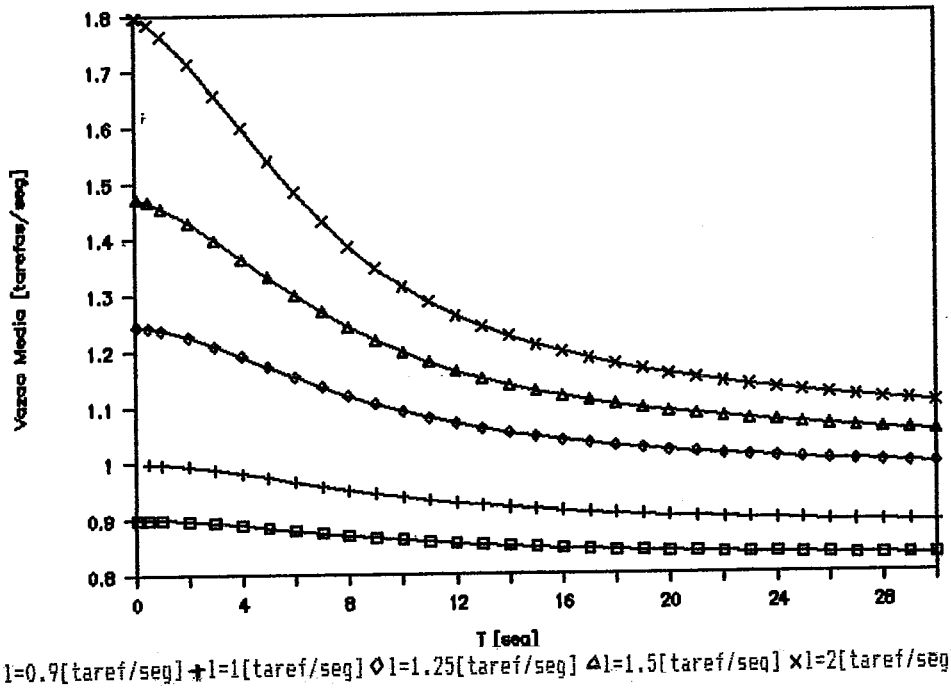

Figura VI.10: Vazão média no sistema vs T para taxas de chegada entre 0.9 e 2

$$P_c = \begin{cases} \frac{(1-\rho_c)\rho_c^K}{1-\rho_c^{K+1}} & , \rho \neq 1 \\ \frac{1}{K+1} & , \rho = 1 \end{cases} \qquad \text{(VI.3)}$$

onde K é o tamanho do buffer no centro.

Para T muito grandes, temos que, evidentemente, cada centro fica alternadamente em um ciclo com chegadas e em um ciclo de não chegadas. Para os ciclos de chegadas tem-se que $\lambda_c = \lambda$, (pois o gerente está enviando todo o fluxo de chegada para ele), o qual permite calcular P_c, que corresponde à probabilidade que o centro 'c' esteja cheio nos ciclos de chegadas. Para calcular P_c^*, que corresponde à probabilidade do centro 'c' estar cheio, considerando ambos tipos de ciclo, como na metade do tempo o centro está em ciclos de chegadas e a outra metade do tempo em ciclos de não chegadas, temos que $P_c^* = P_c/2$.

Por outro lado, para valores de T pequenos, devido ao mesmo tipo de análise para o caso do cálculo de R_c, o valor de P_c é afetado pelos ciclos de chegadas e não chegadas. Tal como foi discutido anteriormente muitos dos ciclos partem em estados balanceados e, portanto, não são do tipo 'chegadas-não chegadas'. Isto implica que o efeito dos ciclos 'chegadas-não chegadas' é limitado e, portanto, não tem grande influência sobre o valor de P_c^* obtido. Ou seja se o ciclo de não chegadas tivesse muita influência a fila se esvaziaria rapidamente. Logicamente, que tal como no caso de R_c, o efeito dos ciclos de chegadas-não chegadas consiste em diminuir o valor de P_c com relação ao valor de $P_c(0)$.

Como exemplo analisa-se o caso de $\lambda = 2$ [tarefas/seg]

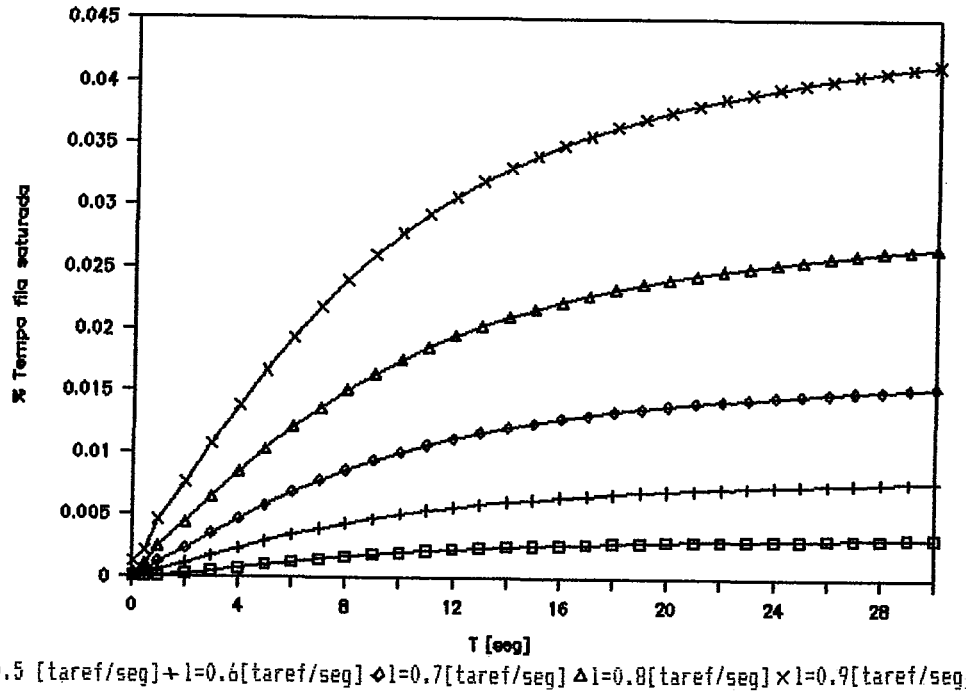

Figura VI.11: Percentual do tempo que a fila permanece saturada em T vs T para taxas de chegada entre 0.5 e 0.9 [tarefas /seg]

- Caso T=0

  Neste caso $\rho_c = 1$, isto implica $P_c \approx 0.17$, o que pode ser verificado na figura VI.12.

- Caso $T = \infty$ Nos ciclos de chegada temos que $\lambda_c = \lambda = 2$ [tarefas /seg], $\mu_c = 1$ [tarefas /seg], isto implica $\rho = 2$. De onde se obtem que $P_c \approx 0.5$, portanto $P_c^* \approx 0.25$. No gráfico da figura VI.12 se ve que a tendência para T grandes é atingir esse valor.

No gráfico da figura VI.11 se vê, para T=0 [seg] e para taxa de chegada variando entre 0.6 e 0.9 [tarefas /seg], o tempo que a fila permanece cheia no intervalo entre atualizações da informação de estado, é zero.

Na figura VI.12 para T=0 e $\lambda = 1$ [tarefas /seg], o percentual do tempo que a fila permanece saturada é aproximadamente zero. Já para o mesmo intervalo e taxa de chegada $\lambda = 1.5$ [tarefas /seg], o tempo que a fila permanece cheia é o 4% do tempo do intervalo T. Este tempo que a fila permanece cheia aumenta para T=30 [seg], onde para $\lambda = 0.9$ [tarefas /seg] é de 4% quando para $\lambda = 1.5$ [tarefas /seg] este percentual passa para 11.5%

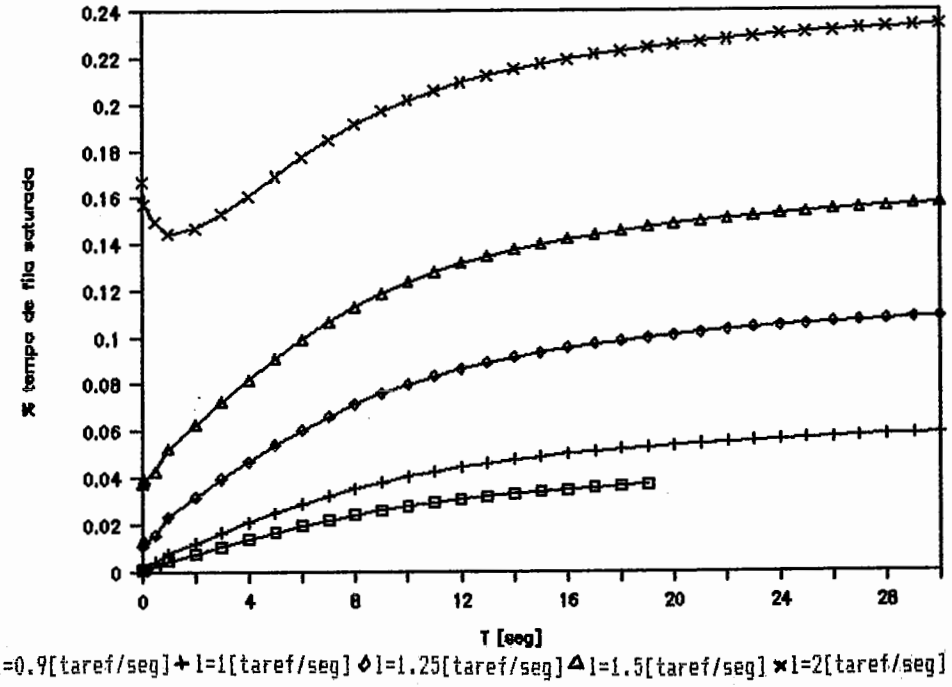

Figura VI.12: Percentual do tempo que a fila permanece saturada em T vs T para taxas de chegada entre 0.9 e 2 [tarefas /seg]

VI.6 Sensibilidade com relação às probabilidades de atribuição

Define-se como probabilidade de atribuição a probabilidade p_1 com a qual o gerente atribui as tarefas que chegam ao sistema à fila de menor comprimento na regra 1. Este experimento consiste em medir a sensibilidade com relação às probabilidades de atribuição especificadas nos parâmetros de entrada, os quais são mostrados a seguir:

- Modelo 1
- Regra de balanceamento 1; $p_1 = 0.5, .., 1$
- buffer dos servidores = 3 [tarefas]
- $\mu = 1$ [tarefas /seg]
- $\lambda = 0.67$ [tarefas /seg]

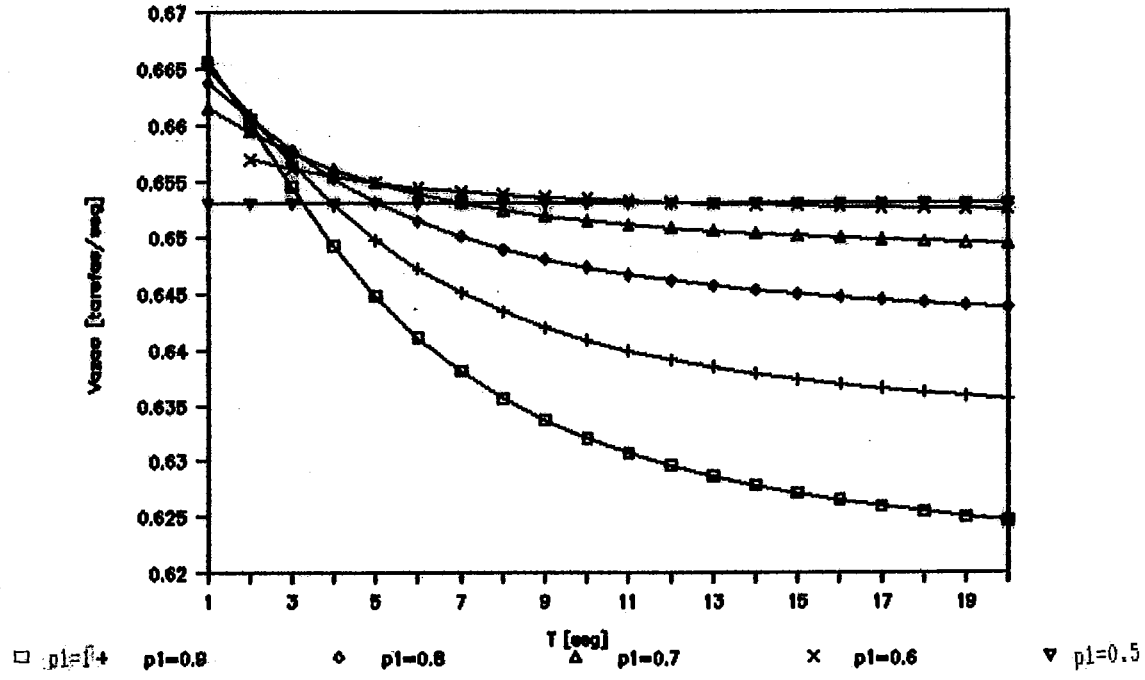

Figura VI.13: Vazão média no sistema vs T para diferentes valores da probabilidade de atribuição

Da figura VI.13 podemos observar o seguinte:

- Quando a probabilidade de atribuição $p_1 = 0.5$ para ambos servidores a vazão média do sistema se mantém constante em relação a T. Isto é equivalente

a calcular a vazão média em estado estacionário para duas filas M/M/1/3 independentes, com taxa de chegada de tarefas $\lambda/2$ [tarefas /seg] e $\mu = 1$ [tarefas /seg], e somá-las.

- Vê-se na figura VI.13 que quando p_1 cresce a vazão média cai rápidamente para T grande. Isto se deve ao fato que quando T aumenta, a atualização de estado do gerente se faz a intervalos de tempo cada vez maiores, produzindo o efeito de informação desatualizada já explicado. Se a probabilidade com que o gerente envia as tarefas à fila de maior tamanho tende a 1, existe a possibilidade que eles cheguem a uma fila cheia, enquanto a outra fila está vazia o que faz diminuir a vazão média. Esta possibilidade é menor quando o gerente atribui as tarefas que chegam à fila menor com probabilidade próximas a 0.5.

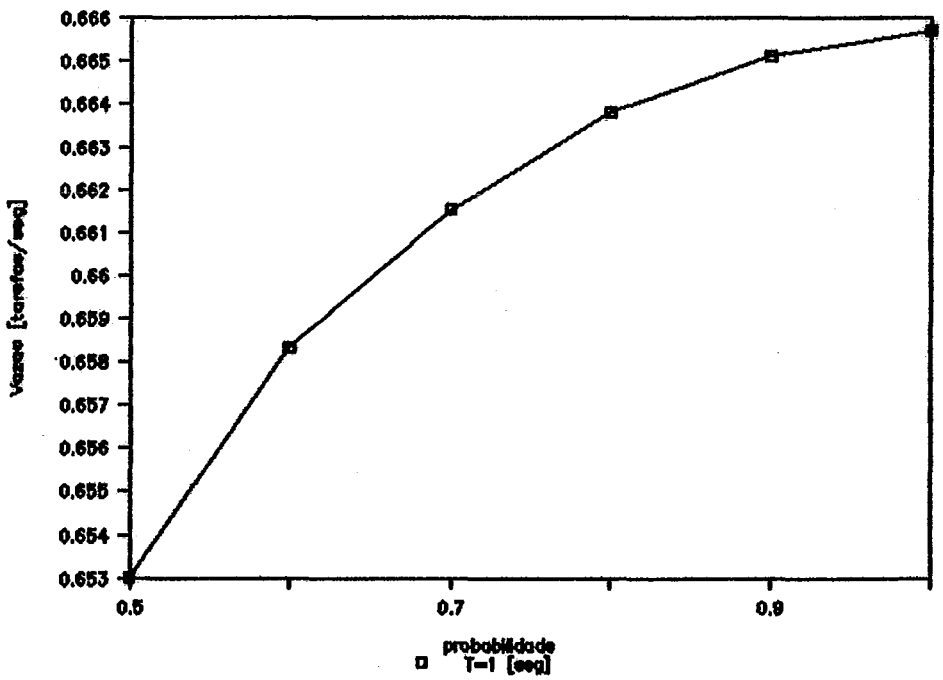

Figura VI.14: Vazão média no sistema vs probabilidade de atribuição para T=1 [seg]

- Existe uma probabilidade ótima p^*, que maximiza a vazão para os diferentes valores de T. No gráfico da figura VI.14 se observa que a probabilidade de atribuição 1 é a que fornece a maior vazão para T=1 [seg].

  No gráfico da fig VI.15 se observa que a probabilidade de atribuição = 0.6 é a que fornece a maior vazão para T=5 [seg] e T=8 [seg] (figura VI.16).

  Para T=12 [seg] e T=20 [seg] a probabilidade de atribuição que maximiza a vazão é 0.5, como mostrado nas figuras VI.17 e VI.18 respectivamente.

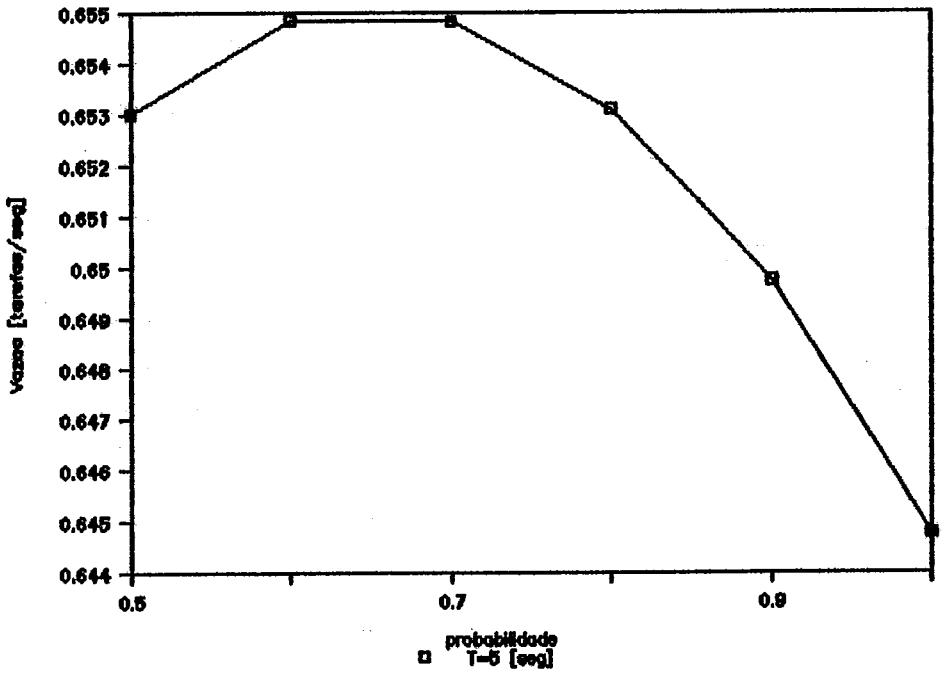

Figura VI.15: Vazão média no sistema vs probabilidade de atribuição para T=5 [seg]

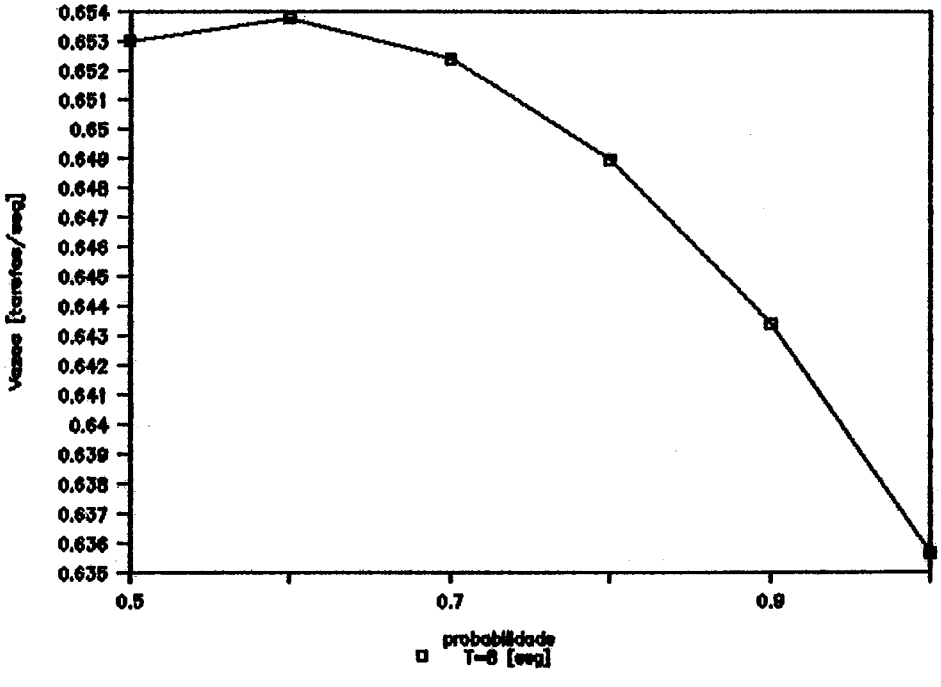

Figura VI.16: Vazão média no sistema vs probabilidade de atribuição para T=8 [seg]

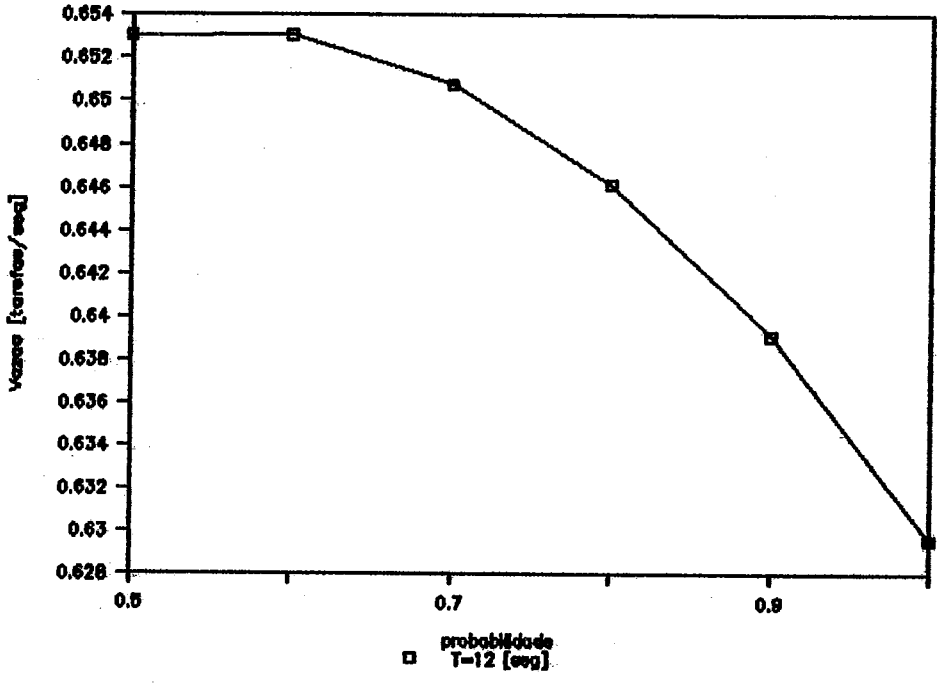

Figura VI.17: Vazão média no sistema vs probabilidade de atribuição para T=12 [seg]

