UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

INVESTIGAÇÃO E DESENVOLVIMENTO DE UM FRAMEWORK PARA TESTES DE ESTRATÉGIAS DE CONTROLE AUTOMÁTICO VISANDO MELHORIA DE DESEMPENHO EM SERVIDORES WEB APACHE

MARCOS VINICIUS SADALA BARRETO

TD 20/2019

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MARCOS VINICIUS SADALA BARRETO

INVESTIGAÇÃO E DESENVOLVIMENTO DE UM FRAMEWORK PARA TESTES DE ESTRATÉGIAS DE CONTROLE AUTOMÁTICO VISANDO MELHORIA DE DESEMPENHO EM SERVIDORES WEB APACHE

TD 20/2019

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MARCOS VINICIUS SADALA BARRETO

INVESTIGAÇÃO E DESENVOLVIMENTO DE UM FRAMEWORK PARA TESTES DE ESTRATÉGIAS DE CONTROLE AUTOMÁTICO VISANDO MELHORIA DE DESEMPENHO EM SERVIDORES WEB APACHE

Tese submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Doutor em Engenharia Elétrica na área de Computação Aplicada

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

Dados Internacionais de Catalogação - na – Publicação (CIP) Sistema de Bibliotecas da UFPA

B273i Barreto, Marcos Vinicius Sadala, 1980-

Investigação e desenvolvimento de um framework para testes de estratégias de controle automático visando melhoria de desempenho em servidores web apache / Marcos Vinicius Sadala Barreto.-2019.

Orientador: Walter Barra Júnior

Tese (Doutorado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2019.

1. Controladores programáveis – Testes. 2. Framework (Programa de computador). 3. Software – Desenvolvimento – Testes. 4. Recursos de redes de computadores. I. Título.

CDD: 23 ed. 629.895

Elaborada por Lucicléa S. de Oliveira – CRB -2/648

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

INVESTIGAÇÃO E DESENVOLVIMENTO DE UM FRAMEWORK PARA TESTES DE ESTRATÉGIAS DE CONTROLE AUTOMÁTICO VISANDO MELHORIA DE DESEMPENHO EM SERVIDORES WEB APACHE

AUTOR: MARCOS VINICIUS SADALA BARRETO

TESE DE DOUTORADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADDA ADEQUADA PARA OBTENÇÃO DO GRAU DE DOUTOR EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA

Aprovado em: __/__/____

BANCA EXAMINADORA

_____
Prof. Dr. Walter Barra Junior
Orientador (UFPA/ PPGEE)

_____
Prof. Dr. Carlos Tavares da Costa Junior
Avaliador Interno (UFPA/ PPGEE)

_____
Prof. Dr. Marcelino Silva da Silva
Avaliador Interno (UFPA/ PPGEE)

_____
Prof. Dr. Fabrício Gonzalez Nogueira
Avaliador Externo (UFC/ PPGEE)

_____
Prof. Dr. Raimundo Nonato das Merces Machado
Avaliador Externo (IFPA)

AGRADECIMENTOS

Ao meu filho, que somente com a sua presença engrandece o sentimento de conquista e realização com intuito de formar o seu caráter e provar que tudo é possível, basta você se dedicar plenamente.

À minha esposa, que participou de todas as dificuldades nesta caminhada, sendo todas superadas com amorosidade e cumplicidade.

À minha mãe, que dedicou a sua vida à minha educação, com horas ensinando uma lição da escola, ou com dias de trabalho para pagar um curso ou com anos de paciência mostrando verdades não aceitáveis por um jovem que achava que sabia de tudo.

Ao amigo, professor e colega de trabalho, Doutor Petrônio Vieira Junior, que, nos momentos de conflitos, teve a sabedoria e a paciência para aconselhar de forma a promover o conciliar. Isso tudo promovendo meu crescimento no mundo acadêmico.

E, finalmente, ao ilustre professor titular Doutor Walter Barra Junior, que sempre acreditou e proporcionou o desenvolvimento deste trabalho, dando o apoio necessário para à conclusão desta etapa de vida deste discente e colega de profissão. Este discente não tem palavras para adjetivar a grandeza do caráter do professor Walter Barra; mas, posso dizer que já ultrapassou o nível dos melhores professores. Professor? Ele já deixou de sê-lo, pois aquele que incute e desenvolve confiança é honesto e transparente, é ético, é modelo para conduta moral, é educado, cortês, calmo e não se importa com provocações, é grato por todas as coisas e é amado por seus alunos, é um *sensei*. Obrigado *sensei* Barra.

SUMÁRIO

1 Definição do Problema / Estado da Arte..... 14

1.1 - DEFINIÇÃO DO PROBLEMA..... 14

1.2 - ESTADO DA ARTE..... 16

1.3 - OBJETIVOS..... 18

1.4 - ESTRUTURA DO TRABALHO..... 18

2 Apache..... 20

2.1 – SISTEMA COMPUTACIONAL DE RESPOSTA "HTTP"..... 23

2.2 – PARAMETRIZANDO O SERVIDOR APACHE..... 28

3 Modelagem do Sistema Computacional "Apache"..... 31

3.1 - LINEARIZAÇÃO DO SISTEMA FENOMENOLÓGICO..... 39

3.2 - MODELO NÃO LINEAR SIMPLIFICADO..... 41

3.3 – PI CLÁSSICO..... 43

3.4 – CONTROLADOR INTERVALAR..... 46

4 Desenvolvimento do Ambiente de Trabalho..... 49

4.1 AMBIENTE DE TRABALHO..... 50

4.2 O SOFTWARE..... 54

5 Resultados..... 69

6 Conclusão..... 85

7 Referências Bibliográficas..... 87

Anexos I..... 89

Anexo II..... 110

LISTA DE ILUSTRAÇÕES

Figura 2.1 -Conexão “HTTP”..... 20

Figura 2.2 -Arquitetura do servidor “Apache”.....24

Figura 2.3 -Servidor “Apache” no Sistema Operacional.....26

Figura 2.4 -Lista de processos do servidor “Apache”.....27

Figura 2.5 -Hereditariedade dos processos do Servidor Apache.....27

Figura 2.6 -Serviço funcionando sem a presença do processo pai.....27

Figura 2.7 -Conteúdo do arquivo mpm_prefork.conf.....29

Figura 3.1 -Esquema da fila do sistema computacional “Apache”.....32

Figura 3.2 -Modelo fenomenológico..... 35

Figura 3.3 -Variação do “MaxClient” no modelo fenomenológico.....36

Figura 3.4 -Tempo de reposta do servidor “web”.....36

Figura 3.5 -Número de processos na fila do Servidor “Apache”.....37

Figura 3.6 -Número de requisições na fila externa..... 37

Figura 3.7 -Tempo de resposta do sistema computação..... 38

Figura 3.8 -Taxa de requisição do servidor “Apache”..... 38

Figura 3.9 -Variação do percentual de memória utilizada pelo servidor “Apache”..... 39