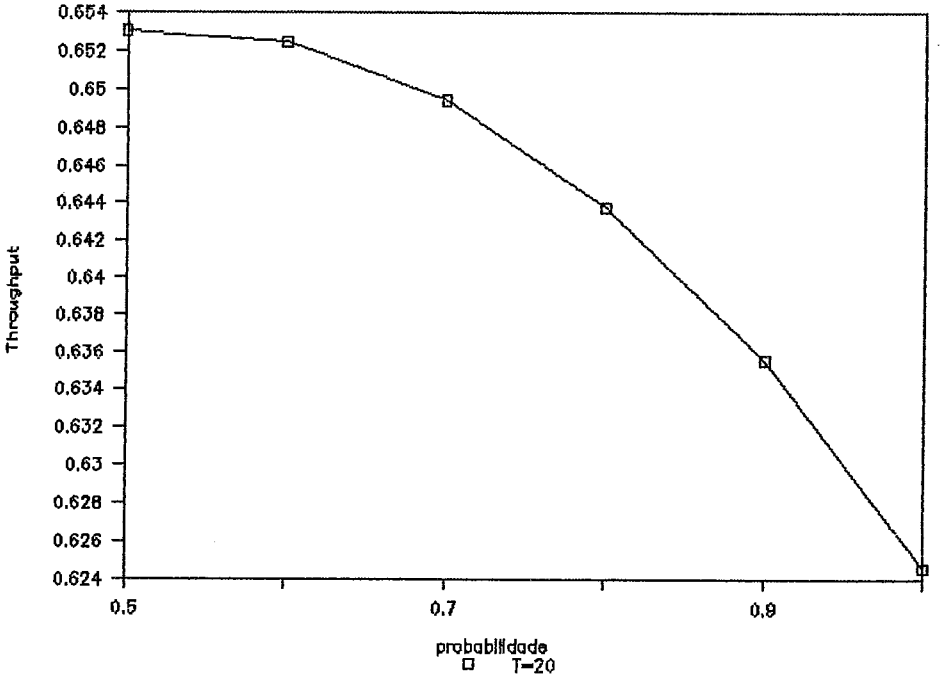

Figura VI.18: Vazão média no sistema vs probabilidade de atribuição para T=20 [seg]

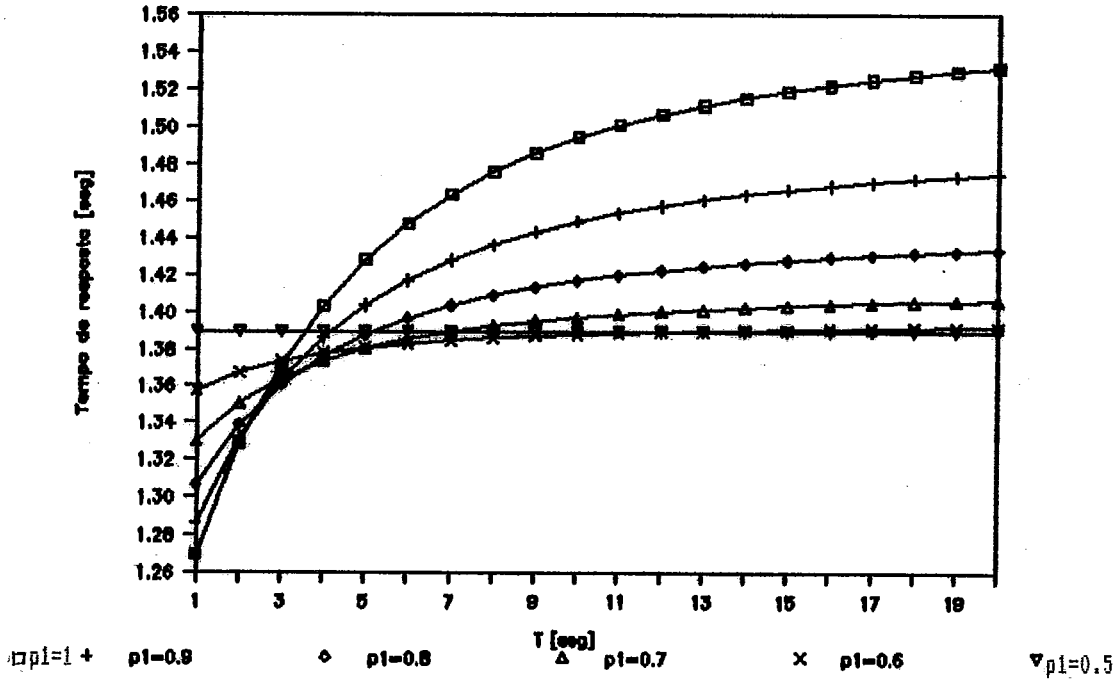

Figura VI.19: Tempo médio de resposta vs T

A figura VI.19 mostra o tempo de resposta médio versus o tamanho do intervalo entre atualizações de estado do gerente. Vê-se, no gráfico, que o tempo médio de resposta aumenta a medida que o T aumenta e esta tendência é maior quando as probabilidades de atribuição de tarefas tem maior diferença entre a probabilidade de atribuição à fila menor com a da outra fila.

O gráfico da figura VI.20 mostra o número médio versus o tamanho do intervalo entre atualizações (T).

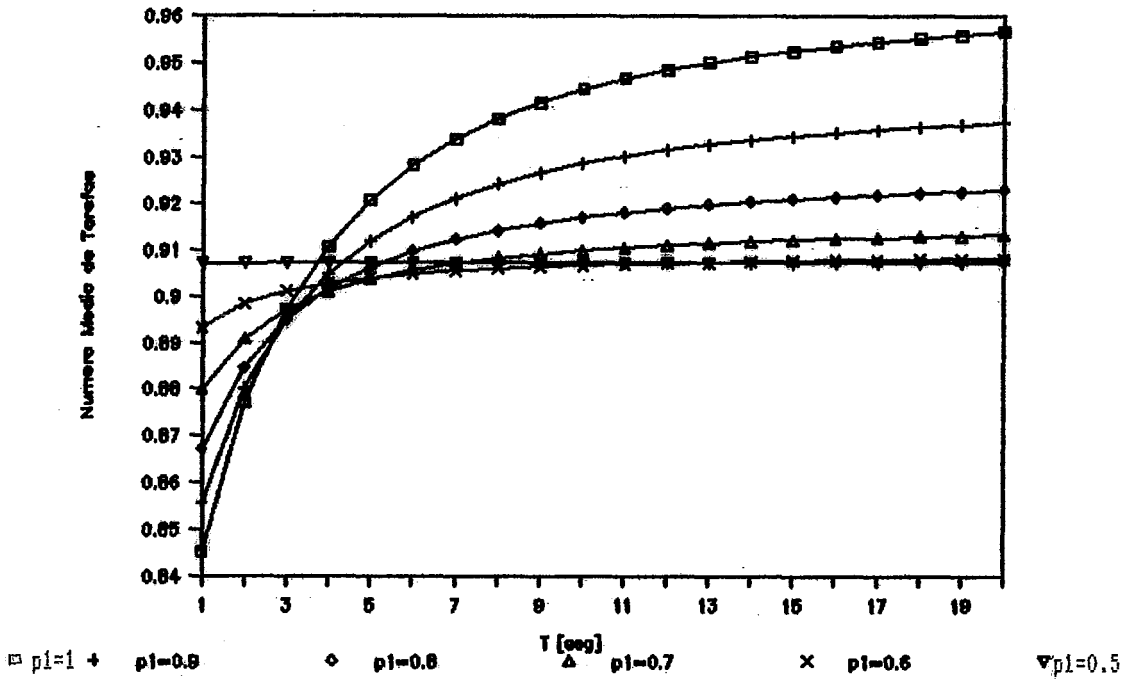

Figura VI.20: Número médio de tarefas no sistema vs T para diferentes valores das probabilidades de atribuição

Outro experimento de sensibilidade foi feito com os seguintes parâmetros:

- Modelo 1
- Regras de balanceamento 1 $p_1 = 0.5, .., 1$
- buffer dos servidores = 5
- $\mu = 1$ [tarefas /seg]
- $\lambda = 1$ [tarefas /seg]

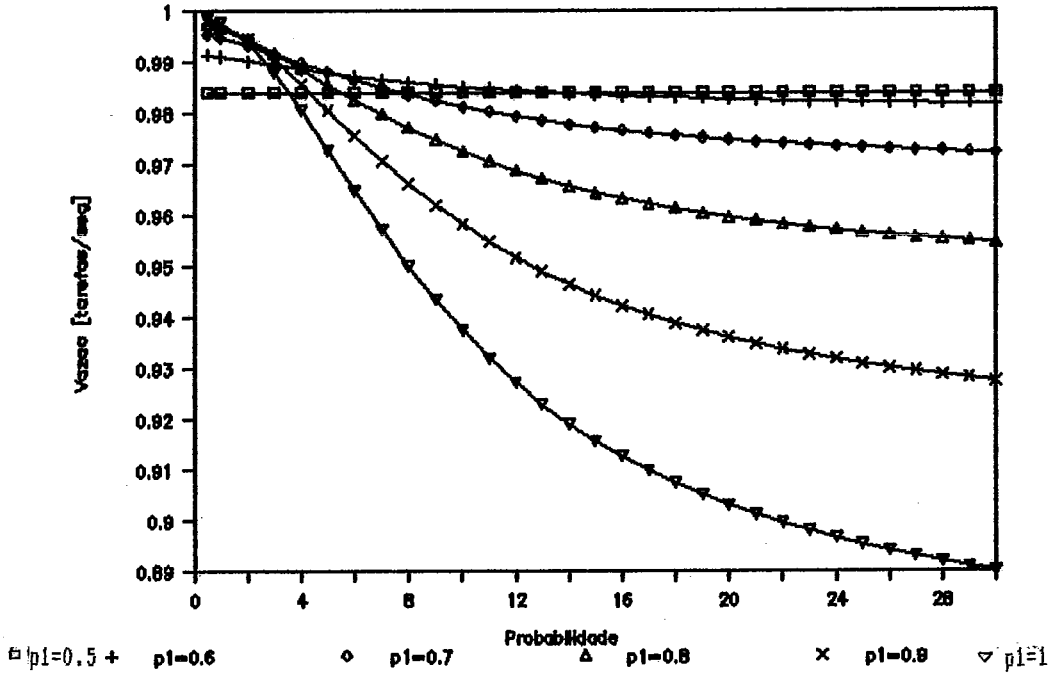

Figura VI.21: Vazão média no sistema vs T para diferentes valores da probabilidade de atribuição

O gráfico da figura VI.21 mostra a vazão média para as diferentes probabilidades de atribuição, em funcão do intervalo de tempo entre atualizacões da informação de estado. Pode ser notado que a vazão decai rápidamente a medida que aumenta T, em razão do gerente trabalhar com informação desatualizada.

A figura VI.22 mostra a vazão em função das probabilidades de atribuição para diferentes tamanhos do tempo entre atualizações da informação. A figura VI.22 indica que a probabilidade de atribuição que maximiza a vazão varia dependendo do T considerado.

As figuras VI.23, VI.24 e VI.25 mostram a vazão versus as probabilidades de atribuição para um T específico.

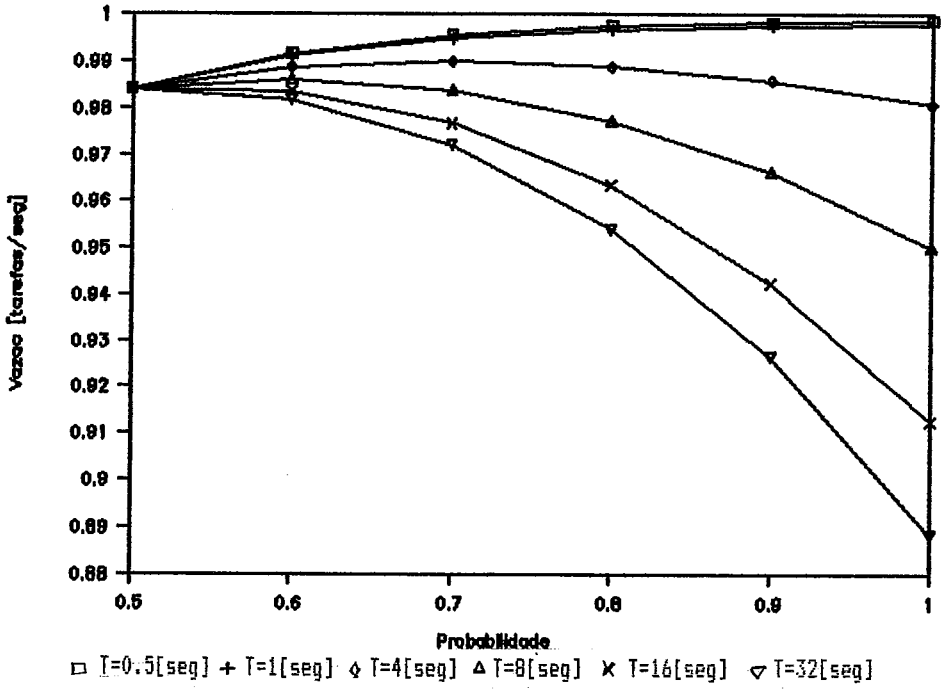

Figura VI.22: Vazão média no sistema vs probabilidade de atribuição para diferentes valores de T

Na figura VI.23 a vazão máxima é alcançada para $p_1 = 1$, pois como T é pequeno (a metade do tempo de serviço), o gerente sabe com maior exatidão qual o servidor com fila menor, e envia suas tarefas para ele. Ao enviar com probabilidade 1 está sendo maximizada a vazão. A mesma situação ocorre para T =1 [seg].

Para T=4 [seg] a vazão máxima é alcançada para a $p_1 = 0.7$, como pode ser visto na figura VI.24.

Na figura VI.25 para $T = 15$ [seg] a probabilidade = 0.5 é a que maximiza a vazão; isto também é verdadeiro para $T > 15$ [seg], pois para estes valores de T, a informação de estado está tão desatualizada que o melhor é mandar as tarefas em forma aleatória para ambas as filas.

Existe então, para um certo T, uma probabilidade ótima (p^*), que maximiza a vazão. A figura VI.26 mostra valores de p^* em função de T para o caso da vazão. Existe também um p^* para o caso do tempo de resposta. Neste exemplo este valor de p^* é igual ao obtido para a vazão.

Pode ser observado na figura VI.28 que o número médio de tarefas permanece constante com T, para $p_1 = 0.5$. Porém, o número médio de tarefas no sistema aumenta a medida que a probabilidade de atribuição aumenta e, também, aumenta quando aumenta T. Ambos os efeitos se devem à influência da aquisição de informação de estado do sistema por parte do gerente.

Quando a probabilidade de atribuição à fila é mais equilibrada, isto é, não se enviam todas as tarefas para uma mesma fila, mas se usam outras proporções, o efeito da informação desatualizada é atenuado diminuindo o número médio de tarefas no sistema, e conseqüentemente, diminuindo o tempo médio de resposta.

Na figura VI.29 é mostrado o gráfico do tempo que a fila permanece saturada no intervalo T. Pode-se notar que este tempo aumenta com o aumento de T e também aumenta para probabilidades de atribuição crescentes.

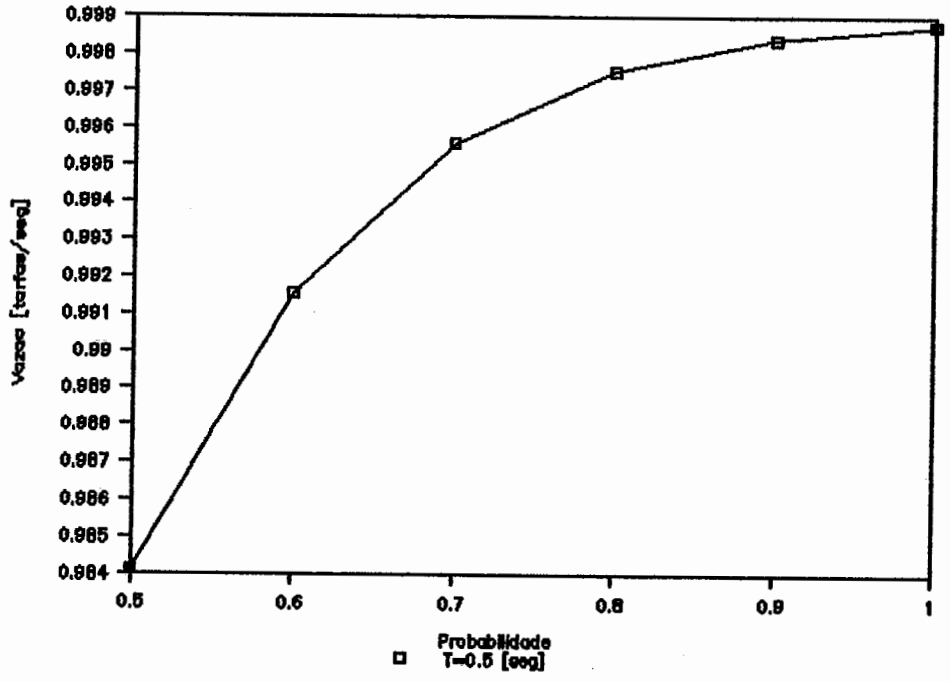

Figura VI.23: Vazão média no sistema vs probabilidade de atribuição para T=0.5 [seg]

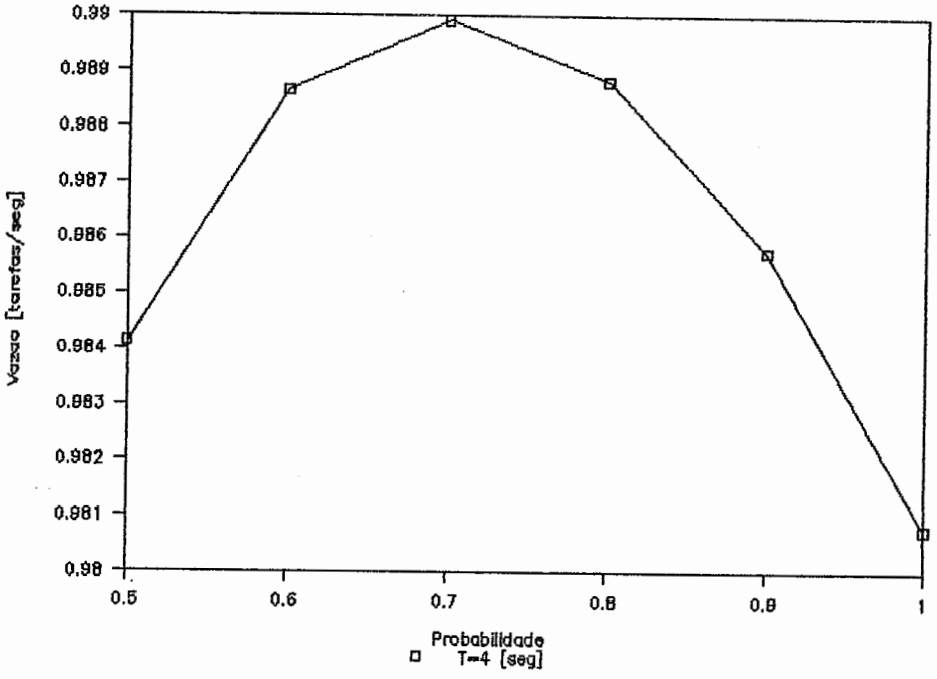

Figura VI.24: Vazão média no sistema vs probabilidades de atribuição para T=4 [seg]

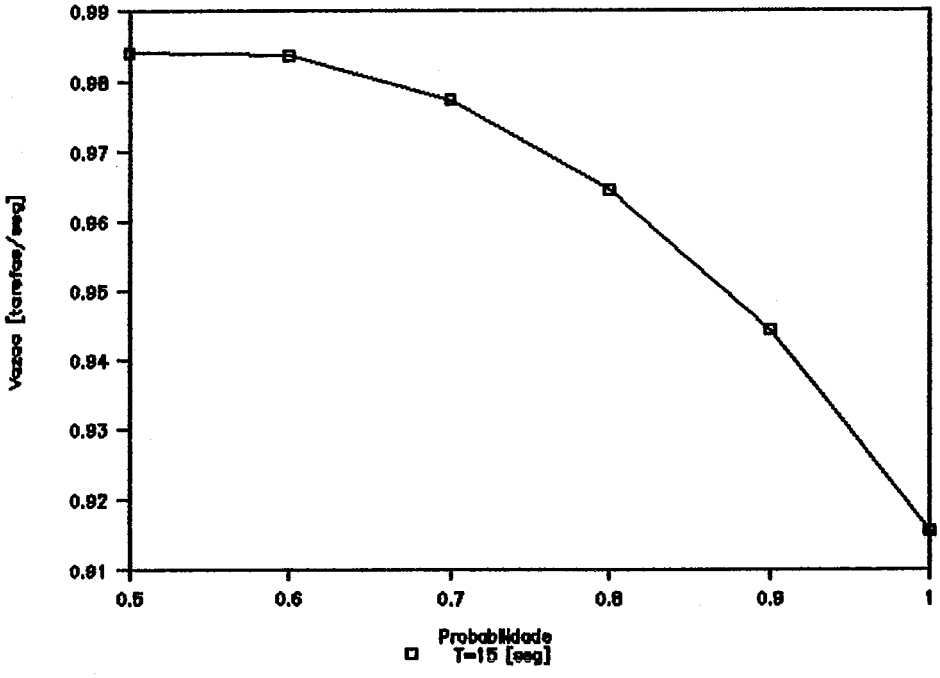

Figura VI.25: Vazão média no sistema vs probabilidade de atribuição para T=15 [seg]