Figura 3.10 -Modelo fenomenológico simplificado Não Linear..... 42

Figura 3.11 -Modelo fenomenológico simplificado Linear..... 42

Figura 3.12 -Controlador PI nominal do modelo fenomenológico.....44

Figura 3.13 -Atuação do controlado PI nominal nos três pontos de operação.....45

Figura 3.14 -Degrau na referência no sistema fenomenológico em malha fechada com controlador PI nominal.....45

Figura 3.15 -Atuação do controlador intervalar na planta fenomenológica.....48

Figura 3.16 -Atuação do controlador clássico nominal..... 48

Figura 4.1 -Relações dos Sistemas.....49

Figura 4.2 -Máquinas virtuais instaladas.....51

Figura 4.3 -Esquema de estrutura de máquina real e virtual..... 51

Figura 4.4 -Recursos computacionais da máquina virtual servidor gerador de carga..... 53

Figura 4.5 -Recursos computacional da máquina virtual servidor “Apache”..... 53

Figura 4.6 -Recursos computacionais da máquina virtual banco de dados..... 54

Figura 4.7 -Esquema de escrita e leitura no *pipe*.....55

Figura 4.8 -Diagrama de fluxo de dados de alto nível.....58

Figura 4.9 -Diagrama de fluxo de dados de baixo nível.....59

Figura 4.10 -Comando “free”.....60

Figura 4.11 Linha de código controlador PI.....60

Figura 4.12 Gerações diferentes em execução..... 61

Figura 4.13 Fluxograma geral..... 62

Figura 4.14 Diagrama de transição de estado do processo controlador clássico..... 67

Figura 4.15 Diagrama geral do sistema computacional..... 68

Figura 5.1 -Valores dos sinais de entrada para a identificação paramétrica.....70

Figura 5.2 -Quantidade de memória por unidade.....71

Figura 5.3 -Porcentagem de CPU utilizada.....71

Figura 5.4 -Variação e resposta do sistema “httpd”.....72

Figura 5.5 -Ampliação 3º transitório.....72

Figura 5.6 -Ampliação 5º transitório.....73

Figura 5.7 -Sinal SBPA projetado.....75

Figura 5.8 -Sinal SBPA projetado com estado inicial da planta.....75

Figura 5.9 -Resposta ao sinal SBPA projetado.....76

Figura 5.10 -Autocorrelação e coeficiente de correlação cruzada.....77

Figura 5.11 -Comparação modelo obtido e sistema real.....77

Figura 5.12 -Esquema do controlador retroalimentado..... 78

Figura 5.13 -Esquema da simulação do sistema em malha fechada..... 79

Figura 5.14 -Resposta da simulação..... 80

Figura 5.15 -Sinal de Erro da simulação em malha fechada.....80

Figura 5.16 -Sinal de controle da simulação em malha fechada.....81

Figura 5.17 -Resposta do sistema real em malha fechada.....82

Figura 5.18 -Sinal de controle em malha fechada.....82

LISTA DE TABELAS

Tabela 1.1 -Sítios mais Visitados do Mundo.....15

Tabela 3.1 -Tabela de ensaios de aproximação da instabilidade do sistema.....43

Tabela 4.1 -Máquina utilizada na experimentação.....50

Tabela 4.2 -Símbolos de notação do dicionário de dados.....63

Tabela 4.3 -Dicionário de Dados.....63

Tabela 5.1 -Parâmetros utilizados no “httperf” para a geração de carga no servidor Apache.....83

Resumo

Neste trabalho, é realizado um estudo para o desenvolvimento de um ambiente computacional para testes de controle automático em sistemas computacionais de resposta às requisições “HTTP” “Apache”. Em particular, são discutidos os detalhes de implementação e resultados parciais em um ambiente para controle digital de desempenho do sistema de resposta HTTP”. Desta forma, foi necessário projetar e desenvolver um ambiente para a coleta de dados no sistema computacional que fornece o serviço de resposta a solicitações “HTTP” e faz intervenções nos valores dos parâmetros disponíveis para controlar o sistema automaticamente em malha fechada. Os parâmetros a serem obtidos como variáveis endógenas do sistema computacional são “MaxRequestWorks” (“MaxClient”), e o exógeno é a quantidade de memória consumida. A ferramenta desenvolvida não modifica nem o código-fonte do software de resposta para as solicitações “HTTP” dos clientes nem o código do sistema operacional do computador “host”, não sendo intrusivos em relação à configuração original do sistema. Um controlador proporcional integral (PI) foi projetado para manter a média de resposta do tempo percebido de mensagens “HTTP” reguladas. Testes experimentais realizados em um servidor “Apache” mostram a melhoria obtida na resposta temporal do sistema.

PALAVRAS-CHAVE: Controlador PI, Apache, *MaxClient*.

Abstract

In this work, a study is carried out for the computational development of automatic control tests in computer response systems to “HTTP” “Apache” request. In particular, the implementation details and results in an environment for digital performance control of the HTTP response system are discussed. Thus, it was necessary to design and develop an environment for performing data collection in the computational system that provides the response service to “HTTP” requests and makes interventions in the values of the available parameters in order to control the system automatically in closed loop. The parameter to be obtained as endogenous variable of the computational system is MaxRequestWorks, and the exogenous one will be the amount of memory consumed. The tool developed modifies neither the source code of the response software to the clients’ “HTTP” requests nor the operating system code of the host computer, being non-intrusive regarding original system configuration. A proportional-integral (PI) controller was designed in order to keep the average perceived time response of “HTTP” messages regulated. Experimental tests carried out on an Apache server show the improvement obtained on the system's time response.

KEYWORDS: PI controller. Apache. MaxClient.

1 – INTRODUÇÃO

Adaptação automática às mudanças dinâmicas no comportamento de um sistema de computacionais é um assunto que ganha cada vez mais o interesse da academia e de empresas fornecedoras de *software* e serviços, já que tais sistemas têm sido tradicionalmente configurados de uma forma manual (via arquivos de configuração). Objetiva-se, portanto, oferecer um serviço confiável para os clientes, com baixo custo de operação [1]. As variáveis ambientais dos sistemas computacionais começaram a ser estudadas nos últimos anos para a otimização do desempenho dos serviços que são propostos por cada *software* desenvolvido [12].

As novas exigências contratuais, que visam tanto assegurar qualidade no serviço suprido ao usuário bem como assegurar confiabilidade de operação diante de grandes variações de carga, têm sido os principais impulsionadores do recente crescimento da pesquisa em aplicação da teoria de controle por realimentação em sistemas computacionais [1][12][3][4][5].

1.1 - DEFINIÇÃO DO PROBLEMA

Na maioria das arquiteturas computacionais, os administradores (as pessoas responsáveis pelo setor de Tecnologia da Informação - TI) disponibilizam uma única máquina para prover diversos serviços, tais como: servidor de e-mail, servidor de banco de dados, servidor de resposta às requisições "HTTP" (Hypertext Transfer Protocol) etc. Desse modo, o problema é como alocar a quantidade correta de recursos para cada serviço disponível sem prejudicar a qualidade do outro, levando-se em consideração a carga que está sendo aplicada. Esse é um problema difícil de lidar através de técnicas convencionais de sintonia manual de arquivos de configuração [1] [7] [8].

Um dos principais sistemas de informação utilizados atualmente, também chamados de servidores "web", é aquele que proporciona o serviço de resposta às requisições "HTTP". O "HTTP" é o protocolo de comunicação utilizado para sistemas de hipermídia, distribuídos e colaborativos, e que é a base do "*WWW*" (*World Wide Web*), o sistema eletrônico que dá acesso aos sítios eletrônicos [1][2][11].

Particularizando para o caso de otimização de desempenho dinâmico em servidores "web", as configurações de quantidade de memória e demais parâmetros influenciadores de desempenho são realizadas manualmente por meio de arquivos com extensão "txt". Ou seja, deve-se realizar os devidos cálculos de uso de recurso, localizar o respectivo arquivo de

configuração e linha pertinente, realizar as devidas alterações, salvar os arquivos e reiniciar o serviço.

A seleção das configurações corretas para os parâmetros de ajustes não é uma tarefa simples. Os melhores ajustes irão depender do tipo de *hardware*, das cargas de trabalho e de todos os trabalhos simultaneamente em execução na máquina hospedeira. O tipo de *hardware* e os trabalhos sendo executados simultaneamente são planejados, e em consideração às alterações de cargas de trabalho, podem ser considerados estáticos. Entretanto, as cargas dos trabalhos mudam ao longo do tempo conforme a imprevisibilidade do comportamento humano [8].

Softwares que são executados em concorrência podem provocar o aumento do tempo de latência. Por exemplo: um verificador de vírus, quando faz a busca no sistema operacional, causa uma demora na resposta de outro *software* que se deseja executar. Isso é uma pequena escala de um único usuário solicitando requisições [9]. Agora, se for considerado um servidor "web" que, dependendo da popularidade do sítio eletrônico, pode chegar a muitas requisições concorrentes, tem-se um problema crítico para garantir a qualidade dos serviços que devem ser providos. Para exemplificar, a Tabela 1.1 [6] mostra a quantidade de acessos em um único dia de alguns dos sítios eletrônicos mais visitados do mundo.

Tabela 1.1 - Sítios mais Visitados do Mundo(2013)

Sítio Eletrônico	Quantidade de Acessos por Dia
Facebook.com	*836,7 milhões de visitantes únicos*
Google.com	*782,8 milhões de visitantes únicos*
Youtube.com	*721,9 milhões de visitantes únicos*
Yahoo.com	*469,9 milhões de visitantes únicos*
Wikipedia.com	*469,6 milhões de visitantes únicos*

O sítio eletrônico do facebook.com, por exemplo, distribuindo uniformemente a quantidade de acessos durante o dia, tem um total de 9.684 usuários concorrentes por segundo, o que terá a carga média de consumo de 9.684 requisições por segundo nos servidores desse sítio.

Entretanto, uma notícia importante pode alterar drasticamente o comportamento do sítio facebook.com, ou de qualquer outro sítio eletrônico. Por exemplo, os sítios da Universidade Federal do Pará e do Instituto Federal de Educação, Ciência e Tecnologia do Pará, quando divulgam a lista dos aprovados no vestibular ou outros processos seletivos, têm um aumento elevadíssimo no número de visitantes.

Para lidar com essas grandes variações, uma solução convencional tem sido o emprego de recursos de hardware superdimensionados em relação às demandas normais de carga, o que resulta em desperdício de recursos e encarecimento do serviço provido. Dessa forma, para otimizar o consumo de recursos, os algoritmos de controle automático avançado via realimentação devem ser investigados, pois é neste campo que a teoria de controle retroalimentado tem muito a oferecer.

1.2 - Estado da Arte

Em [12], é apresentada uma estratégia de controle para servidor Apache, sugerindo estratégias para a seleção de emparceiramento entrada/saída, intervalo de amostragem, identificação de modelo do sistema e quais as medidas de desempenho a serem adotadas. Também em [12] foi investigada uma estratégia "MIMO", cujo projeto de controle foi baseado em posicionamento de polos, com estudos computacionais efetuados em ambiente "Matlab".

A teoria de controle multivariável foi utilizada em sistemas computacionais para o projeto de controladores realimentados em [13]. O enfoque adotado por esses autores consistiu em relacionar as métricas de "*MaxCliente"* e "*KeepAlive"* com os recursos computacionais existentes, quantidade de memória disponível e utilização da unidade de processamento, fornecendo o benefício em lidar com o "*trade-off*"[1] entre a velocidade de convergência e a sensibilidade métrica para flutuações aleatórias, uma vez que a técnica "MIMO" utilizada foi a de regulação quadrática linear *"RQL"* (*Linear Quadratic Regulation* – LQR).

Kjaer [1] cita que a utilização limitada do processamento do buffer para o projeto de controle pode provocar efeitos críticos no sistema, uma vez que o que seria considerado como uma medida de uma perturbação acaba por ser dependente do estado, e um zero entra na função de transferência descrevendo a relação entre o sinal de controle e a saída do controle. A análise de estabilidade foi apresentada e uma nova estratégia foi desenvolvida para a melhora da robustez do controle. Toda a experimentação sendo realizada em ambiente de simulação Matlab.

Lindegren [14] realiza intrusões no código do servidor "web" Apache/2.0.52, tendo como sistema operacional o Red Hat Linux, preparando-o para inclusão de um controlador automático via realimentação. Caso haja alteração no controlador, o Apache deverá ser recompilado, sendo inviável devido ao tempo gasto para a conclusão da tarefa.

[1] Expressão que define uma situação em que há conflito de escolha. Caracterizada por uma ação que visa a resolução de um problema, mas acarreta em outro, obrigando a uma escolha. Ocorre quando se abre mão de algum bem ou serviço distinto para se obter outro bem ou serviço distinto.

O artigo [3] apresenta os resultados de estudos experimentais visando estimar modelos paramétricos autorregressivos (AR) para diferentes pontos de operação e de carga para o sistema computacional de resposta às requisições "HTTP", providos pelo "Apache". Os parâmetros considerados foram o número máximo de clientes ("MaxClient") e o percentual de utilização da CPU. O estudo mostrou que, para valores relativos baixos de MaxClient, um modelo AR de 7ª ordem pode ser satisfatório; já para valores relativos altos, modelos de ordem mais elevada são necessários. Dessa forma, foi possível modelar a dinâmica do comportamento do servidor Apache; entretanto, a mudança do MaxClient ocorreu de forma manual.

O controle do fluxo de requisições no servidor de resposta "HTTP" pode ser realizado sem a utilização de código intrusivo[2]. Em [15], utiliza-se a antecipação da análise de quantidade de processo que o servidor irá perceber. Cria-se uma fila intermediária e gerencia-se a quantidade de processos enviados ao servidor. A manipulação de porta de comunicação *socket* é essencial para o êxito do método.