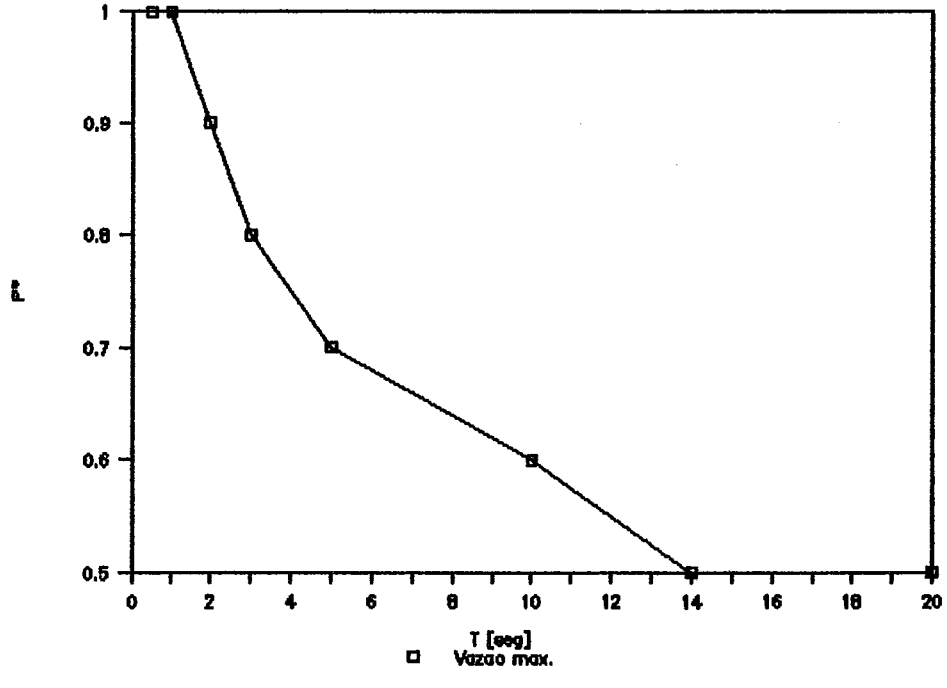

Figura VI.26: p^* vs T para a vazão

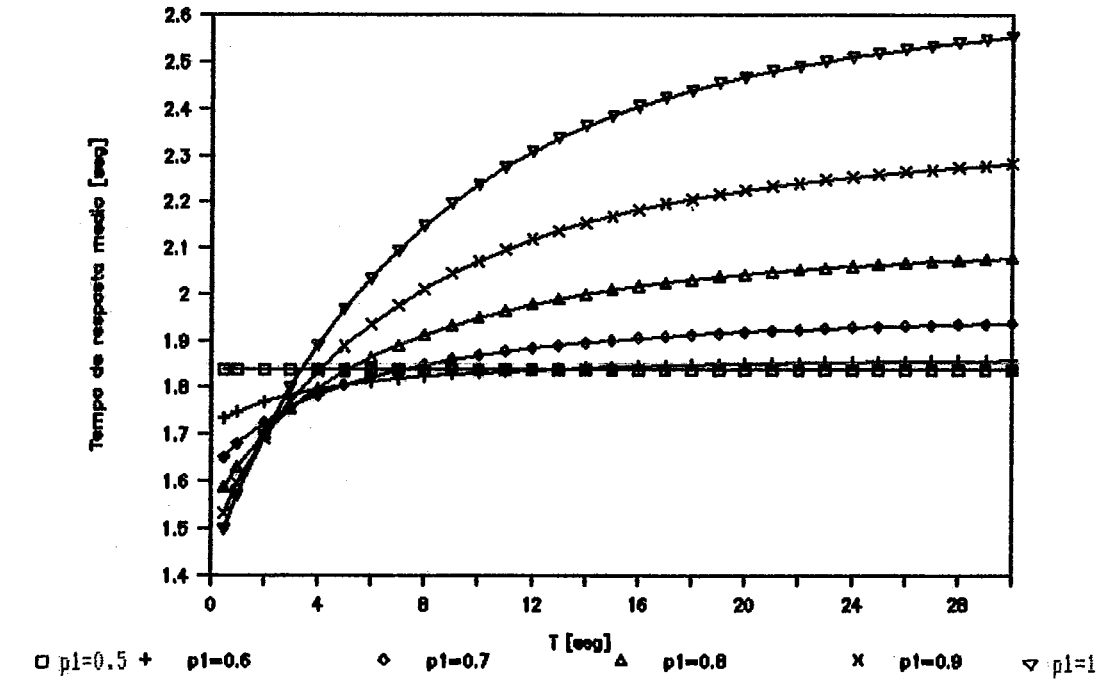

Figura VI.27: Tempo de resposta médio no sistema vs T para diferentes valores da probabilidade de atribuição

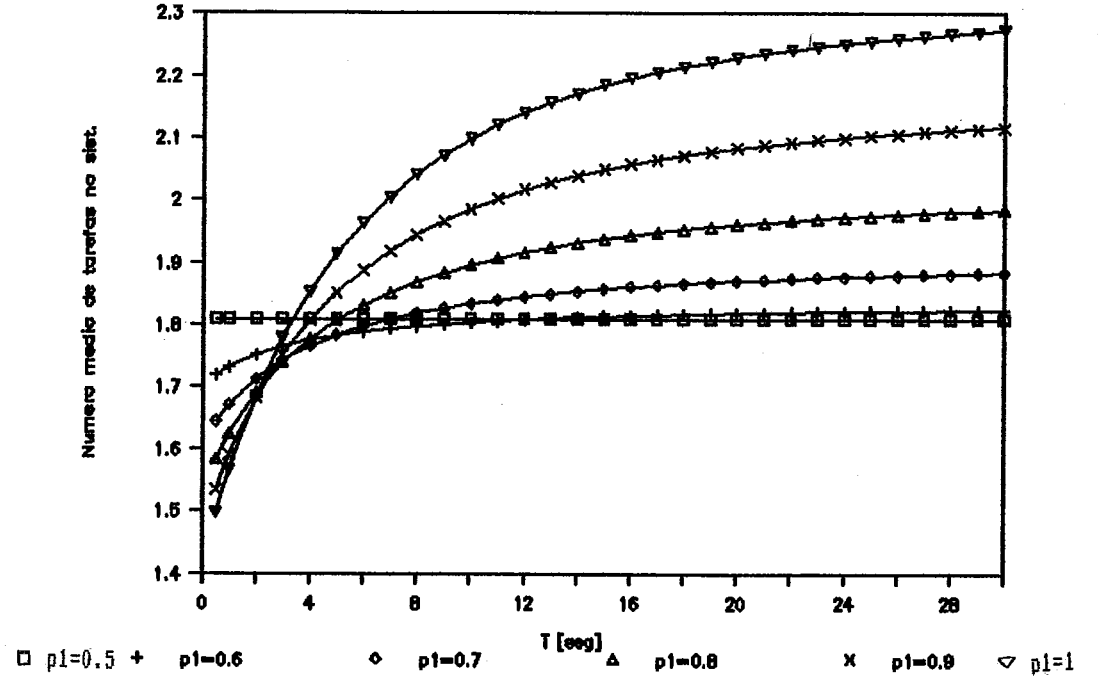

Figura VI.28: Número médio de tarefas no sistema vs T para diferentes valores da probabilidade de atribuição

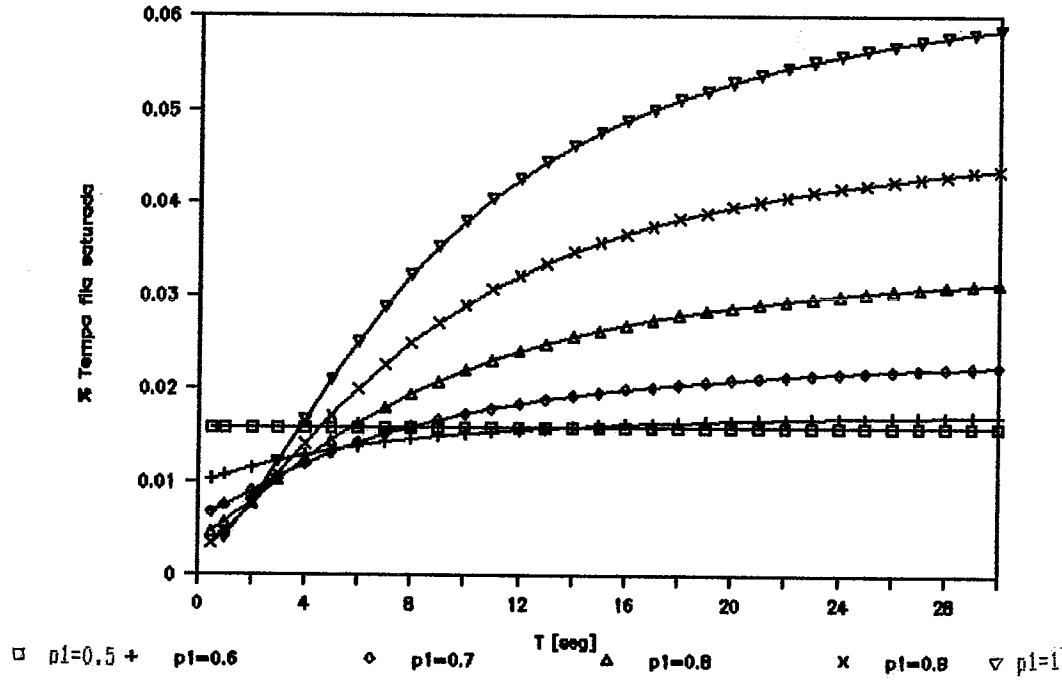

Figura VI.29: Tempo de fila saturada vs T para diferentes valores da probabilidade de atribuição

Mudando para $\lambda = 2$ [tarefas /seg], e mantendo os outros parâmetros iguais ao experimento anterior, foram obtidas medidas de desempenho que serão mostradas a seguir.

Parâmetros usados:

- Modelo 1
- Regras de balanceamento 1, $p_1 = 0.5, .., 1$
- buffer dos servidores = 5 [tarefas]
- $\mu = 1$ [tarefas /seg]
- $\lambda = 2$ [tarefas /seg]

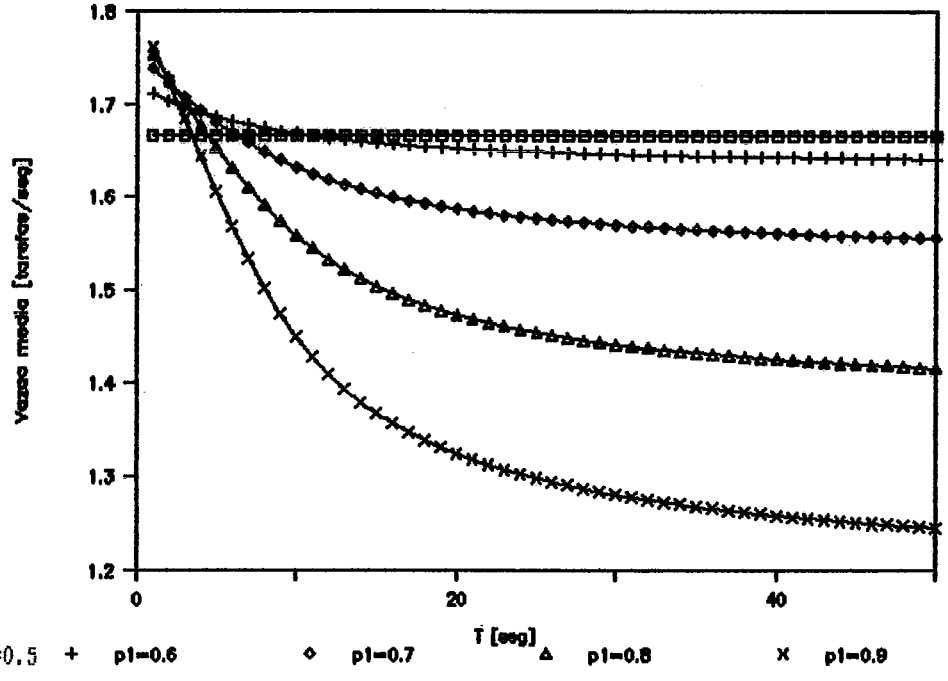

Figura VI.30: Vazão média no sistema vs T para diferentes valores da probabilidade de atribuição

A figura VI.30 mostra a vazão em função de T para diferentes probabilidades de atribuição. O comportamento é igual ao explicado para o exemplo anterior.

A VI.31 mostra a vazão como uma função da probabilidade de atribuição para diferentes T. Pode ser observado que para T=1 [seg] a vazão ótima é obtida para $p_1 = 1$. Para T=3 [seg] a vazão ótima é obtida para $p_1 = 0.7$ (para este mesmo T, o exemplo anterior com $\lambda = 1$ [tarefas /seg], a vazão é maximizada quando $p_1 = 0.8$).

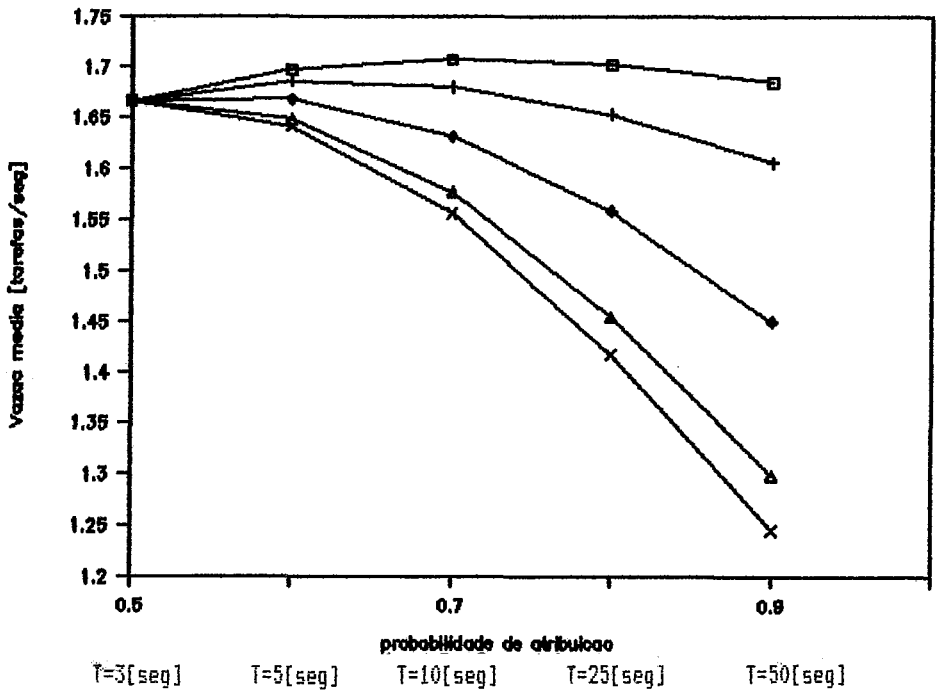

Figura VI.31: Vazão média no sistema vs probabilidade de atribuição para diferentes valores de T

A figura VI.32 mostra o tempo de resposta em função de T para diferentes probabilidades de atribuição.

Para $p_1 = 0.5$, o $\rho = 1$ para cada uma das filas, devido a que a taxa de chegada ao sistema se divide igualmente para cada centro, formando duas filas M/M/1/5. Neste caso, o número médio em cada centro é 2.5 (eq. VI.1), e portanto, o número médio de tarefas no sistema é igual à soma do número médio das duas filas, isto é, 5 (ver figura VI.33).

A figura VI.34 mostra o percentual do tempo que a fila permanece saturada em função da probabilidade de atribuição

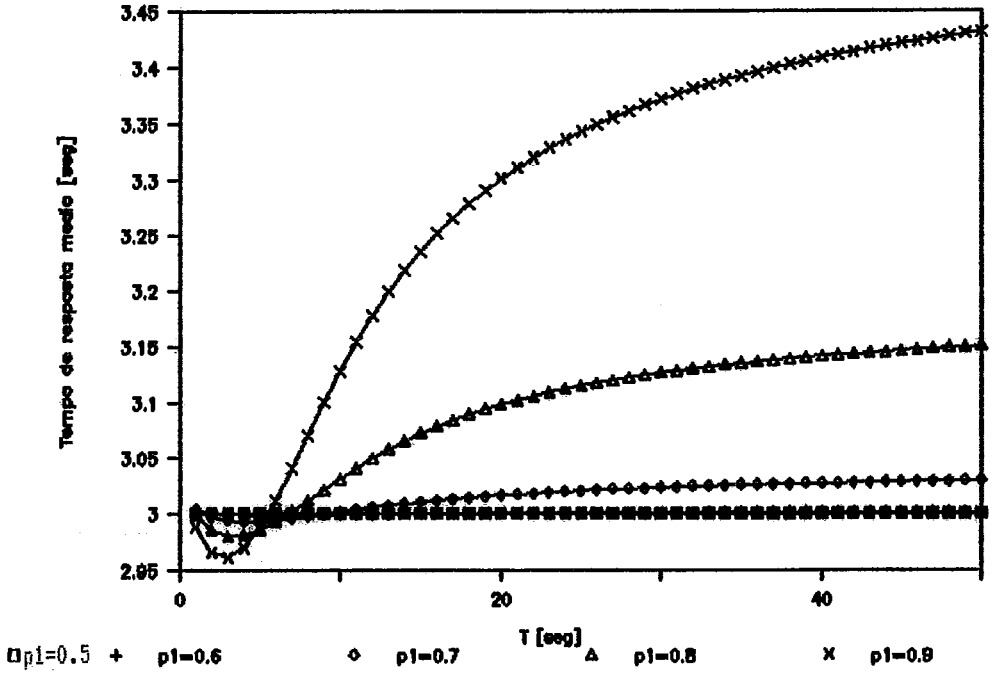

Figura VI.32: Tempo médio de resposta vs T para diferentes valores da probabilidade de atribuição

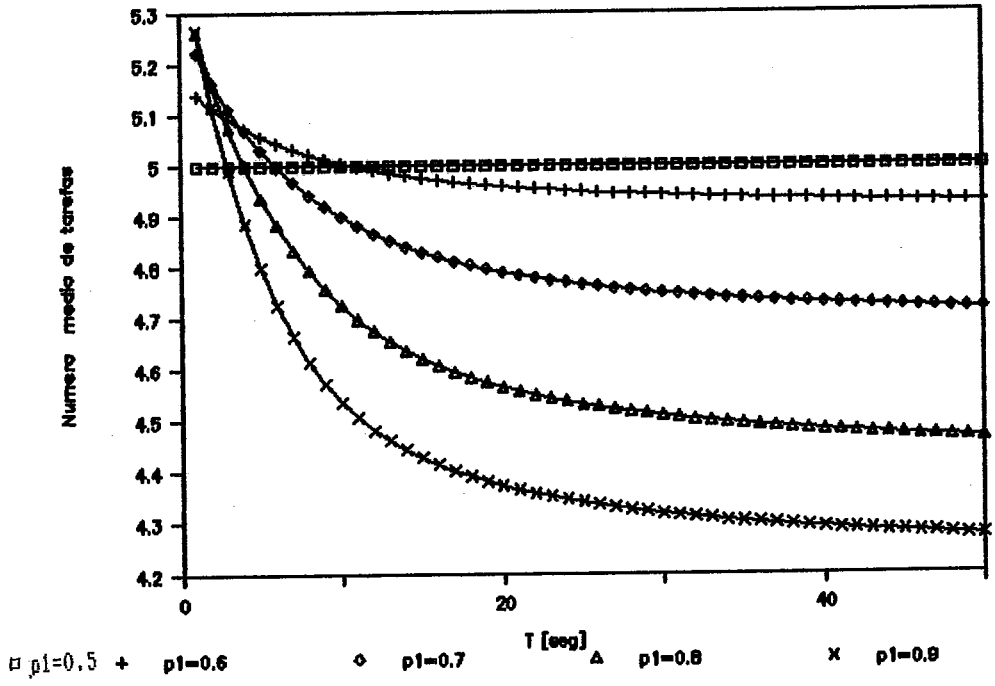

Figura VI.33: Número médio de tarefas no sistema vs T para diferentes valores da probabilidade de atribuição

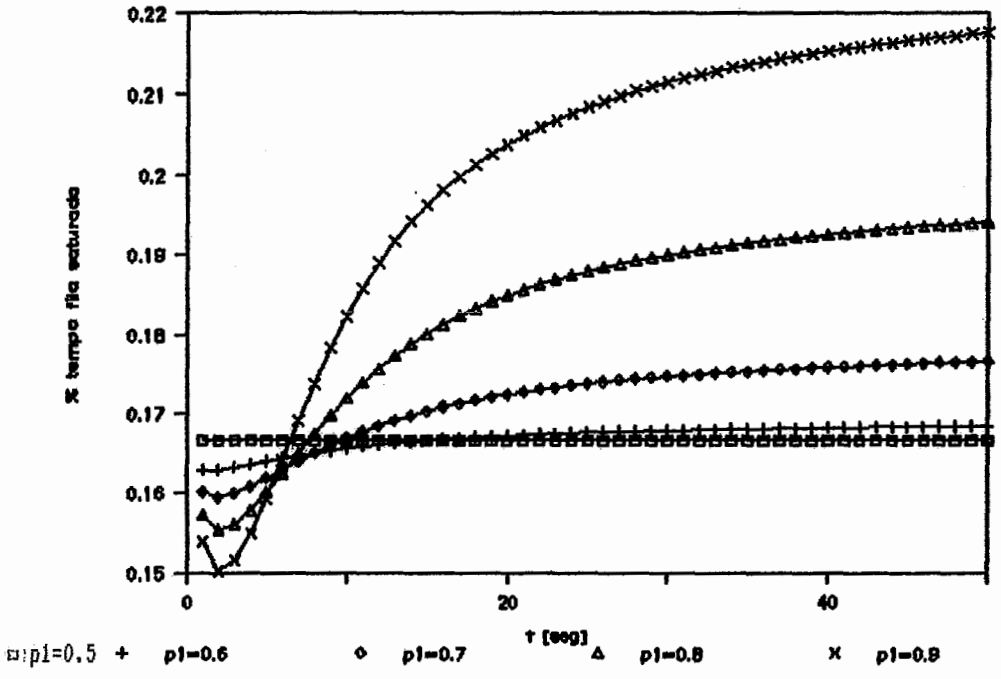

Figura VI.34: Percentual de tempo que a fila permanece saturada vs T para diferentes valores da probabilidade de atribuição

VI.7 Sensibilidade em relação ao tamanho do buffer

Neste experimento se procurou determinar a sensibilidade com relação ao tamanho do buffer dos servidores. Os parâmetros usados são:

- Modelo 1
- Regra de balanceamento 1
- buffer dos servidores =1, 2, 3, 4, 5, 6, 7, 8, 9 e 10 [tarefas]
- $\mu = 1$ [tarefas /seg]
- $\lambda = 0.7$ [tarefas /seg]

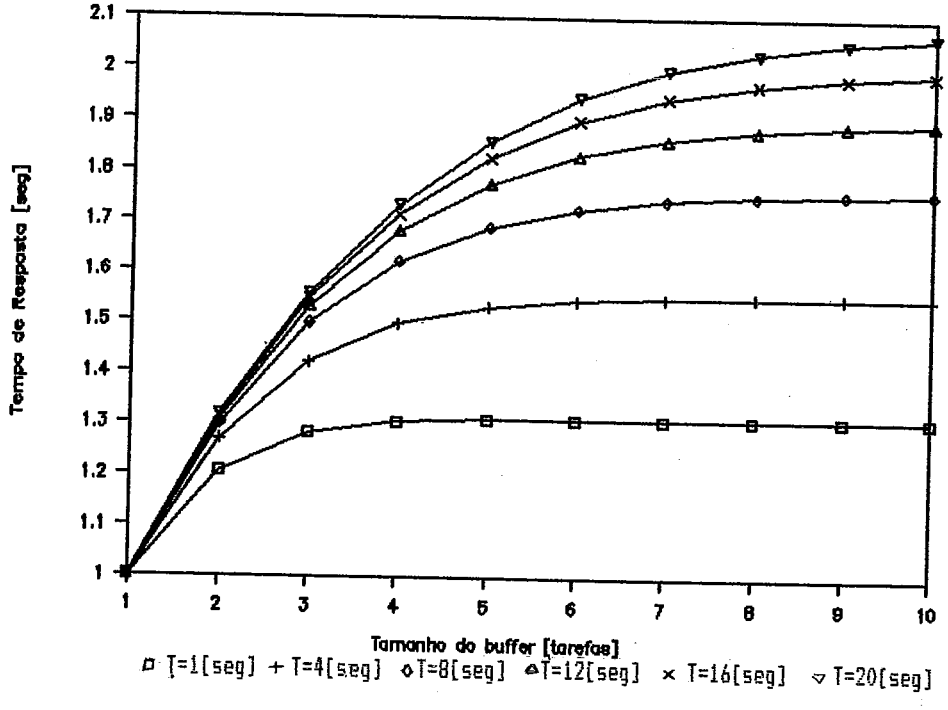

Figura VI.35: Tempo médio de resposta vs tamanho do buffer para diferentes T

A figura VI.35 mostra o gráfico do tempo médio de resposta em função do tamanho do buffer para diferentes T. Pode ser observsdo que, neste gráfico, o menor tempo de resposta é obtido para T=1 [seg]. Neste caso, o gerente enviará as tarefas para as filas de menor comprimento, fazendo desta maneira que as tarefas sejam atendidos com mais rapidez. Para T maiores, a desatualização da informação de estado faz com que o gerente envie as tarefas para filas cheias aumentando desta maneira o tempo de resposta das tarefas que ficam no sistema.