Em [4], projeta-se um controlador retroalimentado clássico com ênfase em situações críticas como o *overload*. A metodologia de atuação no sistema é a de [15]. Entretanto, percebe-se que a preocupação de alocação de recursos ficou focada no sistema de resposta à requisição "HTTP", outros *software*s que estão hospedados na máquina são desprezados.

O artigo [10] mostra, brevemente, como o Kernel do Linux 2.6 realiza a distribuição do tempo de utilização do processador para cada tarefa, o agendamento de tarefas, e apresenta um *software* não intrusivo que reserva o tempo do processador para uma tarefa ou um conjunto de tarefas. Nesse trabalho, entende-se por *software* não intrusivo aquele que não altera o código fonte do Kernel do Linux 2.6. O mecanismo descrito proporcionou a separação de desempenho de servidores que são executados no mesmo *hardware*, ou seja, um servidor permanece operacional enquanto o outro servidor está sobrecarregado.

Para incrementar a importância da existência de um controle dos recursos computacionais, pode-se invocar temas de cunho administrativo e legislativo que estejam diretamente relacionados. Dentre esses, pode-se citar os Acordos de Nível de Serviços – "ANS" que cada empresa possui com seus clientes.

Um "ANS" consiste em um documento formal, negociado entre as partes, na contratação de um serviço de "TI" ou Telecomunicações. O "ANS" é colocado geralmente como anexo do contrato e tem por objetivo especificar os requisitos mínimos aceitáveis para o

[2] Código inserido no fonte de um software sendo necessário refazer compilação para a execução.

serviço proposto. O não cumprimento do "ANS" implica em penalidades, estipuladas no contrato, para o provedor do serviço.

Muitos provedores de serviço de reposta às requisições "HTTP" utilizam a estratégia de "Quality of Service – QoS" para atingir as metas estipuladas nos "ANS". O "QoS" é uma estratégia de regras que é implementada nos roteadores que priorizam determinados pacotes de tráfego na rede da corporação. A metodologia de algoritmo genético foi implementada com sucesso para adquirir os parâmetros de permanência e largura de banda para determinados pacotes de dados[5].

Entretanto, o problema da variação repentina de acesso a determinado pacote de dados hospedado em um servidor será intensificado. O motivo é que o "QoS" foi projetado para dar preferência a tráfegos com determinados critérios previamente cadastrados, multiplexando os recursos da rede. Se todos os recursos de redes forem direcionados a uma única máquina, hospedeira com pacote de dados desejado, a fila de requisições no sistema operacional crescerá e consequentemente o aplicativo de reposta "HTTP" irá ocupar mais recursos, antecipando o *"overload"*.

1.3 - OBJETIVOS

As principais contribuições cientificas deste trabalho incluem:

- Propor um *software* não intrusivo que altere as configurações do servidor Apache sem a interrupção do serviço;
- Desenvolver um controle retroalimentado que relaciona o total de memória consumida com o número máximo de processos ativos para responder requisições "HTTP";
- Desenvolver um *software* que previna o serviço de resposta "HTTP" contra os efeitos causados pelo *overload*

Espera-se que a sintonização do controlador estabilize o sistema no ponto de operação desejado, que o *software* desenvolva sua atribuição sem interrupções ou travamentos, que a parada do sistema de resposta "HTTP" não seja percebida pelos usuários e que as mudanças dos parâmetros do Apache de alocação de recursos sejam adequadas para o desenvolvimento das atividades em conformidade com o "ANS" desejável.

1.4 – Estrutura do Trabalho

No Capítulo 1, o trabalho é apresentado com a definição do problema, com a literatura de escopo de problema, com os objetivos e, finalmente, como o trabalho está estruturado.

No Capítulo 2, será pesquisada uma forma de realizar as alterações no “Apache” sem a necessidade de compilar o código do servidor e, consequentemente, realizar uma nova instalação, tendo que garantir a continuidade do serviço sem interrupções.

No Capítulo 3, o estudo em um ambiente simulado das relações entre os parâmetros endógenos e exógenos através de equações fenomenológicas é importante para o entendimento do comportamento transitório e estável do sistema. Neste estudo, será incluído o projeto e uma comparação entre um controlador PI clássico e um controlador PI intervalar.

No Capítulo 4, é desenvolvido um *software*, com todas as características desejadas, para realizar a coleta de dados do sistema computacional estudado e aplicar o controlador de teoria clássica a ser projetado a partir das amostras obtidas.

No Capítulo 5, serão mostrados os resultados da coleta de dados, o projeto de controlador PI e o comportamento do sistema computacional de resposta às requisições “HTTP” com a atuação do controlador projetado.

No Capítulo 6, é apresentam-se as conclusões da pesquisa com sugestões de trabalhos futuros.

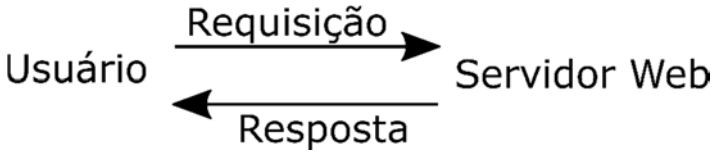
No Capítulo 7, são apresentadas as referências bibliográficas. No Anexo I, o código fonte do *software* desenvolvido e no Anexo II o script do banco de dados utilizado como repositório das informações extraídas do sistema computacional, foco da presente tese.

2 – APACHE

"Apache" é o nome de um *software* criado por Rob MCCooll em 1995, que tem por objetivo responder às requisições que chegam na máquina hospedeira com o protocolo "HTTP". O servidor "HTTP APACHE" ou servidor "Apache" é o servidor "web" livre mais bem sucedido, utilizado principalmente no sistema operacional "LINUX".

Como o sistema "Apache" faz parte de um sistema computacional maior, faz-se necessário, primeiramente, analisar como as requisições chegam ao computador hospedeiro do sistema-alvo. O servidor "HTTP APACHE" irá responder todas as conexões que utilizam o protocolo "HTTP" (*Hypertext Transfer Protocol*), o qual é responsável por transportar documentos "HTML". A Figura 2.1 é o esquema que mostra como a conexão "HTTP" é realizada.

Figura 2.1 - Conexão "HTTP"



A transferência começa com a requisição do cliente ao servidor utilizando um navegador, que estabelece uma conexão pelo método de requisição. A reposta será enviada contendo uma linha de status (versão do protocolo mais o código de sucesso ou erro), informação do servidor que respondeu à requisição, informação sobre a entidade e o possível conteúdo. Assim, pode-se dividir a transação "HTTP" em quatro etapas [18]:

1. O navegador abre a conexão;
2. O navegador envia uma requisição ao servidor;
3. O servidor envia uma resposta ao navegador;
4. A conexão é fechada.

Devido à existência do fraco acoplamento entre os protocolos de comunicação, é possível utilizar o "HTTP" em qualquer modelo de controle e transmissão desenvolvidos para redes de computadores. O mais utilizado na comunicação "HTTP" é o modelo "TCP/IP", utilizando conexões do tipo "TCP" tendo como porta padrão "TCP" 80.