Pode ser observado no gráfico da figura VI.35 que o melhor tempo de resposta, para qualquer valor de T, obviamente, está dado para um sistema que não possui fila (buffer=1 [tarefa]), pois as tarefas que chegam são atendidas, se o servidor está vazio, ou são perdidas. É por esta mesma razão que se obtém a menor vazão do sistema, como se vê na figura VI.36. O tempo médio de resposta aumenta a medida que aumenta o tamanho do buffer. Isto porque ao aumentar o buffer as tarefas rejeitadas devido ao espaço reduzido do buffer, agora entram em fila para esperar serviço. Estas são as tarefas que farão aumentar o tempo de resposta.

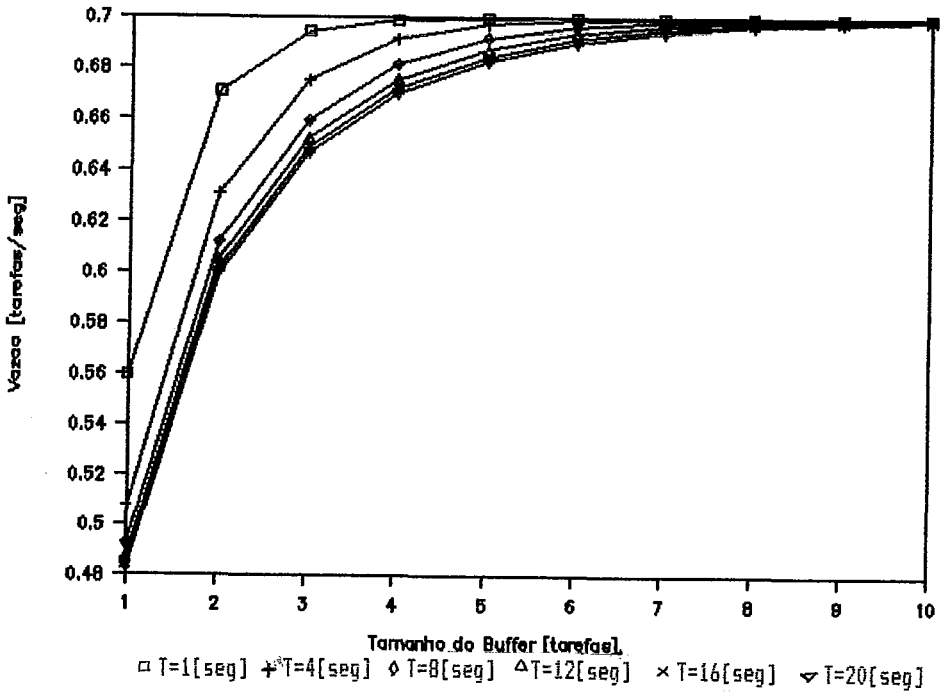

Figura VI.36: Vazão do sistema vs tamanho de buffer para diferentes valores de T

A figura VI.36 mostra a vazão média no sistema versus tamanho de buffers para diferentes T. Pode ser observado que a medida que aumenta o buffer aumenta também a vazão do sistema, pois menos tarefas são perdidas. A vazão é bastante afetada pelo tamanho de buffer. Ao variar o tamanho do buffer de 1 para 2 tarefas, a vazão aumenta aproximadamente 19%, para T=1 [seg]. Para T=20 [seg], este aumento é de aproximadamente 24 %. Estes aumentos são menores a medida que o buffer é maior. Por exemplo, para T =1 [seg] e buffers de 6 e 10 tarefas, não existe muita influência. Porém para T=20 [seg] o aumento entre buffer=6 [tarefa] e buffer=10 [tarefa] é aproximadamente 1.45%. Portanto, a sensibilidade da vazão com o tamanho do buffer é maior para T maior.

O gráfico da figura VI.37 mostra o número médio de tarefas no sistema em função do tamanho do buffer para diferentes valores de T. Pode ser visto que, para qualquer T, o menor número médio é para um sistema com buffer=1 [tarefa], pois neste caso as tarefas só permanecem no sistema durante o serviço. Para T=1 [seg], o número médio de tarefas para buffer=1 [tarefa] é maior que para os outros valores de T, já que neste caso mais tarefas são servidas. Para $T > 1$ [seg],

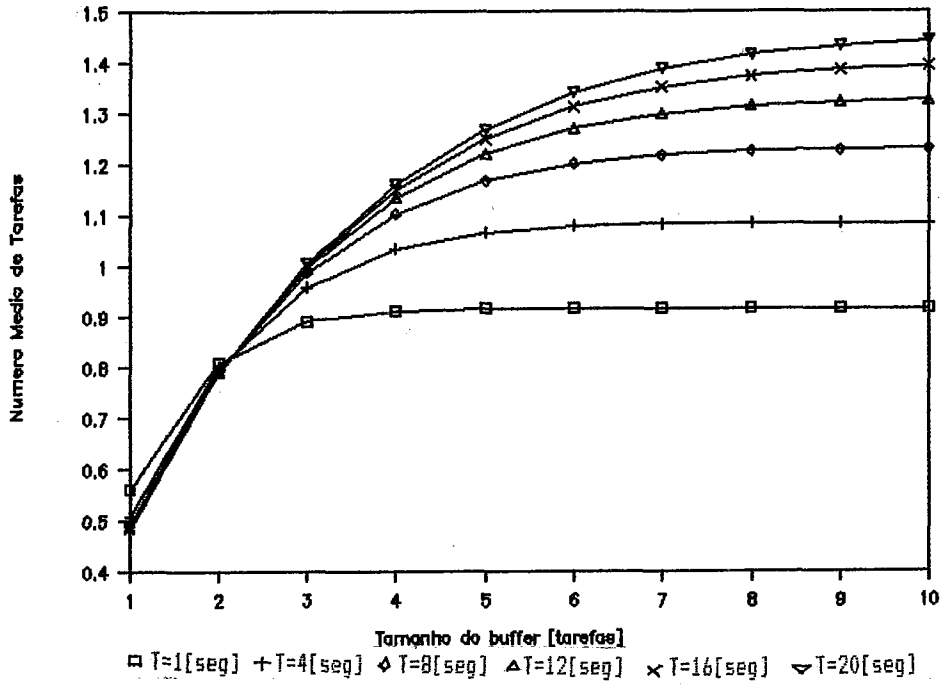

Figura VI.37: Número médio de tarefas no sistema vs tamanho do buffer para diferentes valores de T

tarefas são perdidas devido à informação desatualizada.

A sensibilidade do número médio de tarefas do sistema aumenta a medida que o tamanho do buffer aumenta. Por exemplo ao variar o buffer de 1 a 2, o número médio se incrementa em aprox 60%, para $T > 1$ [seg]. Para $T > 4$ [seg], a medida que o buffer aumenta, aumenta também o número médio no sistema, isto devido à influência da informação atrasada que faz o gerente enviar tarefas para filas cheias, aumentando desta maneira o número de tarefas no sistema.

A figura VI.38 mostra o percentual de tempo que a fila permanece saturada em T. Nesta figura pode ser visto que para buffer=1 [tarefa] e T=1 [seg] o percentual de tempo que a fila permanece saturada é maior quando o tempo entre atualizações é menor, pois o gerente distribui melhor as tarefas, já que sabe com maior certeza para que fila envia-las, isto implica que ambos buffers ficarão sempre ocupados. Para T grandes (buffer=1 [tarefa]) o gerente repartirá erradamente as tarefas mandando-as para a fila já saturada e deixando a outra fila vazia, fazendo com isto que o percentual de tempo saturado diminua.

Para tamanhos de buffer diferentes de 1, a medida que o intervalo T cresce, aumenta também o percentual de tempo que a fila permanece saturada, já que nestes casos a informação está tão atrasada que o gerente enviará tarefas para filas que já possuem tarefas, fazendo com isto que a fila sature com maior freqüência

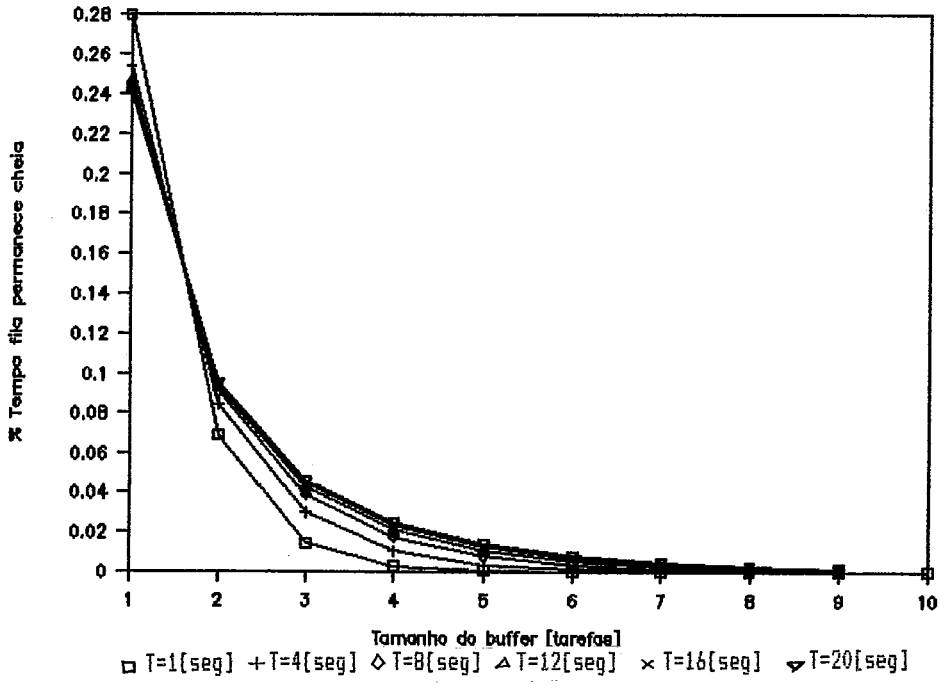

Figura VI.38: Percentual de tempo que a fila permanece saturada vs tamanho do intervalo para dif. valores de T

Pode ser observado que a medida que o buffer aumenta, o percentual de tempo que a fila permanece saturada diminui. Para tamanhos de buffer grandes (8,9 10), o percentual de tempo que a fila permanece saturada, cai aproximadamente a zero pois a utilização do sistema não é muito elevada.

VI.8 Sensibilidade com relação à taxa de serviço

O propósito deste experimento é verificar a sensibilidade com relação à taxa de serviço dos servidores de um sistema homogêneo. Os parâmetros são:

- Modelo 1
- Regra de balanceamento 1, $p_1 = 1$
- buffer dos servidores = 8 tarefas
- $\mu = 1, 2, 10$ [tarefas/seg]
- $\lambda = 0.8$ [tarefas/seg]

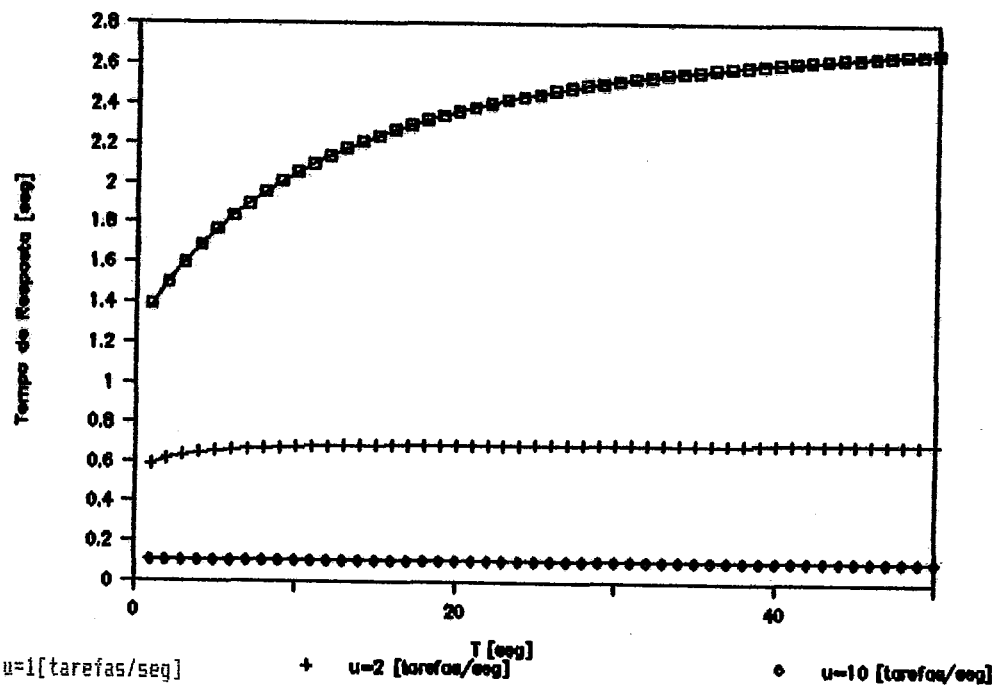

Figura VI.39: Tempo médio de resposta vs T para diferentes taxas de serviço

A figura VI.39 mostra o gráfico de tempo de resposta vs T para as as taxas de serviço 1, 2 e 10 [tarefas / seg], de um sistema homogêneo de dois servidores.

Para $\mu = 10$ [tarefas /seg], pode ser observado no gráfico VI.39, que o tempo de resposta tem um valor estacionário aproximadamente igual ao tempo de servico. Isto se deve a que a taxa de chegada de tarefas ao sistema é baixa ($\lambda = 0.8$ [tarefas / seg]) em relação a μ. Para valores de $\mu = 2$ [tarefas /seg] o tempo de resposta varia desde 0.589 seg para T=11 [seg], a 0.6947 seg. para T=50 seg, isto é

aprox 17.94% . Para $\mu = 1$ [tarefas /seg] este incremento é mais acentuado, sendo que entre T=1 [seg] e T=50 [seg] o incremento é de aprox. 106.2 % . Portanto, quanto mais rápido o sistema, menor a influência da velocidade de atualização de estado para a mesma faixa de valores de T.

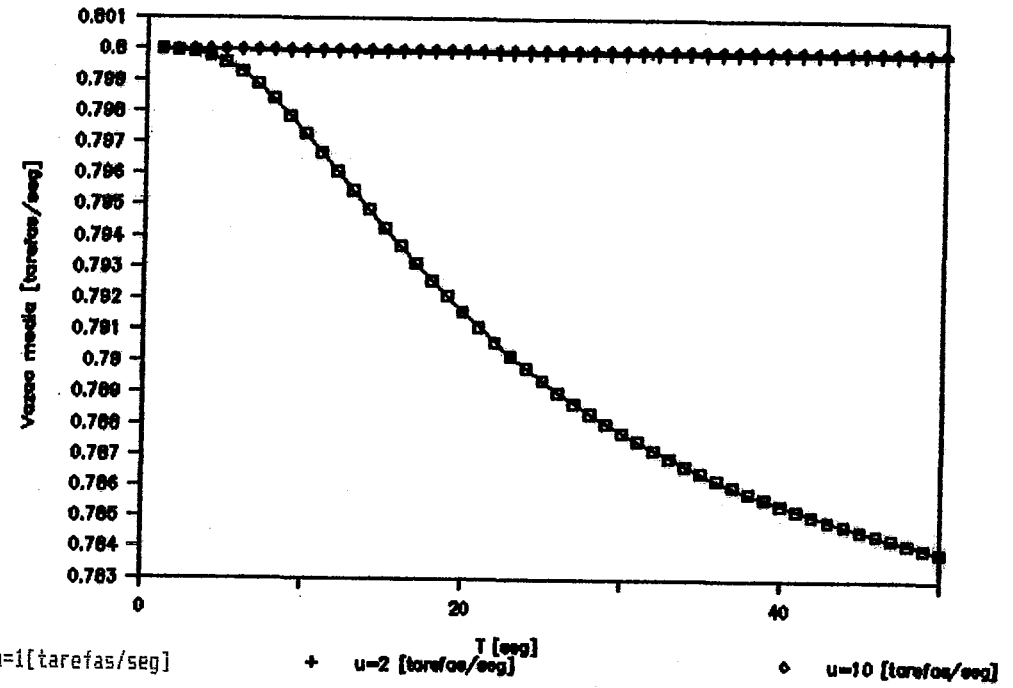

Figura VI.40: Vazão média no sistema vs T para diferentes taxas de serviço

A figura VI.40 mostra a sensibilidade da vazão média do sistema com relação à taxa de serviço. Pode ser observado, no gráfico, que a vazão tem um comportamento similar, para as três taxas de serviço, até para T=3 [seg]. A partir daí, a vazão diminuiu à medida que aumenta T, para $\mu = 1$ [tarefas /seg]. Enquanto que para $\mu = 2$ [tarefas /seg] e $\mu = 10$ [tarefas /seg], continua sendo similar para qualquer tamanho de intervalo. Isto é devido a que a taxa de chegadas de tarefas ao sistema é baixa e os servidores são rápidos, não deixando que se forme muita fila. (Veja figura VI.42). A vazão para $\mu = 1$ [tarefas /seg] cai a partir de $T > 3$ [seg], chegando a uma queda aproximada de 2 % em relação às outras taxas de serviço, para T=50 [seg].

No gráfico da figura VI.41 pode ser observado que o número médio de tarefas no sistema permanece constante, para quando o $\mu = 10$ [tarefas /seg], e é crescente com T quando o $\mu = 1$ [tarefas /seg].

Neste gráfico da figura VI.42 pode ser notado a influência da taxa de serviço no tempo em que a fila permanece cheia. Como era de se esperar, quando o tempo de serviço é maior ($\mu = 1$ [tarefas /seg]), a fila no servidor fica mais saturada, aumentando mais ainda para tempos grandes de atualização da informação de estado no gerente.

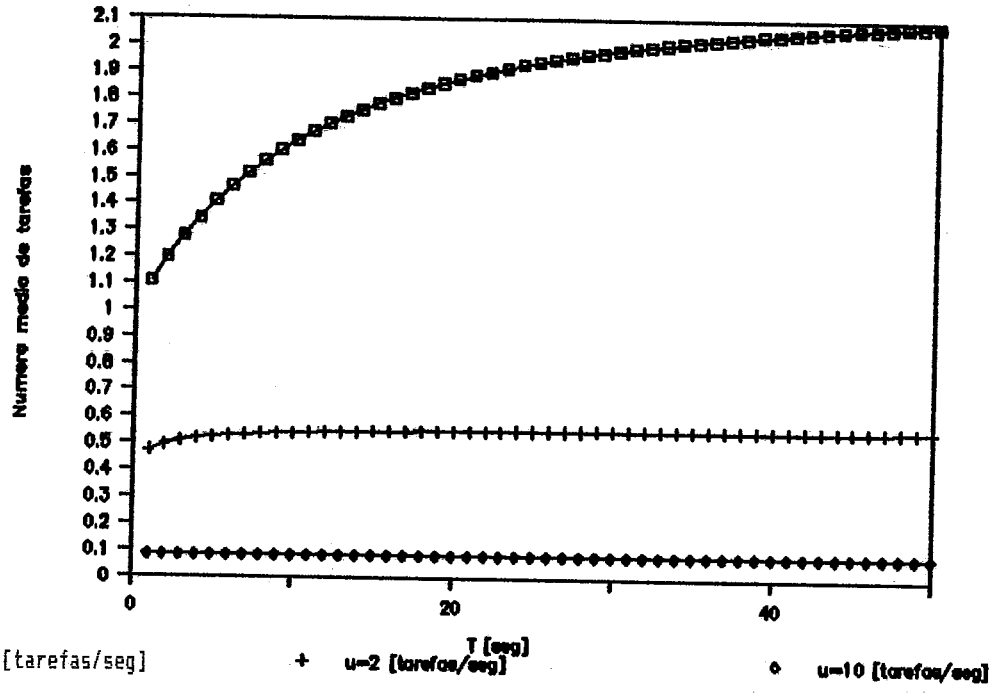

Figura VI.41: Número médio de tarefas no sistema vs T para diferentes taxas de serviço

O tempo que a fila fica saturada para $\mu = 2$ [tarefas /seg] é, aproximadamente, zero. E é zero para $\mu = 10$ [tarefas /seg].