O protocolo "HTTP" não registra informações a respeito das conexões. Por isso, tanto o cliente como o servidor devem ter a capacidade de fechar uma conexão prematuramente através da ação do usuário, de temporização automática ou de falha do programa. Outra situação

é que, para cada objeto existente no documento “HTML”, é criada uma conexão (e.g., se uma página possui duas figuras, serão realizadas três conexões, uma para cada figura e uma para o documento “HTML”) utilizando os recursos existentes de forma intensa, caso a página possua vários elementos, como é o caso de um bom número de páginas “web” atuais.

Essa metodologia foi empregada no “HTTP” 1.0, que foi definido no documento RFC 1945. Entretanto, a partir do “HTTP” 1.1, RFC 2068, há a redução do consumo devido proporcionar uma conexão para cada tipo de objeto, ou seja, todos os objetos de um determinado tipo que compõem um documento serão enviados através de uma mesma conexão. Outra modificação do “HTTP” 1.1 foi a criação de conexão persistente, de maneira que é criada uma única conexão “TCP” para o servidor, levando em conta que as versões anteriores do protocolo “HTTP” 1.1 geram uma conexão “TCP” para cada “URL” existente na máquina hospedeira.

As requisições que chegam à máquina hospedeira são recepcionadas pelo sistema operacional e as armazena em uma fila, ou seja, a primeira requisição será a primeira a sair para ser respondida pelo servidor “web”. Para agilizar a resposta à requisição, foram desenvolvidos nos servidores “web” os módulos de multitarefas, os quais proporcionam um processamento da requisição independentemente de outra, mesmo existindo um único processador. No “Apache”, existem dois módulos que não coexistem: “threads” e “prefork”.

O módulo multitarefa baseado em “threads”, conhecido como fio de execução ou encadeamento de execução, divide a execução do processo em várias tarefas que podem ser executadas concorrentemente (UCP[3]s ou multicores[4]) ou aparentemente concorrentes (única UCP). Os “threads” compartilham o espaço de endereçamento na memória, variáveis globais, processos filhos, arquivos abertos, sinais, tratadores de sinais, entre outros. Entretanto, contadores de programa, registradores, pilha e estados são privativos de cada “thread”.

O compartilhamento de recursos proporciona a este método um “overload” mais tardio, quando se pensa em escalonar os “threads”. Entretanto, problemas de sincronização são bem comuns ao utilizar essa técnica de programação, podendo causar comportamentos inesperados no sistema que utiliza o encadeamento de execução.

O módulo multitarefa baseado no “prefork” cria gerações de processos, ou seja, existe um processo que é denominado de pai que, ao tomar consciência de uma requisição existente na fila, cria um processo filho que a responderá. Esse processo filho é uma cópia idêntica do

[3] Unidade Central de Processamento
[4] Mais de um núcleo de processamento

processo pai; porém eles não compartilham recursos. O endereçamento de memória é diferente, variáveis globais, processos filhos abertos, sinais etc.

Os processos filhos possuem independência do processo pai e dos demais processos irmãos. Falhas que podem vir a ocorrer no processo pai ou em algum filho não irá afetar o processo que não possui falha.

Outro ponto importante em escolher o módulo de multiprocessamento "prefork" é que não existem incompatibilidades com bibliotecas incorporadas ao projeto de desenvolvimento do sítio eletrônico. Assim, qualquer sítio eletrônico pode ser hospedado, excluído-se a possibilidade do não atendimento a clientes das mais diversas tecnologias de desenvolvimento de sítios eletrônicos.

O "prefork" é o módulo mais utilizado nos servidores de resposta "HTTP" devido à sua estabilidade e compatibilidade com diversas tecnologias de desenvolvimento de sítios eletrônicos. Assim, esse será o módulo-alvo de multiprocessamento estudado nesta tese.

O multiprocessamento "prefork" possui diversos ajustes de configuração. Dentre esses, o mais importante é o que se denomina de "Maxclient". Esse parâmetro dever ser ideal para lidar com todas as solicitações simultâneas, mas pequeno o suficiente para garantir que não faltem recursos computacionais para todos os processos

Então, o parâmetro "MaxClient" define o número máximo de solicitações simultâneas que serão atendidas, o valor padrão é 256 processos filhos. Ao ultrapassar esse valor, as requisições permanecem armazenadas em fila até o valor definido no parâmetro "ListenBacklog". Havendo a liberação de um processo filho, a primeira requisição que estiver na fila é executada pelo processo.

Para que não haja um *overload* o "Apache" possui um parâmetro denominado "ServerLimit". Tal parâmetro limita a quantidade total de processos filhos. Se o "MaxClient" for configurado no valor de 300 processos filhos e o "ServerLimit" for 256 processos filhos, valor padrão, o processo pai do "Apache" irá criar apenas 256. Assim, o "ServerLimit" é a segurança do sistema para que, independentemente de fatores, o valor do número de processos não será ultrapassado.

A determinação do tamanho do processo criado varia enormemente de um servidor para outro, com base nos módulos que estão rodando e no que está sendo pedido ao servidor. A utilização dos comandos "*top"* e *"ps"* mostra a lista de processos em execução, podendo, assim, extrair o valor do tamanho do processo criado pelo servidor "Apache".

Por exemplo, se foi determinado que cada processo utilizará 4MB de memória e, tendo uma carga máxima no servidor que tenha sido configurado para ter 125 processos em paralelo,

o mínimo de memória necessário para realizar as execuções dos processos será de 500MB, não esquecendo de memória para o sistema operacional e qualquer outra aplicação ou serviço que esteja rodando neste momento.

Outro parâmetro é o "KeepAliveTimeOut", que é responsável pelo tempo que o cliente pode passar sem realizar requisições continuando a ocupar um processo filho, é o limite do "KeepAlive" (parâmetro booleano que ativa a conexão permanente, ou seja, uma vez definido qual processo filho irá responder à requisição do cliente, ele irá permanecer com o cliente até o cliente liberar ou ultrapassar o limite de tempo de ocupação). A escolha equivocada do valor desse parâmetro pode ocasionar um atraso na entrega da resposta até a parada do sistema.

Se houve um valor muito grande do "KeepAliveTimeOut", a CPU será subutilizada, uma vez que neste tempo o processo pode ser de outro cliente, causando problemas de desempenho de sobrecarga (muitos processos ocupando recursos computacionais sem estarem em processamento) [19].

Já se o valor do "KeepAliveTimeOut" for muito pequeno, diminuir-se-á o tempo em que o processo estará no estado de espera e aumentado o tempo no estado de ocupado, elevando-se a utilização da CPU e demais recursos computacionais devido à demanda para o sistema operacional [8]**.** É uma conjectura semelhante a não ativação do "KeepAlive" do servidor "Apache". Um cliente irá necessitar de muitas conexões em curto período de tempo, atrasando o tempo de resposta do serviço.