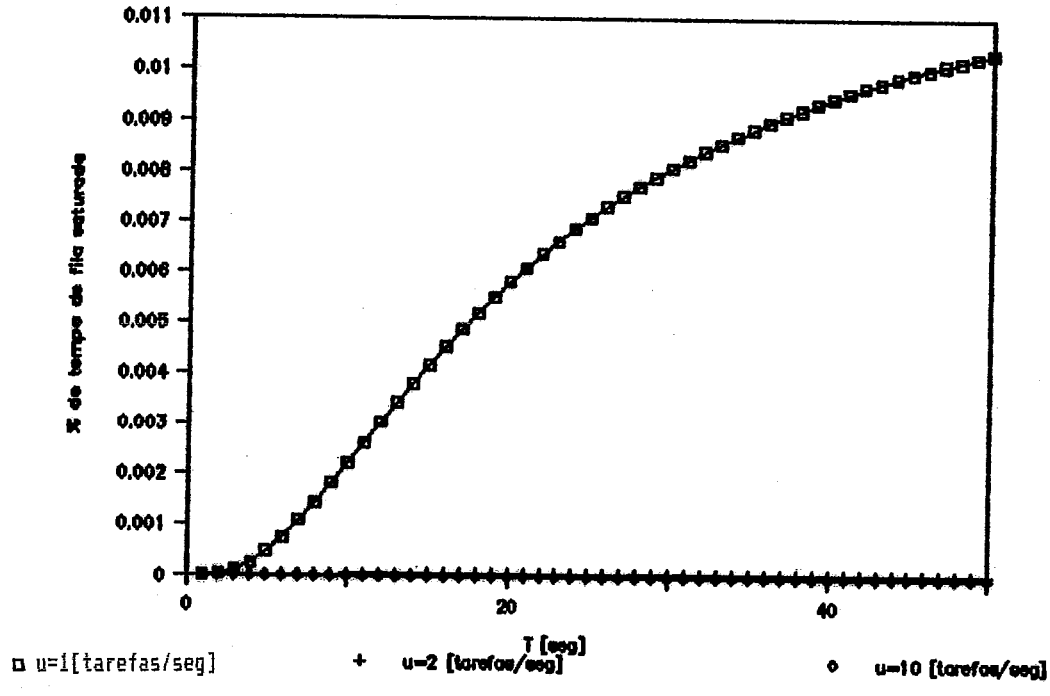

Figura VI.42: Tempo que a fila permanece cheia vs T para diferentes taxas de serviço

VI.9 Análise da regra 2

Os parâmetros usados para analisar esta regra são os seguintes:

- Modelo 1
- Regra de balanceamento 2
- buffer dos servidores = 9 e 10 [tarefas]
- $\mu = 1$ [tarefa / seg]
- $\lambda = 0.8, 0.9$ [tarefas / seg]

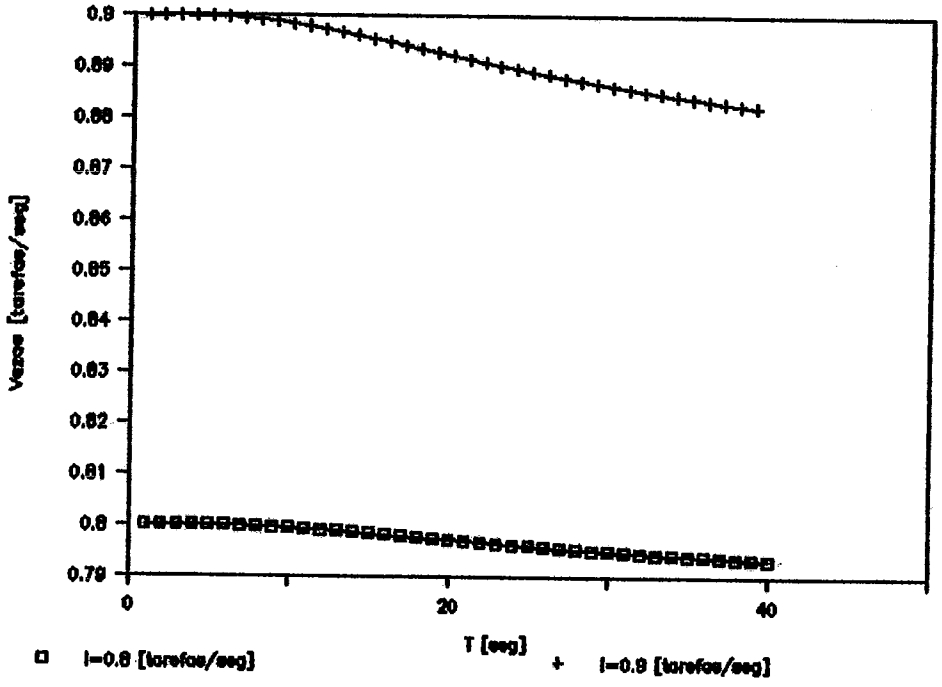

Figura VI.43: Vazão média do sistema vs T para buffer=10 [tarefas]

O gráfico da figura VI.43 mostra a vazão média do sistema para $\lambda = 0.9$ [tarefas /seg] e $\lambda = 0.8$ [tarefas /seg]. Pode-se notar que, para ambas taxas de chegada, a vazão diminui a medida que T aumenta. Por exemplo, para $\lambda = 0.9$ [tarefas /seg] a vazão se mantém constante para $T < 6$, e diminui para valores maiores. A maior diminuicão ocorre no intervalo [0,40] e é de, aproximadamente, 2 % .

Pode ser observado no gráfico da figura VI.44 que o tempo médio de resposta aumenta à medida que T aumenta, pois o gerente está atuando com informação desatualizada (ver fig VI.45). Neste gráfico, para T=1 [seg], o tempo de resposta é minimizado, para ambas taxas de chegadas. A variação da taxa de

chegada também faz com que o tempo de resposta aumente. Pode ser observado que, o incremento do tempo de resposta, ao passar de $\lambda = 0.8$ [tarefas /seg] a $\lambda = 0.9$ [tarefas /seg], é de aprox. 7.14 % para T=1 [seg], e para T=40 [seg] este incremento é de 21.3 % .

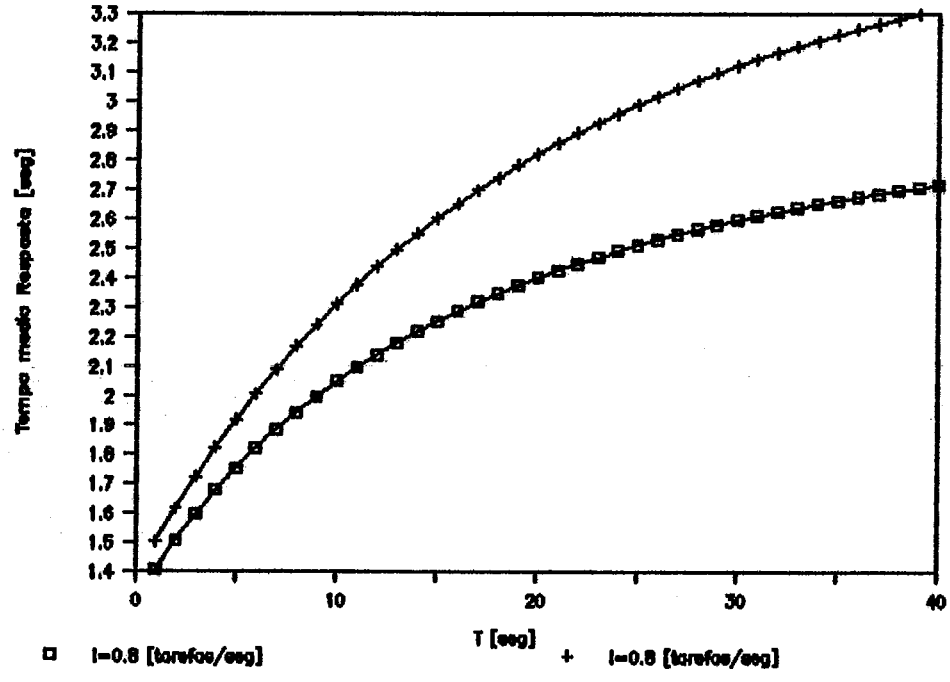

Figura VI.44: Tempo médio de resposta vs T para buffer=10 [tarefas]

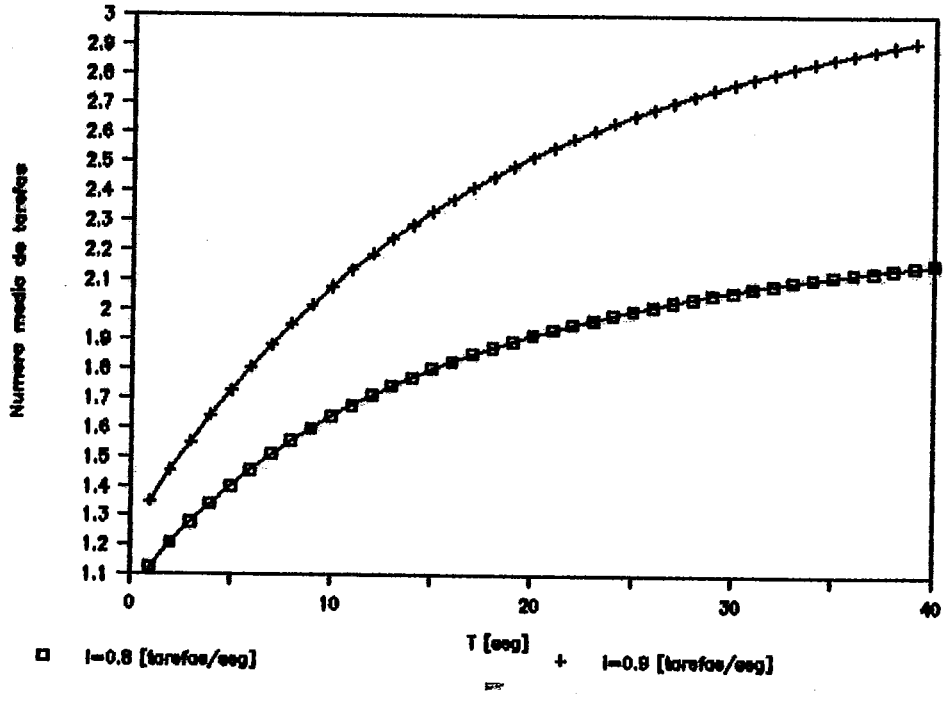

Figura VI.45: Número médio de tarefas no sistema vs T para buffer=10 [tarefas]

VI.9.1 Comparação entre a regra 1 e a regra 2

Nesta seção são apresentados os gráficos resultantes da comparação entre a regra 1 com p_1 variando de 0.5 a 1, e a regra 2 (probabilística). Os parâmetros usados são:

- Modelo 1
- Regras de balanceamento 1, $p_1 = 0.5, .., 1$ e regra 2
- buffer dos servidores = 8
- $\mu_1 = 1$ [tarefas /seg]
- $\mu_2 = 1$ [tarefas /seg]
- $\lambda = 0.8$ [tarefas /seg]

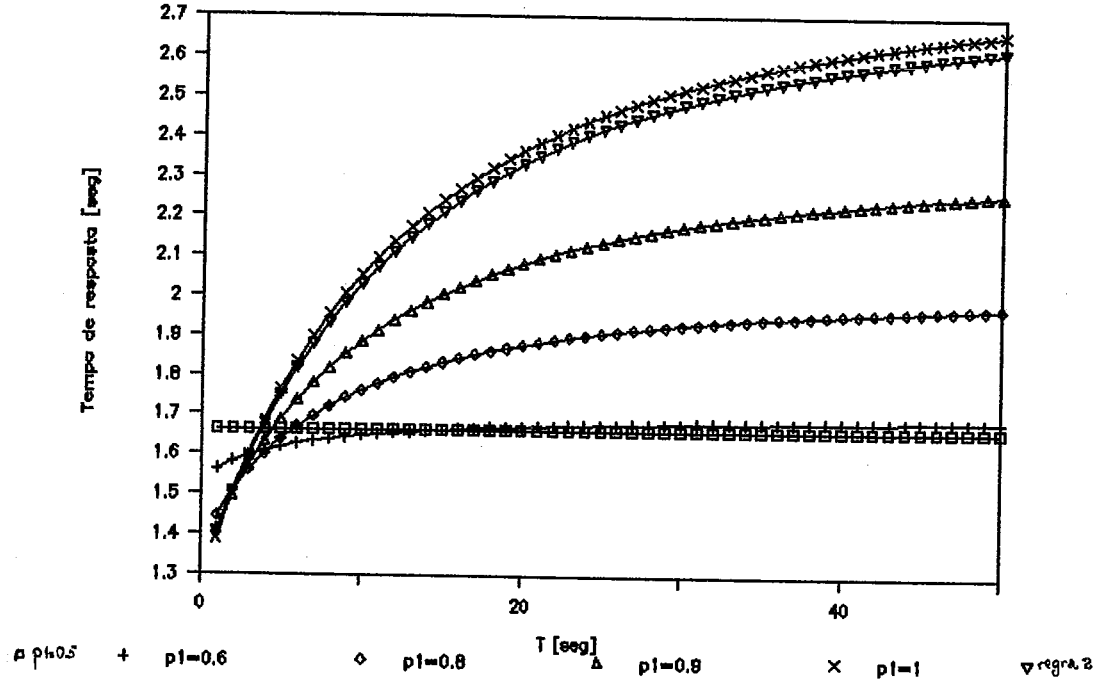

Figura VI.46: Tempo médio de resposta vs T para regra 1 com $p_1 = 0.5, .., 1$ e regra 2

A figura VI.46 mostra o tempo médio de resposta. Pode ser observado no gráfico que a regra 1 fornece o menor tempo de resposta, para T pequenos, está e com $p_1 = 1$, seguido pelo tempo dado para a regra 2. E o maior tempo de resposta ocorre para a regra 1 com $p_1 = 0.5$. Já para T grandes o pior tempo de resposta é para a regra1 com $p_1 = 1$.

A figura VI.47 mostra a vazão média do sistema para as diferentes regras de balanceamento. Pode-se observar nas figuras VI.48, VI.49 e VI.50 que,

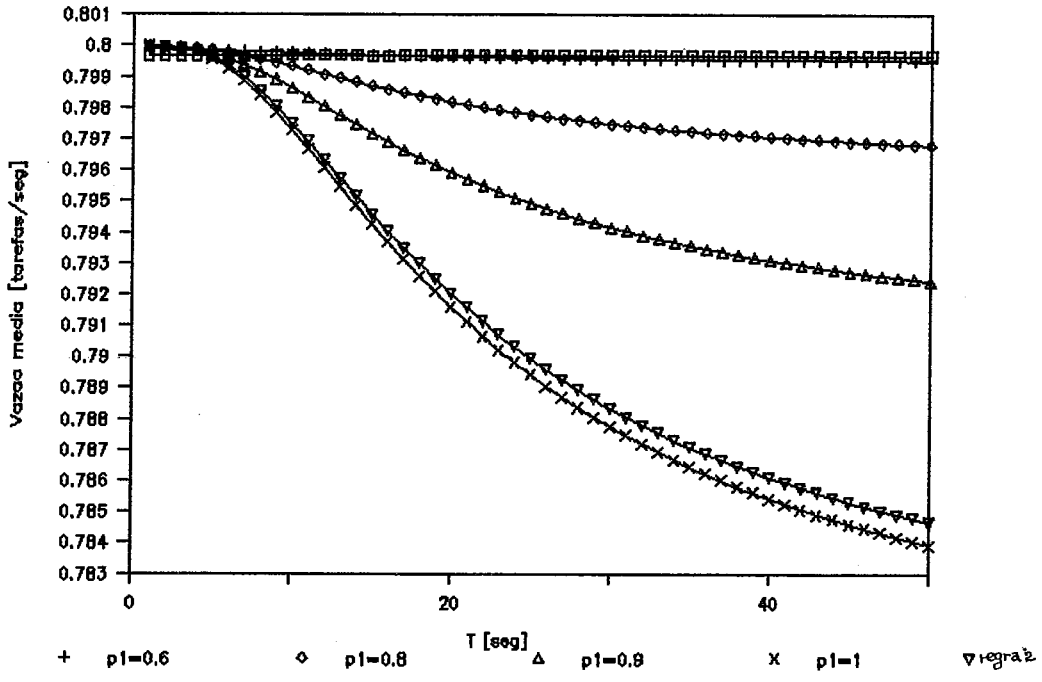

Figura VI.47: Vazão média vs T para regra 1 com $p_1 = 0.5, .., 1$ e regra 2

dependendo do tempo entre atualizações de informação de estado, existe uma regra que maximiza a vazão do sistema. Note-se, no gráfico da figura VI.48, que a regra 2 se comporta melhor para $T = 1$ [seg] do que a regra 1 para $p_1 = 0.5, 0.6$ e 0.7. A regra 2 tem comportamento idêntico à regra 1 com $p_1 = 0.8$. Porém, quando a regra 1 tem valores de $p_1 = 0.9$ e 1, a regra 2 tem um comportamento pior. A vazão máxima, para este T, é atingida pela regra 1 com $p_1 = 1$.

Para $T = 4$ [seg], figura VI.49, a vazão máxima está dada para a regra 1, $p_1 = 0.7$. Neste gráfico pode ser visto também que a regra 2 se comporta melhor que a regra 1 com $p_1 = 0.5$ e 1. Na figura VI.50 vê-se que para $T = 10$ [seg], a melhor vazão é dada pela regra $p_1 = 0.6$. Pode ser visto também, neste gráfico, que a regra 2 tem um comportamento pior que a regra 1, exceto para regra 1 com $p_1 = 1$.

Para $T > 10$ [seg], a regra que fornece a maior vazão é a $p_1 = 0.5$.

Dos resultados anteriores podemos observar que a regra 2 é sempre pior que a regra 1 quando o p ótimo é usado.

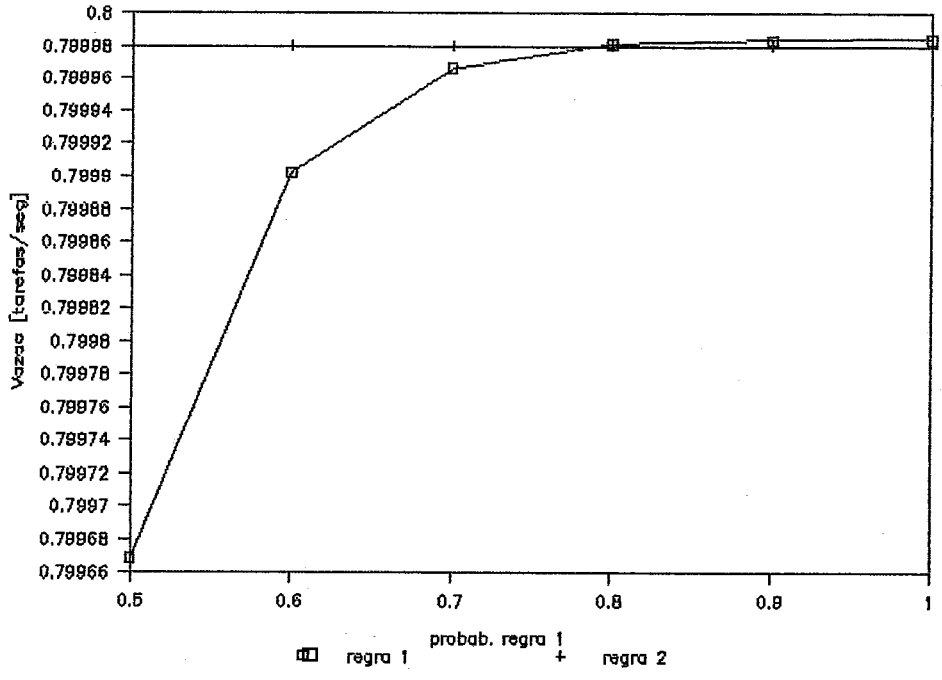

Figura VI.48: Vazão vs probabilidade de atribuição para a regra 1 e regra 2 para T=1 [seg]

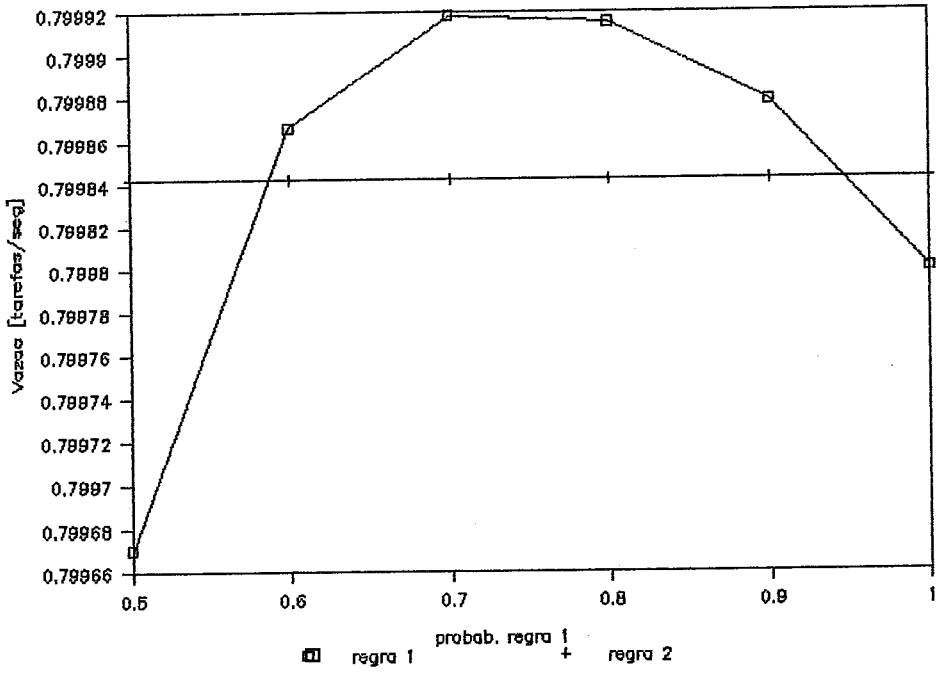

Figura VI.49: Vazão média vs probabilidade de atribuição para a regra 1 e regra 2 para T=4 [seg]

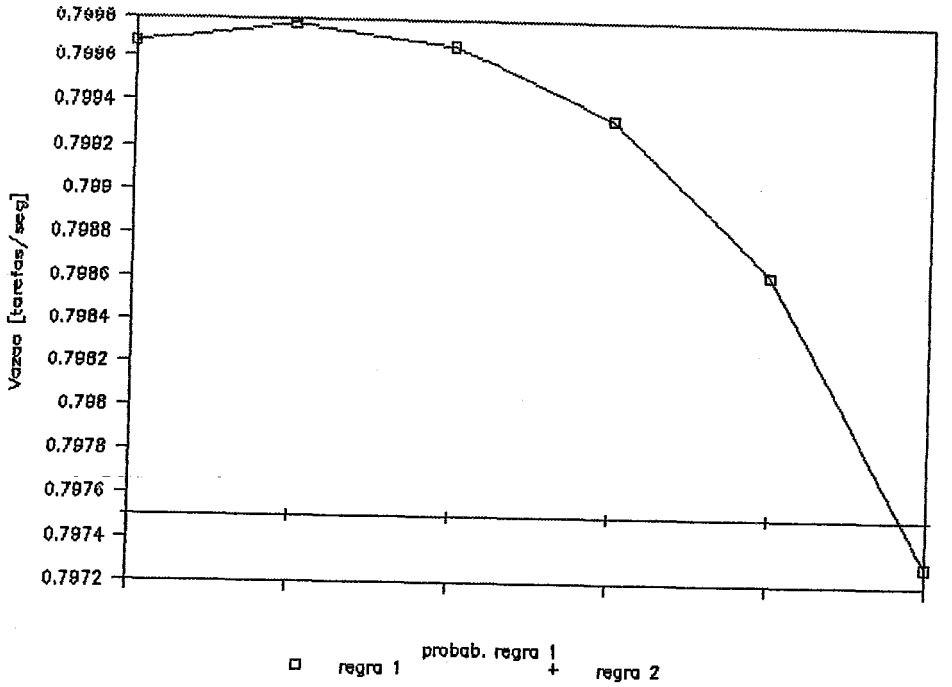

Figura VI.50: Vazão média vs probabilidade de atribuição para a regra 1 e regra 2 para T=10 [seg]

VI.10 Análise da regra 3

A seguir são mostrados os resultados obtidos para a regra 3. Esta regra é usada para um sistema descentralizado. As tarefas chegam a cada centro que toma então a decisão de executá-la ou não, contrastando com os exemplos acima onde um gerente central decide onde cada tarefa será executada. Para este tipo de modelo foi analisada a regra 3 com os seguintes parâmetros:

- Modelo 1
- Regra de balanceamento 3
- buffer dos servidores = 10 tarefas
- $\mu_1 = 1$ [tarefa /seg]
- $\mu_2 = 1$ [tarefa /seg]
- $\lambda = 1$ [tarefa /seg]
- $nmin = 2$ tarefas
- $nmax = 4$, 6 e 8 tarefas

Nota: mesmo neste exemplo 'descentralizado', os dados foram gerados considerando que o gerente de cada servidor possui informação de estado do servidor local atrasada. Entretanto, a mesma metodologia pode ser usada no caso em que informação de estado local é obtida imediatamente, e atrasos só ocorrem na obtenção de informação de outros servidores.

O gráfico da figura VI.51 mostra a vazão para diferentes valores de $nmax$. Pode ser visto que existe para cada T um certo $nmax$ que maximiza a vazão. Para $T < 4$, o $nmax$ que fornece a vazão máxima é $nmax$=4. Nestes casos, a informação do gerente está mais atualizada, e, por este motivo, ele envia as tarefas para filas mais vazias fazendo com que a tarefa não seja perdida. Quando T grande o limiar $nmax = 4$ [tarefas] é facilmente alcançado. Por isto, o gerente do centro deve transferir esta tarefa para o outro centro o qual, por simetria, se encontra na mesma situação. Isto faz com que muitas tarefas se percam, já que as filas tendem a encher, e portanto, a vazão cai. Para valores de $nmax$ maiores, esta tendência de transferir tarefas diminui e, portanto, a vazão não decresce tanto.

Pode ser visto no gráfico da figura VI.52 que o número médio de tarefas no sistema depende do valor do limiar $nmax$. Pode ser notado que quanto menor é o $nmax$ maior a sensibilidade do número médio de tarefas em relação a T. Já para $nmax$=8 não existe variação do número médio com relação a T. Neste caso, o sistema se está comportando como duas filas M/M/1/10, independentes, com taxa de chegada $\lambda/2$. Pela equação VI.1, o número médio para o centro é igual a 0.994 tarefas, fornecendo para o sistema um número médio de aprox 1.98 tarefas, e coincidindo com o resultado obtido.

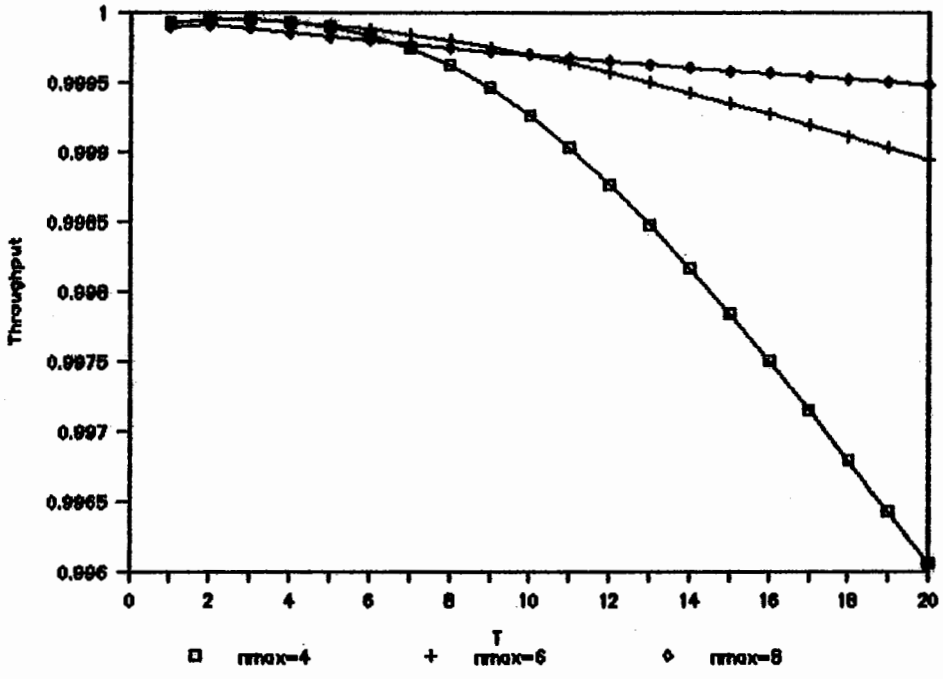

Figura VI.51: Vazão média vs T para dif. valores de *nmax*

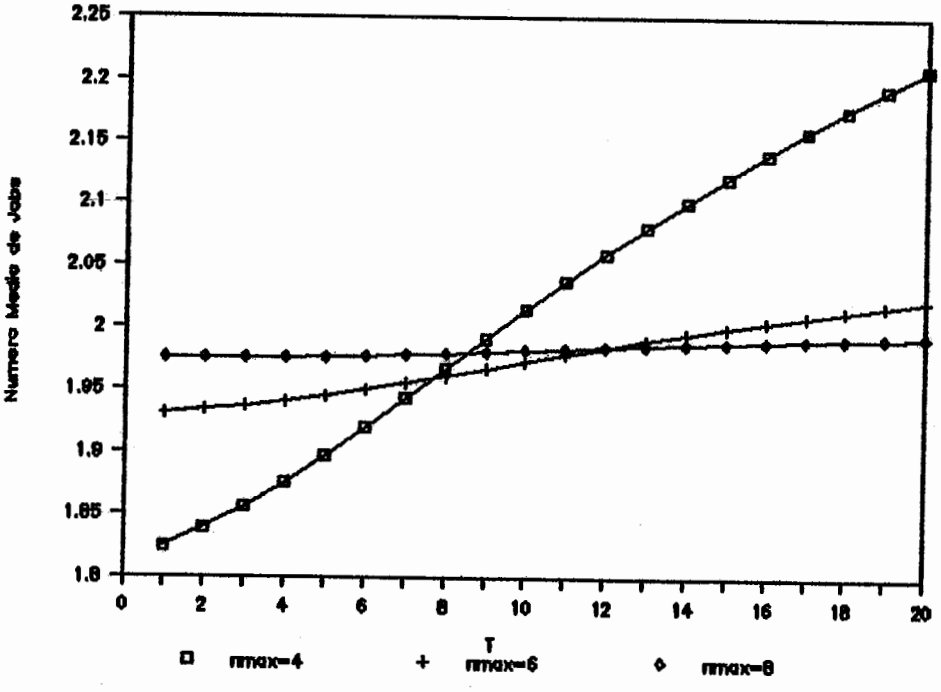

Figura VI.52: Número médio de tarefas no sistema vs T para diferentes valores de *nmax*

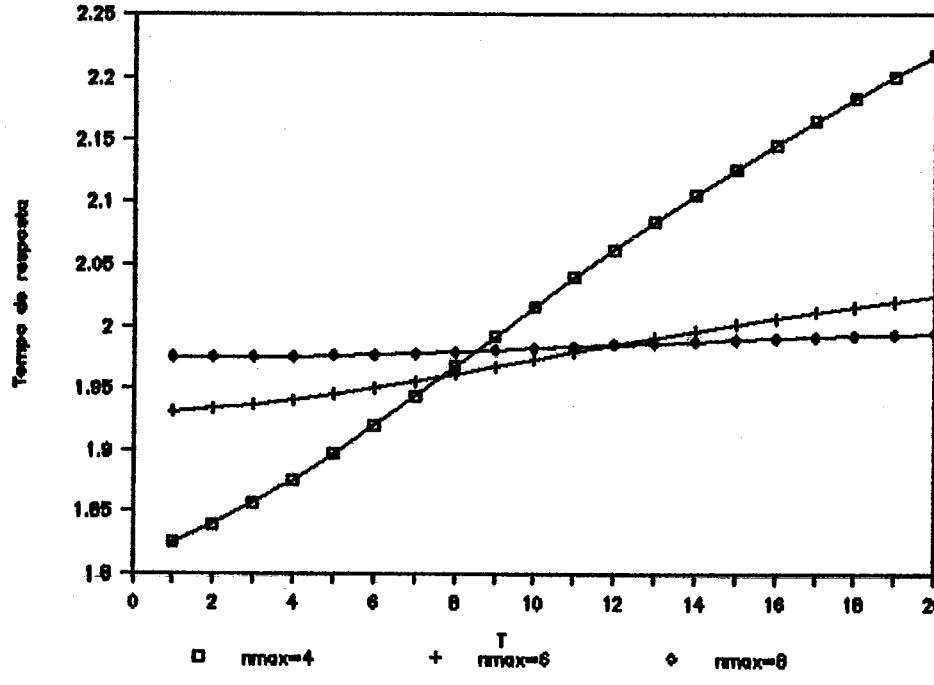

Figura VI.53: Tempo de resposta vs T para sistema descentralizado

VI.11 Servidores Heterogêneos

Neste experimento se estuda o comportamento de um sistema distribuído com dois centros, que possuem servidores com taxas de serviço diferentes. Foram analisados sistemas com diferentes capacidades de processamento. Específicamente, com $\mu_1 + \mu_2 = 0.5$, $\mu_1 + \mu_2 = 1$ e $\mu_1 + \mu_2 = 2$.

Os parâmetros comuns a todos os sistemas são:

- Modelo 2
- Regra de balanceamento 1, $p_1 = 1$
- buffer de cada servidor = 5 [tarefas]
- $\lambda = 1$ [tarefas /seg]

Os seguintes parâmetros são usados para 3 sistemas diferentes, mas com capacidade total de processamento igual a 1. Para este experimento o valor médio, para o sistema, é de $\rho = 1$.

- $\mu_1 = 0.5$ [tarefas /seg]
- $\mu_2 = 0.5$ [tarefas /seg]

- $\mu_1 = 0.67$ [tarefas /seg]
- $\mu_2 = 0.33$ [tarefas /seg]

- $\mu_1 = 0.25$ [tarefas /seg]
- $\mu_2 = 0.75$ [tarefas /seg]

Nota: A regra de balanceamento usada neste caso não leva em consideração a velocidade do servidor. Entretanto, a mesma metodologia pode ser usada para analisar uma regra de balanceamento que leve em consideração este fator.

A seguir são apresentados os resultados dos experimentos feitos.

A figura VI.56 mostra o tempo de resposta médio do sistema. Estes valores podem ser explicados com a metodología mostrada em VI.5

A mesma explicação dada para o caso homogêneo em VI.5 para o caso de percentual de tempo que a fila permanece cheia, serve para o caso de servidores heterogêneos. A seguir analisamos exemplos para alguns conjuntos de parâmetros diferentes. Caso 1: $\lambda = 1$; $\mu_1 = 0.25$, $\mu_2 = 0.75$, buffer=5

Análise do centro 1

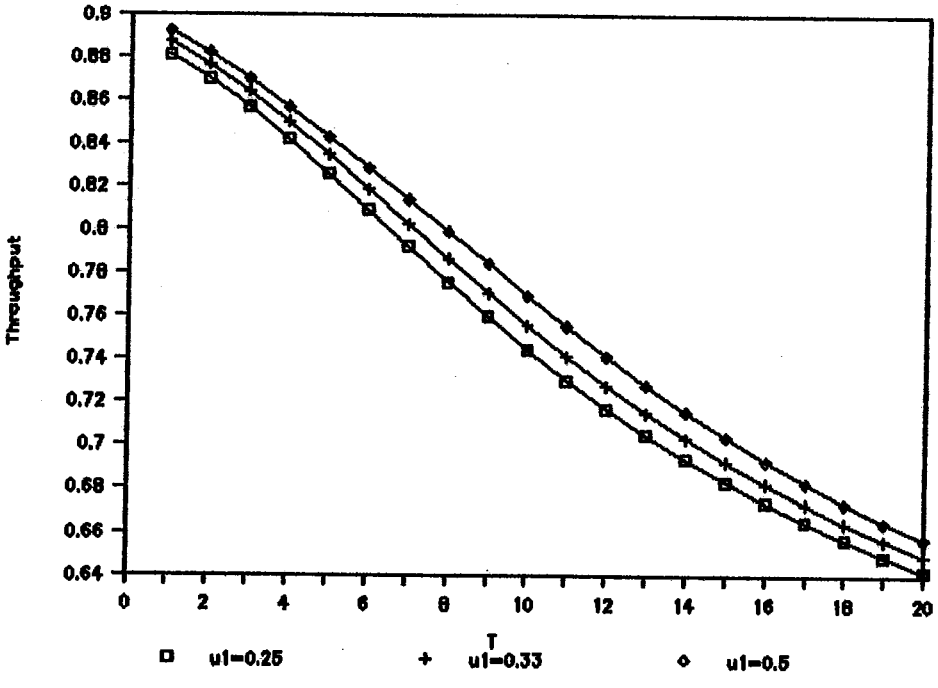

Figura VI.54: Vazão média vs T para servidores heterogêneos com capacidade total de serviço 1

- Caso T=0 [seg]

$$\lambda_1^* \approx \frac{\lambda\mu_1}{\mu_1 + \mu_2} = 0.25$$

portanto, $\rho = 1$ e pela equação VI.3 $P_1 = 0.17$.

Na figura observa-se que P_1 (probabilidade de bloqueio do servidor 1) ≈ 0.26. Pode-se entender a diferença considerando o fato de que, com grande probabilidade o sistema começa seus ciclos em estados balanceados. Nestes ciclos $\lambda_1 = \lambda/2$, portanto o $\lambda^*(0)$ da formula VI.2 é menor que o real, o que implica que o valor de $P_1(0)$ calculado acima é um limite inferior do valor de P_1 real.

- Caso $T = \infty$

Neste caso tem-se que para o ciclo de chegadas, $\lambda_c = \lambda = 1$, e portanto $\rho_c = 4$, de onde se tem que $P_1^*(\infty) \approx 0.75$, portanto $P_1 = 0.75/2 = 0.375$.

Apesar da figura não mostrar apropriadamente a tendência de $T \rightarrow \infty$, nela pode-se ver que o valor calculado corresponde, de forma bastante aproximada, ao resultado experimental.

Por último, a variação do gráfico, ao incrementar T desde 0, coincide com o fenômeno explicado para o caso dos gráficos de R_c no ponto VI.5.

Outro fenômeno interessante e evidente, é que a medida que aumenta μ_c, P_c diminue. Note-se que, o gráfico superior para o servidor 1 corresponde

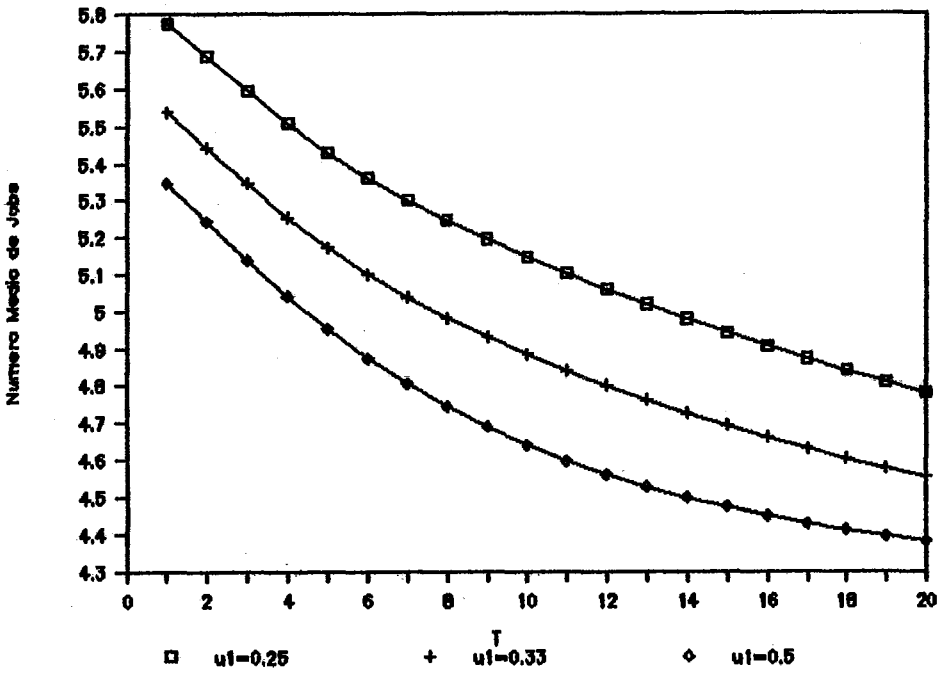

Figura VI.55: Número *médio* de tarefas no sistema vs T para servidores heterogêneos com capacidade total de serviço 1

ao mesmo experimento do gráfico inferior para o servidor 2, e viceversa (ver figuras VI.57 e VI.58).

Análise centro 2

$\lambda = 1$, $\mu_2 = 0.75$, $\mu_1 = 0.25$, B=5

- Caso T=0 [seg]

$$\lambda_2^* = \frac{\lambda\mu_2}{\mu_1 + \mu_2} = 0.75$$

$\rho_2^* = 0.75/0.75 = 1$

$P_2^*(0) \approx 0.17$

valor que corresponde, aproximadamente, ao obtido no gráfico.

- Caso $T = \infty$

Nos ciclos de chegada tem-se que $\lambda_2 = \lambda = 1$, e $\mu_2 = 0.75$, portanto $\rho_2 = 1/(3/4) = 4/3$.

De onde obtem-se que: $P_2^* \approx 0.3$, portanto $P_2 \approx 0.15$. Novamente, como o gráfico está desenhado até T=20 [seg], não se pode notar a tendência de $P_2(T)$ para $T \rightarrow \infty$.

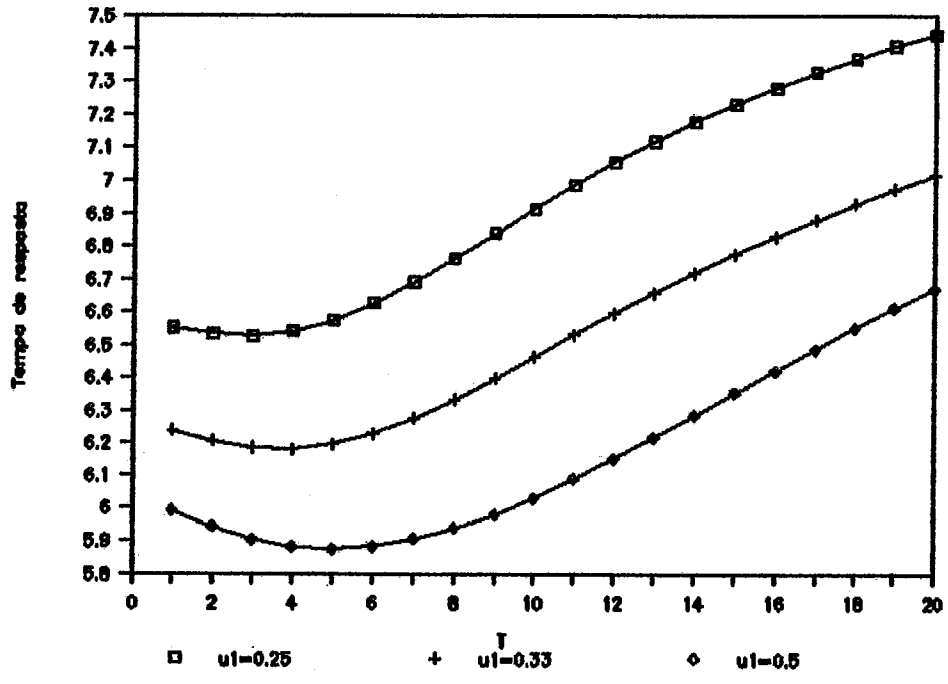

Figura VI.56: Tempo de resposta vs T para servidores heterogêneos com capacidade total de serviço 1

De forma geral, pode ser visto que, procedendo de forma análoga às explicações dadas em VI.5, para número médio, tempo de resposta e percentual de tempo que a fila permanece saturada, a metodologia de análise permite compreender os diferentes gráficos obtidos para estas medidas no sistema heterogêneo.