O ajuste do parâmetro "KeepAliveTimeOut" é muito difícil, já que não se pode prever o tempo que cada usuário utiliza um documento, variando também de página para página no sítio eletrônico analisado. Geralmente toma-se a decisão de fixar o valor do "KeepAliveTimeOut" em 5, já que a tendência é funcionar bem para a maioria dos sítios eletrônicos [2]**.**

Por tudo que foi descrito, deve-se concordar com [2] que "estabelecer a quantidade de memória que você precisa é uma ciência no mínimo inexata. Para conseguir adivinhar esse número corretamente, você precisa observar o seu servidor atuando sob uma determinada carga, e ver quanta memória ele está usando".

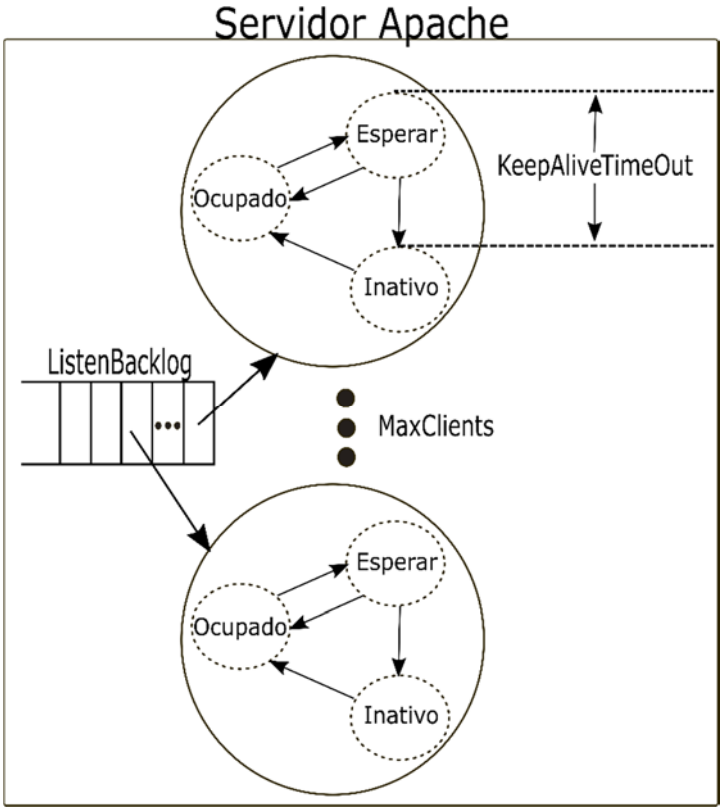
2.1 – Sistema Computacional de Resposta "HTTP"

O gerenciamento do hardware de um sistema computacional é realizado pelo sistema operativo através de abstrações dos códigos em Linguagem de Máquina, sendo denominados de recursos lógicos.

Técnicas como o multiprocessamento, “prefork”, é um exemplo de método disponível nos sistemas operacionais para prover a disponibilidade dos recursos computacionais aos usuários, de forma a maximizar a utilização dos recursos com o objetivo de diminuir o tempo de resposta.

O multiprocessamento ocorre quando um processador é multiplexado, ou seja, o processador é disponibilizado para os diversos processos existentes na máquina hospedeira, sejam processos demandados do sistema operacional ou do usuário. A Figura 2.2 mostra uma arquitetura dos processos para o servidor “Apache”.

Figura 2.2 - Arquitetura do Servidor “Apache” (Fonte: Adaptado de [8])



Existem três estados do processo para o servidor “HTTP”: ocupado, espera e inativo. O estado de ocupado é quando o processo está sendo utilizado por um cliente. O estado de espera, valor armazenado na variável “KeepAliveTimeOut”, é quando o processo está aguardando uma requisição do cliente. O estado de inativo é quando o processo deixa de ter dono e fica aguardando uma requisição de um cliente novo. A variável “MaxClient” é o valor armazenado de quantos processos podem existir concorrentemente.

Na visão do sistema operativo, a arquitetura para os processos servidores é mais complexa, já que o “prefork” utiliza a chamada “fork”, que cria os processos clones (filhos) a

partir de um processo pai. Cada processo servidor novo possui atributos, como o identificador de processo ("Process Identifier-PID"), que serve para identificar, contextualizar (prioridade, segurança e ambiente) e executar a tarefa planejada.

O relacionamento parental entre os processos é traduzido pelas informações que são mantidas pelo núcleo, de forma a poder gerenciar em conjunto o processo pai e os processos filhos. Para alguns sistemas operativos, a morte do processo pai não significa a morte do processo filho, e.g., o sistema operativo Unix sendo executado até o término do código ou envio de sinal para sua morte.

A ordem como os processos são executados no processador não é determinística devido ao mecanismo de base de troca de processo, escalonador, depender de inúmeros fatores, tais como o "time-slice" ou "quantum" e a preempção. Não há como prever qual será a ordem de execução dos processos; assim, um mesmo programa será executado de muitas formas diferentes.

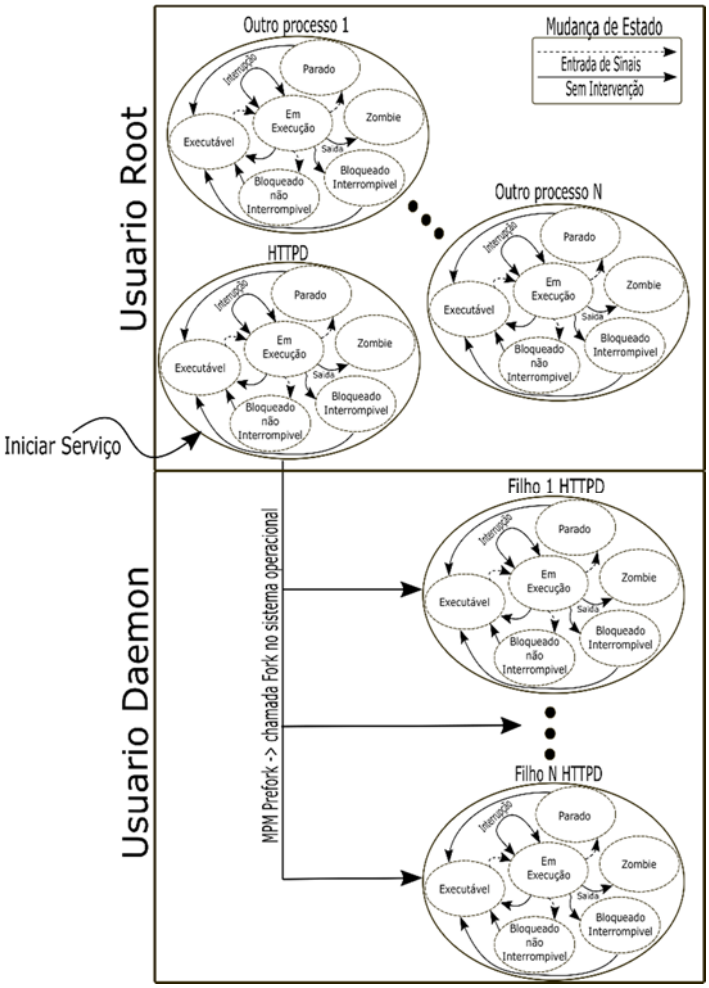
A segurança dos processos possui como base seu isolamento dos demais processos, confinando-os a endereços de memória limitados, ações limitadas pelos privilégios de seu dono (*owner*) e interação controlada através de chamadas ao sistema operativo. Assim, para que um processo possa interagir com outro, deve-se fazer uma chamada ao sistema operativo, o qual se encarregará de verificar as permissões e a execução da chamada.