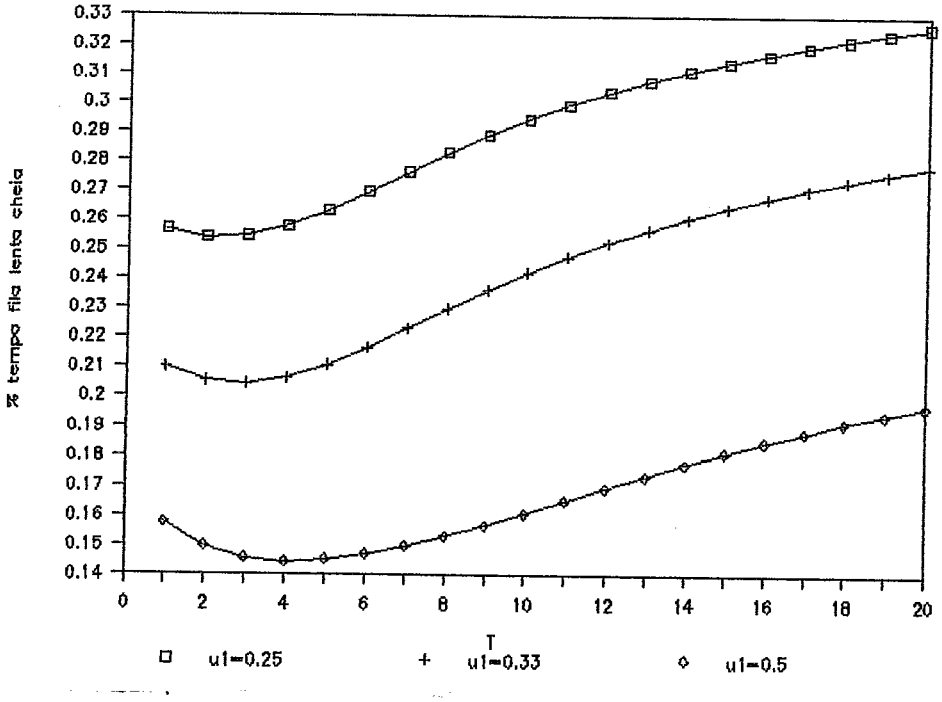

Figura VI.57: Percentual de tempo que a fila lenta permanece cheia vs T para servidores heterogêneos com capacidade total de serviço 1

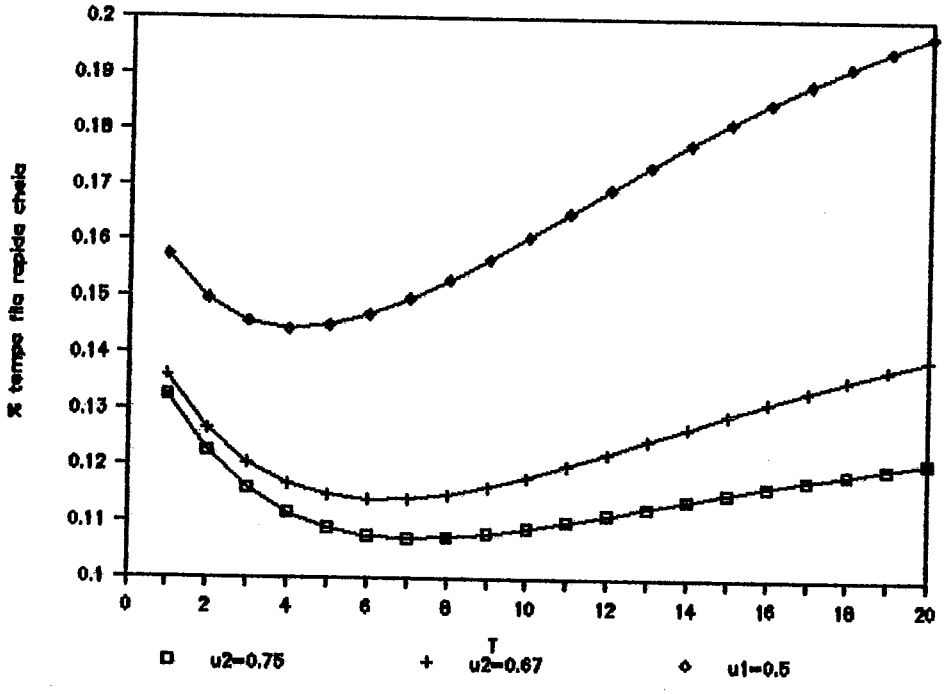

Figura VI.58: Percentual de tempo que a fila rápida permanece cheia vs T para servidores heterogêneos com capacidade total de serviço 1

Neste experimento a capacidade total de serviço do sistema é 0.5, ou seja em valor médio para o sistema o $\rho = 2$

- $\mu_1 = 0.25$ [tarefas /seg]
- $\mu_2 = 0.25$ [tarefas /seg]

- $\mu_1 = 0.125$ [tarefas /seg]
- $\mu_2 = 0.375$ [tarefas /seg]

- $\mu_1 = 0.167$ [tarefas /seg]
- $\mu_2 = 0.333$ [tarefas /seg]

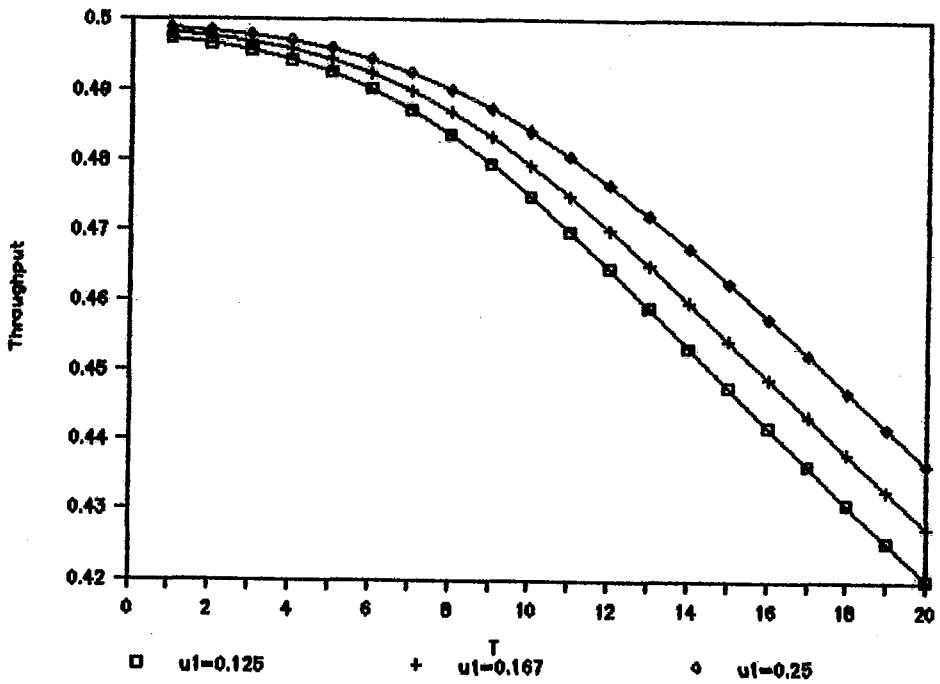

Figura VI.59: Vazão média vs T para servidores heterogêneos com capacidade total de serviço 0.5

Pode ser visto no gráfico da figura VI.59 que a melhor vazão é obtida para o servidor homogêneo, para qualquer tamanho de T. O sistema com os servidores mais heterogêneo ($\mu_1 = 0.125$ e $\mu_2 = 0.375$) é o que atinge a menor vazão pois, apesar de ter um servidor mais rápido que entrega uma vazão alta, a medida global inclui o servidor lento, o qual baixará este rendimento.

O maior número de tarefas (ver figura VI.60) é encontrado para o sistema com $\mu_1 = 0.125$ [tarefas /seg] e $\mu_2 = 0.375$ [tarefas /seg]. Para todos

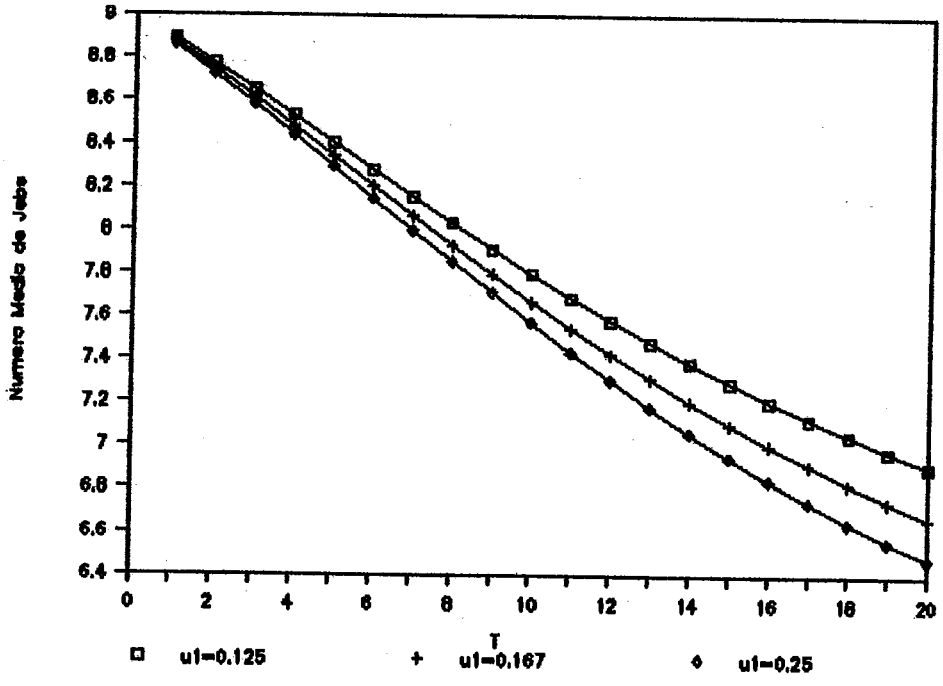

Figura VI.60: Número médio de tarefas no sistema vs T para servidores heterogêneos com capacidade total de serviço 0.5

os sistemas $\rho = 2$, pelo qual o número médio no caso de T pequenos tenderá à capacidade máxima de buffers do sistema (10). Para explicar este caso se estuda o sistema de $\mu_1 = 0.125$ [tarefas /seg] e $\mu_2 = 0.375$ [tarefas /seg]. Para calcular o número médio de tarefas no centro se usa a equação VI.1.

O tempo de resposta para este caso se comporta de forma semelhante ao explicado para o caso homogêneo no ponto VI.5, entretanto o fenômeno é mais acentuado para o sistema heterogêneo. Pode ser visto, na figura VI.61, que o menor tempo de resposta é obtido para o sistema homogêneo. Os valores maiores são obtidos para o sistema heterogêneo $\mu_1 = 0.125$ [tarefas /seg] e $\mu_2 = 0.375$ [tarefas /seg].

As figuras VI.62 e VI.63 mostram o percentual de tempo que a fila lenta e rápida, respectivamente, permanecem saturadas no intervalo de tamanho T.

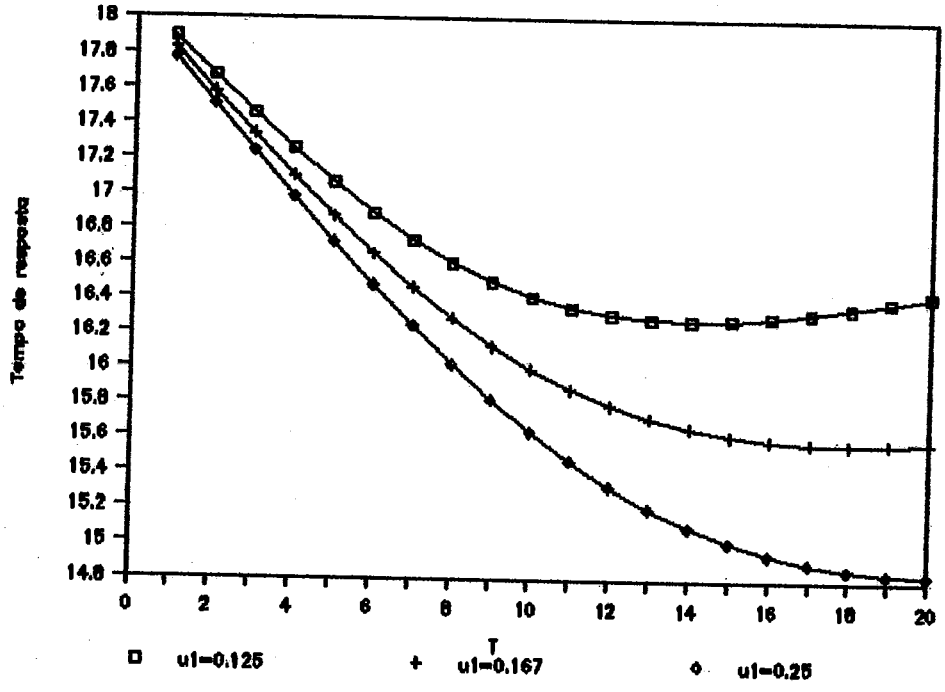

Figura VI.61: Tempo de resposta vs T para servidores heterogêneos com capacidade total de serviço 0.5

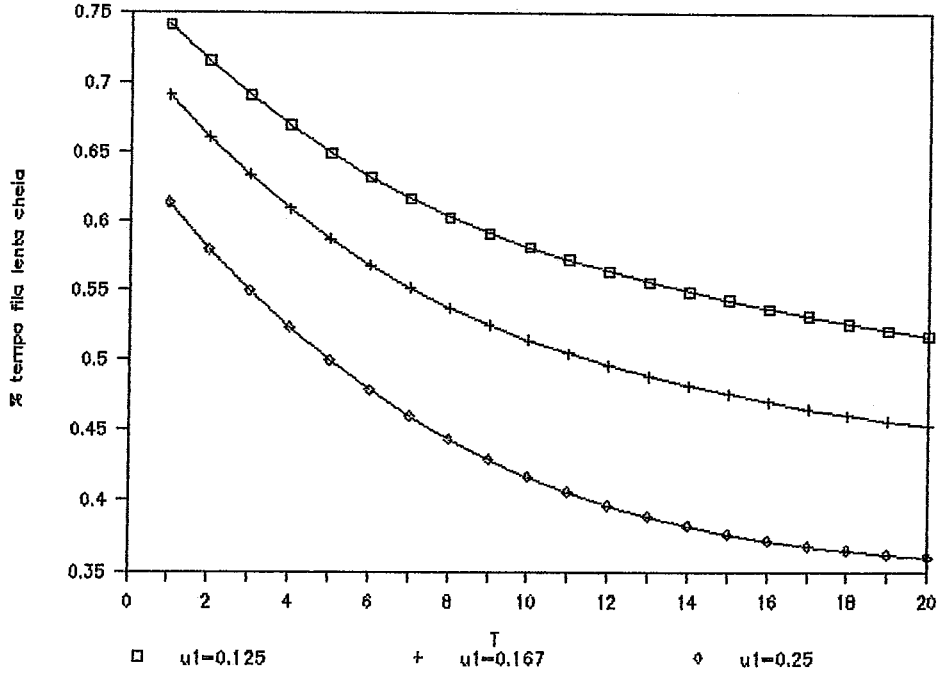

Figura VI.62: Percentual de tempo que a fila lenta permanece saturada vs T para servidores heterogêneos com capacidade total de serviço 0.5

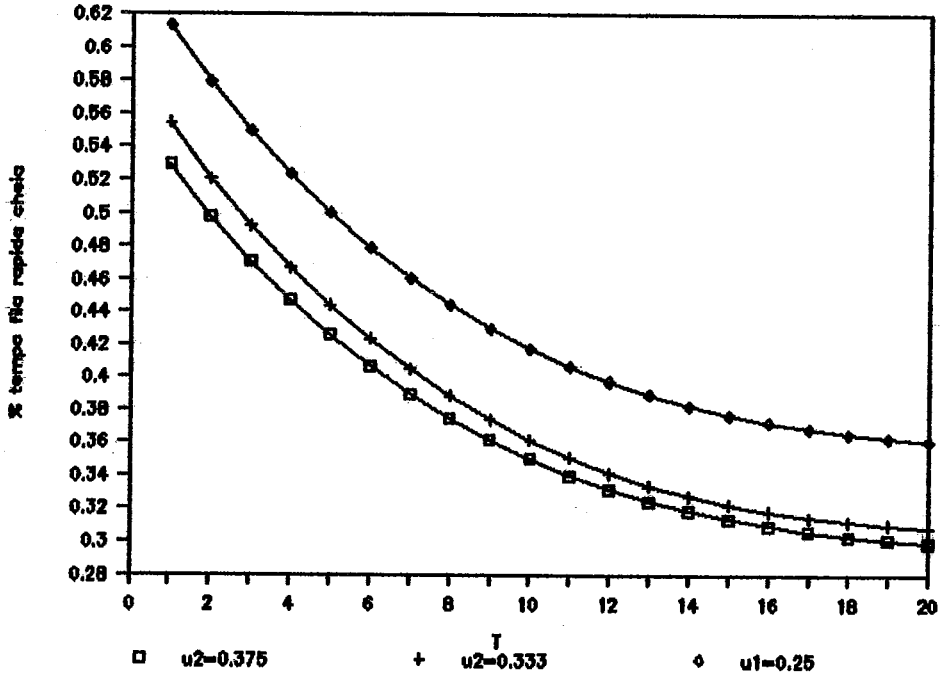

Figura VI.63: Percentual de tempo que a fila rápida permanece saturada vs T para servidores heterogêneos com capacidade total de serviço 0.5

Neste experimento a capacidade total de serviço do sistema é 2, ou seja em valor médio para o sistema o $\rho = 0.5$

- $\mu_1 = 1$ [tarefas /seg]
- $\mu_2 = 1$ [tarefas /seg]

- $\mu_1 = 0.67$ [tarefas /seg]
- $\mu_2 = 1.33$ [tarefas /seg]

- $\mu_1 = 0.5$ [tarefas /seg]
- $\mu_2 = 1.5$ [tarefas /seg]

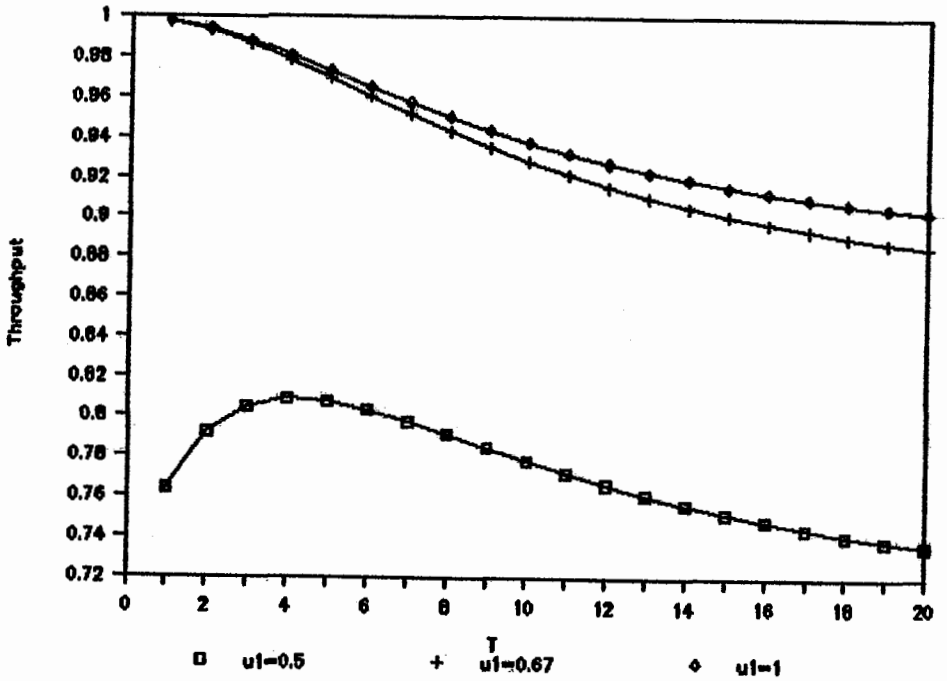

Figura VI.64: Vazão média vs T para servidores heterogêneos com capacidade total de serviço 2

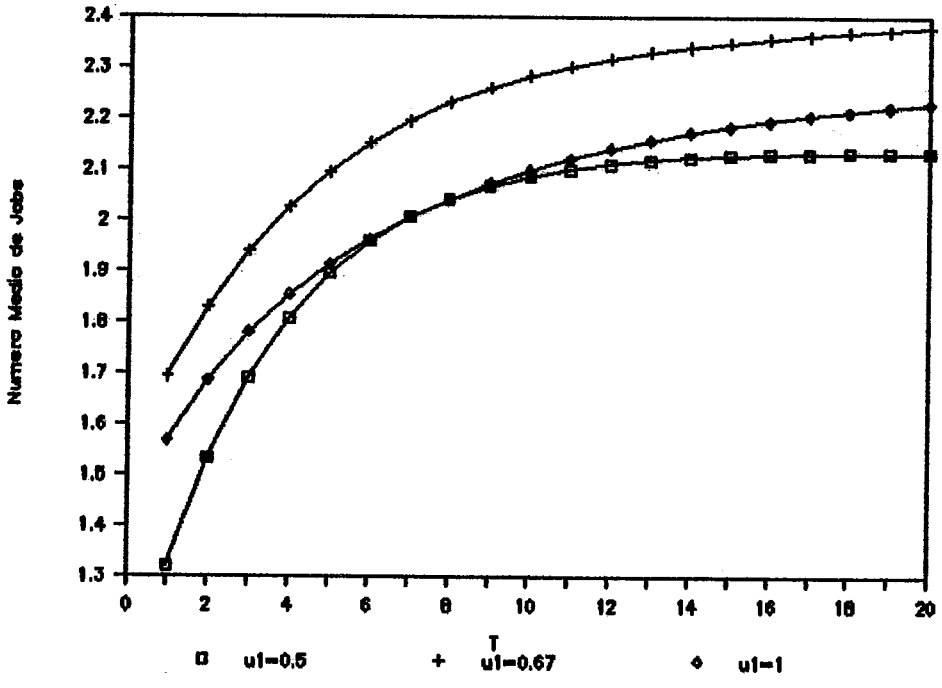

Figura VI.65: Número médio de tarefas no sistema vs T para servidores heterogêneos com capacidade total de serviço 2

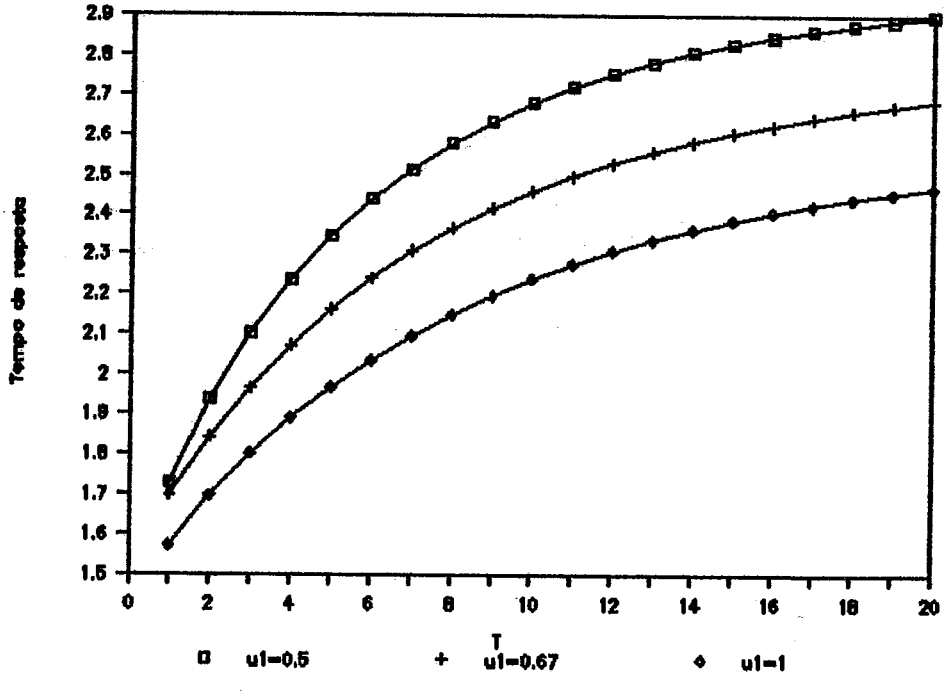

Figura VI.66: Tempo de resposta vs T para servidores heterogêneos com capacidade total de serviço 2