Para iniciar o serviço de resposta às requisições "HTTP", o usuário "root" deve fazer uma chamada ao sistema operacional para que o programa "apachectl" com passagem de parâmetro adequada seja processado. O parâmetro "start" inicia o serviço, caso não esteja ativo; se o servidor já estiver ativo, não terá efeito no serviço e poderá produzir uma mensagem de aviso. O parâmetro "graceful" carrega os arquivos de configuração e reinicia suavemente, ou seja, as conexões continuam até sua finalização e então o servidor é reiniciado. O parâmetro "restart" carrega os arquivos de configuração, mas as conexões são encerradas imediatamente; caso o serviço não esteja ativo, tentará iniciá-lo.

Após o processamento com a passagem de parâmetro adequada, existirá um processo denominado "httpd" de domínio do usuário "root", que será responsável pela criação de processos filhos através da chamada "fork". Se houver requisições na fila para serem respondidas, o processo "httpd" pai irá verificar se existe algum processo filho disponível. Caso todos estejam ocupados, será solicitada ao sistema operacional uma chamada "fork" para que o processo pai seja clonado, criando mais um filho, repetindo-se a operação até o limite estabelecido pelo "MaxClients" e "ServerLimit".

A Figura 2.3 apresenta a arquitetura do servidor “Apache” na concepção do sistema operativo para o gerenciamento dos processos existentes. A única ligação entre os processos são os valores das variáveis do processo que são herdados do processo pai que tem como dono o usuário “root”, podendo interferir na execução do processo filho somente pelas chamadas do sistema operacional, os sinais. O usuário denominado de “Daemon” é responsável por prover os serviços na máquina hospedeira, ou seja, é ele que irá responder ao cliente que solicita uma requisição.

Figura 2.3- Servidor “Apache” no Sistema Operacional



A Figura 2.4 mostra os processos ativos em uma máquina hospedeira de um servidor “Apache” iniciado com o parâmetro “start”, tendo como configuração a variável “StartServer” igual 5. Essa variável no sistema indica o número mínimo de processos filhos que deverão existir, mesmo inexistindo carga no servidor (requisições a serem respondidas). Assim, o total de processos existentes vinculados ao servidor “Apache” é de 6: um processo é pai e 5 sendo processos filhos, clones do processo pai, e todos escutando a porta de comunicação 6405 do protocolo de comunicação TCP/IP da máquina hospedeira.

Observa-se a independência dos processos, quando para cada um dos processos, sendo clone ou não, existe um número “PID” que identifica o processo no sistema operacional. O processo pai com “PID” 4117 e seu dono é o usuário “ROOT”. Nos processos clones, o dono é o usuário “Daemon” e tem os seguintes PIDs: 4118, 4119,4120,4121,4122.

Figura 2.4- Lista de processos do Servidor “Apache”

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# sudo lsof -i:6405
sudo: unable to resolve host ServerApache
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd   4117   root    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4118 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4119 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4120 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4121 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4122 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

A Figura 2.5 confirma a hereditariedade dos processos com a execução do comando “pstree –p 4117”. Esse comando mostra a árvore hereditária de um determinado processo, lembrando que o PID 4117 é do processo pai.

Figura 2.5- Hereditariedade dos Processos do Servidor “Apache”

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# pstree -p 4117
httpd(4117)─┬─httpd(4118)
            ├─httpd(4119)
            ├─httpd(4120)
            ├─httpd(4121)
            └─httpd(4122)
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

O processo “httpd” do “root” é responsável pelo gerenciamento dos processos do “Daemon”. Assim, a criação e a morte dos processos são regidas por parâmetros que são armazenados no momento da inicialização do servidor, criação do processo “httpd” do “root”, e são utilizados para determinar se um processo deve ser morto ou gerado.

Havendo a morte do processo “httpd” do “root”, o serviço de resposta às requisições “http” continuará ativo. Entretanto, o gerenciamento dos processos será interrompido, não realizando tarefa de geração ou morte dos processos “httpd” do “Daemon”. A Figura 2.6 mostra o serviço de resposta “http” escutando a porta 6405 sem o processo “httpd” do “root”.

Figura 2.6- Serviço funcionando sem a presença do processo pai

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# sudo lsof -i:6405
sudo: unable to resolve host ServerApache
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd   4118 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4119 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4120 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4121 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
httpd   4122 daemon    4u  IPv6  27989      0t0  TCP *:6405 (LISTEN)
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

Os processos de usuário “Daemon” ficarão eternamente ativos esperando requisições de clientes, sem aumento nem diminuição do número de processos, ou seja, os recursos computacionais terão um consumo linear com coeficiente de regressão igual a 0.

2.2 – PARAMETRIZANDO O SERVIDOR “APACHE”

Apesar do “Apache” ser um sistema de automação de informação, ele não é automático. Os parâmetros, diretivas, são colocados em um arquivo de texto puro. Já foram vistas algumas das principais diretivas: “MaxClient”, “ServeLimit”, “KeepAliveTimeOut”, “KeepAlive”, e “ListenBackLog”.

Todas as diretivas são associadas a um valor que corresponde a alocar uma determinada quantidade de recurso computacional ou à ativação de serviços que terminará consumindo recursos computacionais também.

No “Apache 1.3”, a configuração era centralizada no arquivo "/etc/apache/httpd.conf", enquanto que no “Apache 2” ela é dividida em vários arquivos, seguindo a proposta de modularizar o serviço. À primeira vista, a organização do “Apache 2” parece muito mais complicada; mas, depois de entender, a coisa se revela muito mais simples e lógica.

Todos os arquivos de configuração estão organizados no diretório "/etc/apache2". Nele, temos as pastas "sites-available" e "sites-enabled", que contêm a configuração dos sites hospedados; as pastas "mods-available" e "mods-enabled", que armazenam a configuração dos módulos, “prefork”; o arquivo "ports.conf", onde vai a configuração das portas TCP que o servidor vai escutar; o arquivo "apache2.conf", que armazena configurações diversas relacionadas ao funcionamento do servidor, e a pasta "conf.d", que armazena arquivos com configurações adicionais.