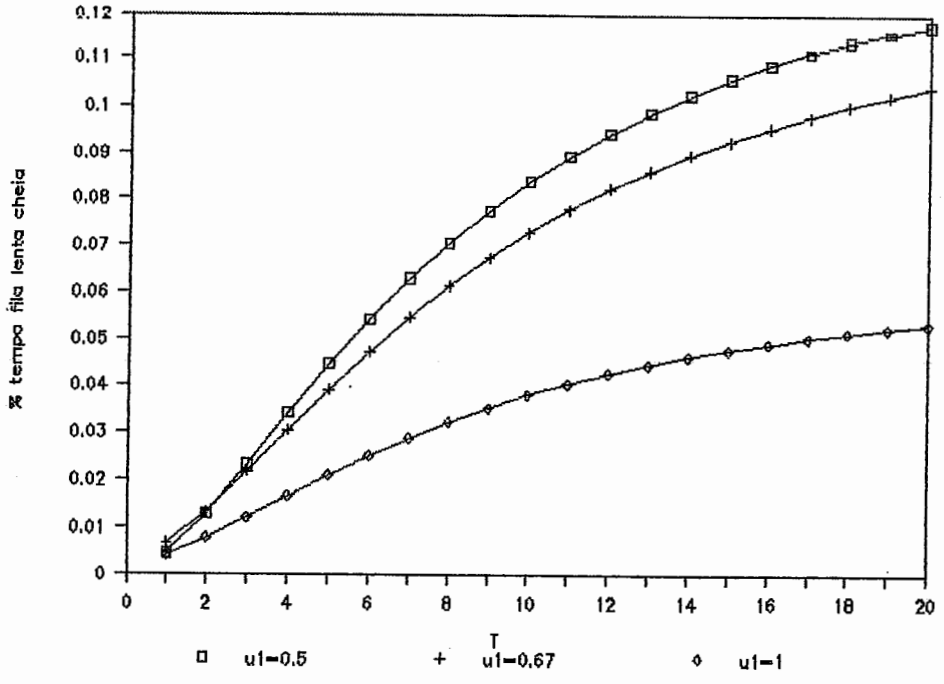

Figura VI.67: Percentual de tempo que a fila lenta permanece saturada vs T para servidores heterogêneos com capacidade total de serviço 2

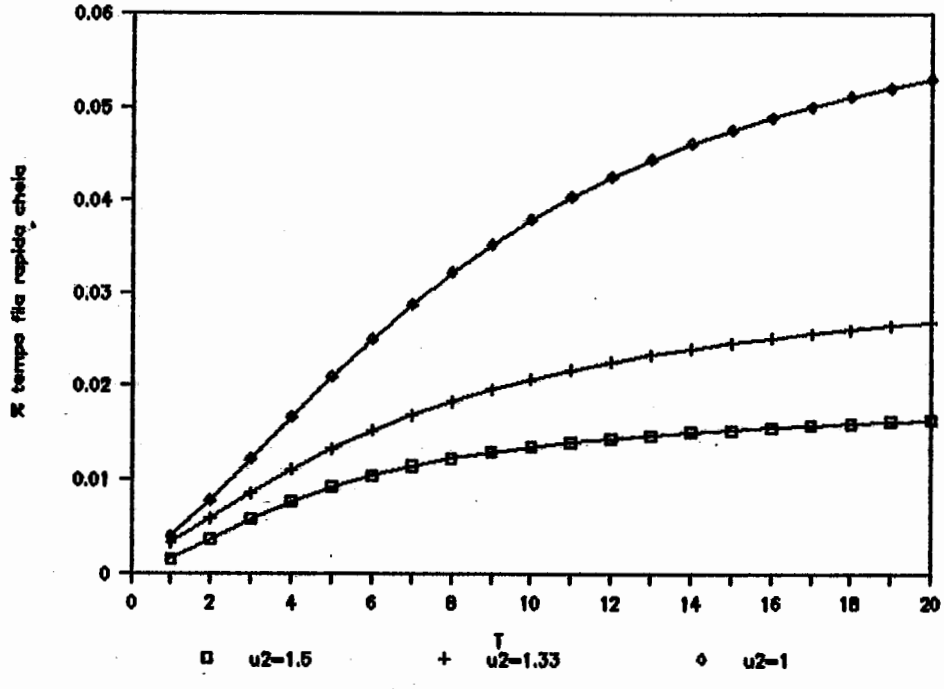

Figura VI.68: Percentual de tempo que a fila rápida permanece saturads vs T para servidores heteregêneos com capacidade total de serviço 2

VI.12 Experimento para 3 centros

Para três centros, os parâmetros usados são:

- Modelo 3
- Regra de balanceamento para 3 centros ($p_1 = 1$)
- buffer de cada servidor = 4 [tarefas]
- $\mu = 1$ [tarefa/seg]
- $\lambda = 3$ [tarefas /seg]; ($\rho = 1$)

Nas figuras VI.69, VI.70 e VI.71 são apresentadas os resultados obtidos para um sistema com três centros. Pode-se ver que o comportamento, para este sistema, é similar ao comportamento obtido para um sistema com dois centros que usa a regra de balanceamento 1 com $p_1 = 1$

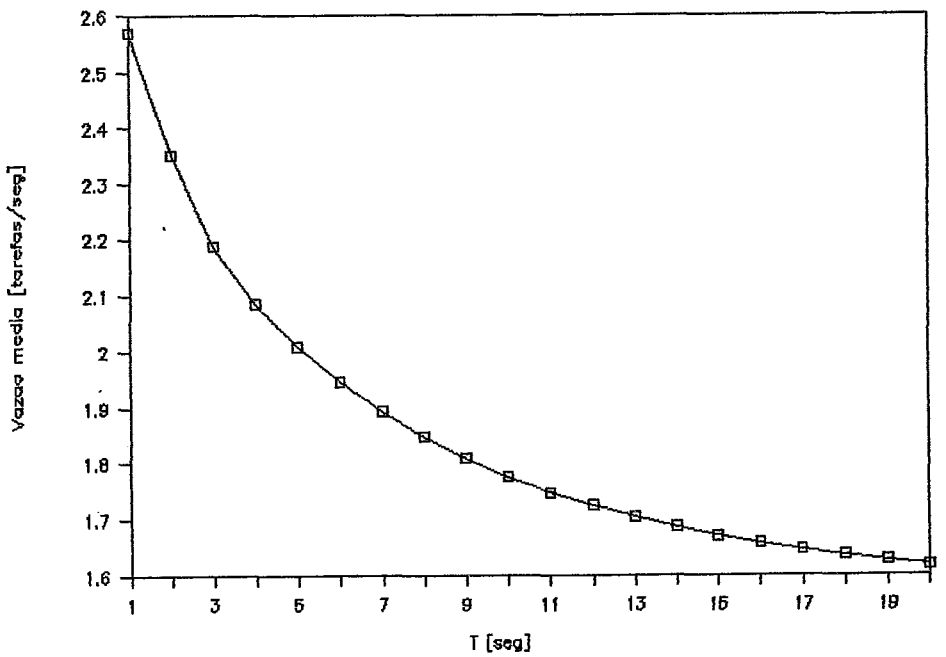

Figura VI.69: Vazão média no sistema

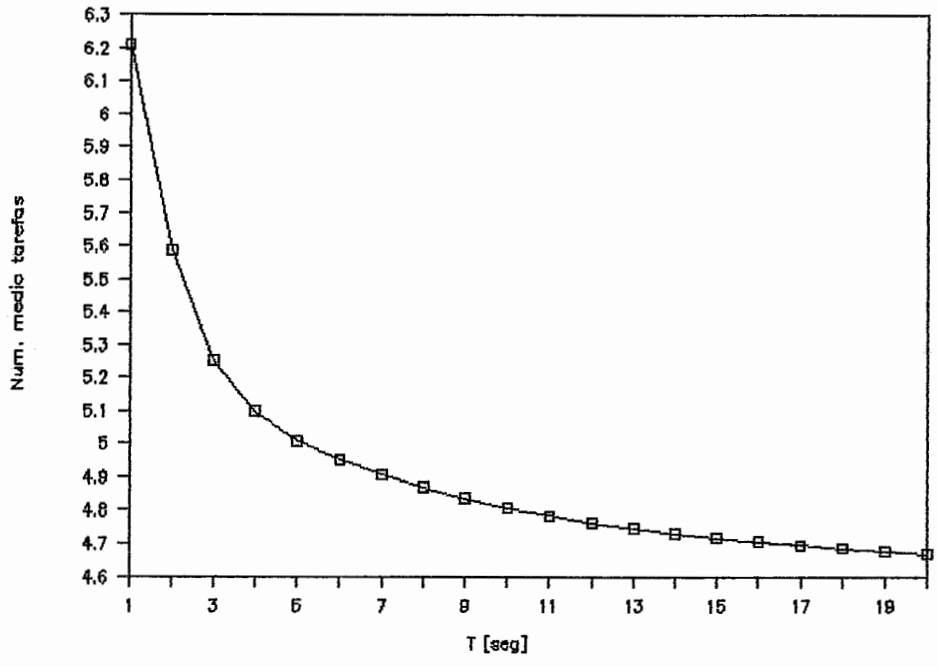

Figura VI.70: Número médio de tarefas no sistema

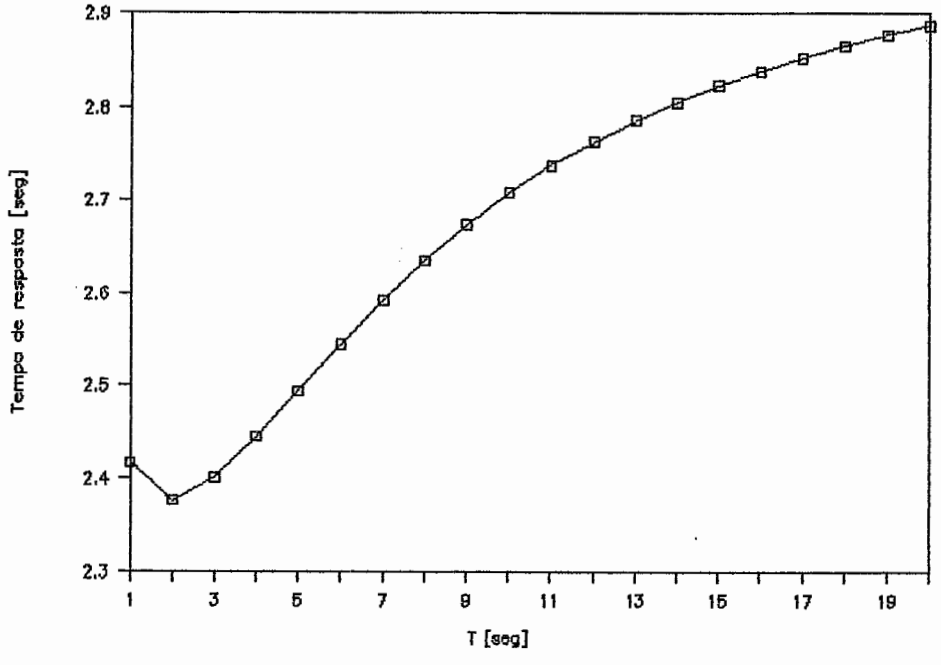

Figura VI.71: Tempo de resposta médio no sistema

Capítulo VII

Conclusões

Dos resultados numéricos, obtidos ao analisar diferentes regras de balanceamento e apresentados no capítulo VI, pode-se notar que:

- O melhores resultados são obtidos obviamente para T pequenos, ou seja T tais que o gerente sabe com maior precis ao qual o centro ao qual pode enviar tarefas.
- A vazão do sistema cai a medida que os tempos entre atualizações de estado aumentam, devido ao efeito de usar informação desatualizada.
- O número médio de tarefas no sistema e o tempo médio de resposta aumentam com o aumento de T, devido, também, ao fato de que o gerente usa informação de estado desatualizada.
- Em geral, quando aumenta a taxa de chegadas de tarefas ao sistema (mantendo constante a taxa de serviço), o tempo de resposta aumenta. Isto se deve ao aumento da utilização do sistema.

  A vazão se mantém quase constante para taxas de chegadas baixas em relação a $\mu = 1$ ($\lambda = 0.5, 0.6$) qualquer que seja o valor de T. E, decai com T para taxas de chegadas maiores ($\lambda = 0.7, 0.8, 0.9$).

  O custo de se atualizar a informação de estado é inversamente proporcional ao comprimento do intervalo T. Portanto, se a utilização do sistema é baixa, valores grandes de T podem ser usados. Quando a utilização aumenta, é importante que se atualize mais freqüentemente as informações de estado. Isto sugere a importância de se estudar políticas que possam ajustar dinâmicamente T com a carga do sistema.

  Para $\lambda \geq 1$ a vazão tem seu valor máximo para T=0 e cai, rapidamente, quando T aumenta, até alcançar um valor estacionário.

  O número médio tem seu valor mínimo para T próximo de zero e aumenta quando aumenta T.

Para $\lambda = 2$ (situação saturada), existem situações diferentes das explicadas anteriormente, por ser o $\rho = 1$. A vazão cai muito rapidamente até para T=8 e para valores maiores de T tem uma taxa de caída quase constante. O número médio de tarefas no sistema tem seu valor máximo para T=0 e decai para valores até T=8, mas esta velocidade de diminuição é menor que a velocidade de diminuição da vazão, o que fornece um tempo de resposta mínimo para T=3, devido ao efeito de limitação de buffer. Para valores de $T > 8$, o número médio tende a um valor estacionário, enquanto a vazão continua diminuindo, o que faz com que o tempo de resposta aumente, tendendo ao mesmo comportamento do caso geral. Este tipo de comportamento do tempo de resposta, (ver figura VI.12), para $\lambda = 2$ se verifica, também, com a política 1 e probabilidades de atribuição variando entre 1,..0.7. Já para $p_1 = 0.5$ e 0.6 o comportamento é igual ao descrito para $\rho < 1$.

- Notou-se que a probabilidade de atribuição que maximiza vazão (p^*), depende do tamanho de T. Pode-se ver na figura VI.26 que para $T = 0.5$ 1 o $p^* = 1$, para $T = 5$ $p^* = 0.7$, para $T = 10$ $p^* = 0.6$ e $T > 14$ $p^* = 0.5$. Para $\lambda = 2$ pode ser notado que os valores especificados acima não mudam.

- Quando o tamanho do buffer aumenta também aumenta a vazão do sistema pois menos tarefas são perdidas. A sensibilidade da vazão com o tamanho do buffer é maior para T maior.

  O tempo médio de resposta aumenta a medida que aumenta o tamanho do buffer, isto porque ao aumentar o buffer mais tarefas ficam em fila, esperando serviço.

  A sensibilidade do número médio de tarefas com o tamanho do buffer é maior para T maior.

  A medida que o buffer aumenta cai o percentual de tempo no qual a fila fica cheia, para um determinado T.

- Para qualquer T, quando a taxa de serviço do sistema homogêneo aumenta, obviamente, diminue o tempo de resposta.

  A vazão tem um comportamento similar para as taxas de serviço analisadas, até T=3. Para $T > 3$ a vazão cai na medida que T aumenta.

  Para os sistemas mais rápidos ($\mu = 2$, $\mu = 10$), o número médio permanece constante, independentemente de T.

- A política 2 tem sempre um comportamento pior que a política 1 com p^*. (Ver figuras VI.48, VI.49 e VI.50)

- Para a política 3, existe para cada T um $nmax$ que maximizará a vazão. Para $T < 4$ o $nmax = 4$. Para $4 < T \geq 11$ o $nmax$ que entrega a maior vazão é 6, e para $T > 11$ o $nmax = 8$. Indicando que quanto mais atrasada for a informação de estado maior o número de tarefas que devem ser executadas localmente. Um estudo a ser feito consiste na determinação de $nmax$ na presença de servidores heterogêneos.

- Para servidores heterogêneos, com a mesma capacidade total de processamento, vê-se que aquele que tem uma diferença menor entre sua taxas de serviço é o que tem maior vazão, menor tempo de resposta. O servidor homogêneo (diferença entre as taxas de serviço de seus servidores igual a zero), com mesma capacidade total de serviço é o que tem melhor desempenho. Pode ser visto que, para ρ médio do sistema igual a 2, se apresenta o mesmo fenômeno que no caso homogêneo, só que neste caso o fenômeno é mais acentuado.

A ferramenta computacional implementada neste trabalho, foi projetada para sistemas que possam ser representados por cadeias de Markov com qualquer número de estados. Entretanto, existe o problema de explosão do espaço de estados da cadeia de Markov.

Mesmo utilizando sistemas 'pequenos' várias conclusões puderam ser obtidas. Para modelos maiores, modelos aproximados poderão ser desenvolvidos. Neste caso, a ferramenta implementada neste trabalho (que fornece valores exatos para as medidas), serveria para validar modelos aproximados.

Referências Bibliográficas

[1] *VM/Programming in Logic, Program Description and Operation Manual.* IBM, 1985.

[2] Arnold O. Allen. *Probability, Statics, and Queueing Theory with computer science applications.* Academic Press, New York, 1978.

[3] S. Berson, E. de Souza e Silva, and R. R. Muntz. An object oriented methodology for the specification of markov models. *First Internatinal Conference on Numerical Solution of Markov Chains,* 1990.

[4] F. Bonomi and A. Kumar. Adaptative optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Transactions on Computers,* 39(10):1232–1250, October 1990.

[5] Y. Chow and W. Kohler. Models for dynamic load balancing in hetereogeneus multiple processor systems. *IEEE Transactions on Computers,* C-28(5):354–361, May 1979.

[6] E. de Souza e Silva and H. R. Gail. Calculating availability and performability measures of reparaible computer systems using randomization. *Journal of the ACM,* 36(1), January 1989.

[7] E. de Souza e Silva and H. R. Gail. Scheduled maintenance policies of repairable computer systems. *IEEE Transactions on Computers,* 1990.

[8] E. de Souza e Silva and M. Gerla. Queueing network models for load balancing in distributed systems. *Journal of the Parallel and Distributed Computing,* 12, 1991.

[9] Morgana C. Diniz. *Uma Ferramenta para Especificação e Geração de Modelos Markovianos Baseada numa Metodologia Orientada a Objeto.* Tese Mestrado, Universidade Federal do Rio de Janeiro, 1990.

[10] D. Eager, E. Lazowska, and J. Zahorjan. Adaptative load sharing in homogeneus distributed systems. *IEEE Transactions on Software Engineering,* SE-12(5):662–675, May 1986.

[11] Donald Gross *et al.* The randomization technique as a modeling tool and solution procedure for transient markov processes. *Operations Research,* April 1984.

[12] E. Shapiro *et al. The Art of Prolog.* Mit-Press, Massachusetts, 1986.

[13] A. Livny and M. Melman. Load balancing in homogeneous broadcast distributed systems. *Performance Evaluation Review*, 11(1), April 1982.

[14] R. Mirchandaney, Towsley D., and J. Stankovic. Analyis of the effects of delays on load sharing. *IEEE Transactions on Computers*, 38(11), November 1989.

[15] Sheldom M.Ross. *Probability Models.* Academic Press, INC, 4 edition, 1989.

[16] L.M. Ni and K. Hwang. Optimal load balancing in a multiprocessor systems with many jobs classes. *IEEE on Software Engeneering*, SE-11(5):491–495, May 1985.

[17] Keith Ross and David. Yao. Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM*, 38(3):676–690, July 1991.

[18] K. Shin and Y. Chang. Load sharing in distributed real-time systems with state-changed broadcast. *IEEE Transactions on Computers*, 38(8), August 1989.

[19] A. Tantawi and D Towsley. Optimal static load balancing in distributed computer systems. *JACM*, 32(2):445–465, April 1985.

[20] Y. Wang and R. Morris. Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3), March 1985.

Apêndice A

Exemplo de uso da ferramenta

A.1 Ambiente Prolog

Para iniciar esta etapa o interpretador prolog VM/PROLOG tem que ser carregado, através do comando :

VMPROLOG GS 5100K LS 3500K TRAIL 800K

A memória virtual definida para o interpretador será dividida estaticamente entre as três pilhas como pode ser visto no comando anterior.

Para definir a quantidade de memória virtual, o usuário deve usar os seguintes comandos :

def stor 10m
cp ipl cms

Estes comandos definem 10 megabytes de memória que é o máximo permitido para um usuário.

Quando o interpretador Prolog está carregado, o programa relativo ao exemplo que se quer estudar, tem que ser consultado como se segue :

$< -CONSULT(EXEMPLO1).$

Ao fazer isto serão carregadas, na memória, as cláusulas correspondentes aos objetos que definem um sistema, como foi explicado no capítulo IV desta tese.

Para gerar a Cadeia de Markov correspondente a este exemplo deve escrever-se:

< – EXPAND(*).

Em seguida depois a mensagem **fail** será apresentada indicando o término da geração da árvore de alcançabilidade. A matriz de transição estará armazenada nos arquivos RATES PROLOG A e STATES PROLOG A. Logo após obter a matriz de transições deve ser gerada a lista de parâmetros que servirão para avaliar as Cadeias de Markov. Cada um destes arquivos que gera a lista de parâmetros deve ter uma cláusula indicando qual o arquivo que contem as regras de balanceamento sendo avaliada. (< – RECONSULT(BALANC).). Para obter a lista de parâmetros deve ser consultado o arquivo LIST_PAR adequado ao caso, da seguinte maneira:

< – RECONSULT(LIST_PAR).
< – LIST_PAR(*).

A lista de parâmetros estará armazenada no arquivo LP PROLOG A.

Depois deverão ser consultados os outros programas, que produzem os arquivos de entrada necessários para a ferramenta que avalia a política de balanceamento e obtem as medidas de desempenho no ambiente Pascal.

Para obter o número total de jobs em cada um dos estados, deve seguir-se os seguintes passos:

< – RECONSULT(SOMANUM).
< – SOMANUM(*).

O número de jobs ficará no arquivo NUMEST PROLOG A.

Para obter quais os estados que possuem centros com fila totalmente cheia, deve consultar-se o arquivo:

< – RECONSULT(GERA_FCH).
< – GERA_FCH(*).

O arquivo de saida é FCH PROLOG A.

O comando $< -FIN.$ tem que ser usado para sair do interpretador PROLOG.

A.2 Ambiente Pascal

Para iniciar esta etapa deve-se, primeiro, definir na seção CONST, os parâmetros a serem usados, tais como:

QDim representa o número de estados da matriz de transição.

TimeUnit é a unidade de tempo entre atualizações da informação de estado.

PointsNumber número de pontos a avaliar. A multiplicação de TimeUnit * PointsNumber fornece os tamanhos de intervalo simultâneos que serão avaliados com a ferramenta. Por exemplo, se TimeUnit = 10 e PointsNumber = 2, então o sistema será avalido para dois tamanhos de intervalo entre atualizções da informação de estado, um serde 10 [seg] e o outro de 20 [seg].

PathI especifica o lugar onde a data de entrada se encontra.

PathO especifica o lugar onde a data de saida deverá ser colocada.

Nmedio especifica o nome do arquivo de saida do valor do número médio de tarefas.

ThrMedio especifica o nome do arquivo de saida para a vazão média.

TRespMedio especifica o nome do arquivo de saida para o tempo médio de resposta.

MatrizD arquivo de saida para a matriz de transições da cadeia nos pontos embutidos.

Populacao especifica o nome do arquivo de entrada, obtido no ambiente Prolog, e que cont'em o número de tarefas de cada estado.

TransMarcadas especifica o nome do arquivo de entrada, que contém as transições marcadas.

CM nome do arquivo onde se encontram as cadeias de Markov. Por exemplo: 'CM.'.

Uma vez definidos os parâmetros, o programa deve ser compilado e rodado.