Para o “Apache 1.3”, por padrão, os arquivos de configuração do “Apache” residem no diretório /etc/apache, e são os seguintes:

- ✓ “httpd.conf”: possui a configuração principal do “Apache”; possui diretivas que controlam a operação do daemon servidor. Um arquivo de configuração alternativo pode ser especificado através da opção "-f" da linha de comando.
- ✓ “srm.conf”: contém diretivas que controlam a especificação de documentos que o servidor oferece aos clientes. O nome desse arquivo pode ser substituído através da diretiva “ResourceConfig” no arquivo principal de configuração.

- ✓ “access.conf”: contém diretivas que controlam o acesso aos documentos. O nome desse arquivo pode ser substituído através da diretiva “AccessConfig” no arquivo principal de configuração.

O servidor “web” lê os arquivos descritos na ordem em que são mostrados. As configurações também podem ser especificadas diretamente no arquivo “httpd.conf”. Note que não é obrigatório usar os arquivos “srm.conf” e “access.conf”, mas isso proporciona uma melhor organização das diretivas do servidor, principalmente quando se tem um grande conjunto de diretivas.

Para o “Apache 2.4.3”, versão a ser utilizada no presente trabalho, a organização dos arquivos de configuração está por módulo de funcionalidade. Existe um arquivo de configuração do módulo de multiprocessamento “prefork”. O arquivo localizado em “/etc/apache2/mods-enabled/” tem a denominação de “mpm_prefork.conf”. A Figura 2.7 mostra o conteúdo desse arquivo.

Figura 2.7- Conteúdo do arquivos mpm_prefork.conf

```
cat /etc/apache2/mods-enabled/mpm_prefork.conf
#
<IfModule mpm_prefork_module>
        StartServers                5
        MinSpareServers             5
        MaxSpareServers             10
        MaxRequestWorkers           150
        MaxConnectionsPerChild        0
</IfModule>
```

A diretiva “StartServers” ajusta o número de processos filhos que devem ser criados na inicialização do “Apache”. Uma vez que o número de processos filhos é controlado dinamicamente, em função da carga de trabalho do servidor web, geralmente há poucas razões para alterar este número.

As diretivas “MinSpareServers” e “MaxSpareServers” criam um intervalo de número de processos ociosos existentes, ou seja, os processos que não estão respondendo a uma requisição não serão mortos pelo processo pai se a quantidade de processos estiver dentro do intervalo, caso contrário a quantidade excedente será morta. Esse ajuste só e realizado em sites muito ocupados. Ajustar tais parâmetros para valores extremos irá provocar um excesso de carga de trabalho ou *overload*, pois precisará terminar ou iniciar, sem necessidade, os processos de reserva em execução no “backgroud”.

A diretiva “MaxRequestWorkers” delimita o número máximo de processos filhos que é permitido ao servidor criar de acordo com a carga imposta. Em versões anteriores ao “Apache 2.4”, era chamado de “MaxClients”.

A diretiva “MaxConnectionsPerChild” delimita o número de conexões com que cada processo filho pode trabalhar individualmente. Após atingir o valor definido em “MaxConnectionsPerChild”, o processo filho irá terminar (morrer). Você pode ajustar o valor para 0 (zero) se quiser que o processo nunca expire. Ao usar valores diferentes de 0, contudo, você vai limitar a quantidade de memória que cada processo poderá consumir.

Após a alteração dos arquivos de configuração dos módulos desejados, o servidor deve ser parado e iniciado. O parâmetro “reload” na chamada do “apachectl” não carrega todas as configurações, principalmente se foi alterado o módulo de multiprocessamento “prefork”.

Assim, para que haja alteração nas configurações dos módulos do “Apache”, em qualquer versão, o responsável pela disponibilização do serviço deve projetar o valor do parâmetro desejado, saber qual é o arquivo em que está o parâmetro, alterar o arquivo (significa saber em qual linha está a informação), salvar o arquivo modificado, parar o serviço e inicializar o serviço.

E é dessa forma que as “warehouse” gerenciam os serviços de resposta às requisições “HTTP”. Dentre as pesquisas realizadas, existem ferramentas *cases* que proporcionam aos responsáveis pelo serviço um acesso rápido ao arquivo a ser alterado com parada do serviço. Entretanto, não foi observada uma ferramenta que realize o projeto do valor adequado às especificações ambientais do sistema com aplicação no servidor sem a parada completa do sistema de resposta às requisições “HTTP”.

3 - MODELAGEM DO SISTEMA COMPUTACIONAL "APACHE"

Em se tratando do sistema computacional "Apache", que pode ser dividido em *hardware* e *software*, aspectos dinâmicos do *hardware* relacionados a capacitores e transistores podem ser desconsiderados, tornando o sistema discreto, por exemplo. Na parte do *software*, as variáveis são formadas por diversos bits, mas serão tratadas como uma única informação, ou seja, o valor será real. Outras dinâmicas serão desconsideradas, fazendo-se uma abordagem na modelagem estática, utilizando-se a teoria de filas, ou seja, será tratada apenas a abordagem orientada aos dados do sistema computacional.

A teoria de filas é uma ferramenta matemática que trata de eventos aleatórios e o estudo da espera em filas. Proporciona uma maneira de definir o ambiente de um sistema de filas matematicamente permitindo prever respostas prováveis e tempos de espera. Os sistemas de filas diferem entre si de acordo com as hipóteses que se faz a respeito dos padrões de chegada e das taxas de serviço. Na análise, precisa-se adotar hipóteses sobre o comportamento do sistema. Caso contrário, não se tem por onde começar.

Inicialmente considere λ como sendo o número de requisições que são enviadas ao sistema computacional "Apache" pelos clientes, distribuídas estaticamente em um intervalo de tempo. As requisições serão processadas dependendo da lógica empregada no *software* de resposta à requisição. Então o tempo $\bar{n}$ que uma requisição leva para ser respondida é dado por $\bar{n} = \frac{1}{\mu}$, onde μ é a taxa de serviço do servidor (quantidade de requisições que são respondidas em um período de tempo). Se $\mu<1$, o sistema está recebendo mais requisições do que é capaz de processar, ocasionando *overload* e para a teoria de fila o sistema se torna instável, considerando-se o tamanho da fila finito. O inverso é considerável estável.

A lei de Little relaciona o número de clientes no sistema com o tempo médio despendido no sistema. É considerada como uma lei de conservação da teoria das filas (em condições estacionárias), sendo válida para qualquer propriedade estática e política adotada no *software* de reposta às requisições.[1]

$$d = \frac{\bar{n}}{\lambda} \qquad \text{(Lei de Little)}$$

Na teoria de filas existe uma notação especial para expressar propriedades importantes da fila de uma maneira compacta, notação de Kendall, tendo a seguinte sistemática: A/S/m/B/K/DS.

Onde:

A = Distribuição de tempo interchegada (M,Ek,Hk,D,G)

S = Distribuição de tempo de serviço (M,Ek,Hk,D,G)

m = Número de canais de serviço simultâneos, neste caso servidores / processadores

B = Quantidade de Buffer ou capacidade do sistema

K = Tamanho da população

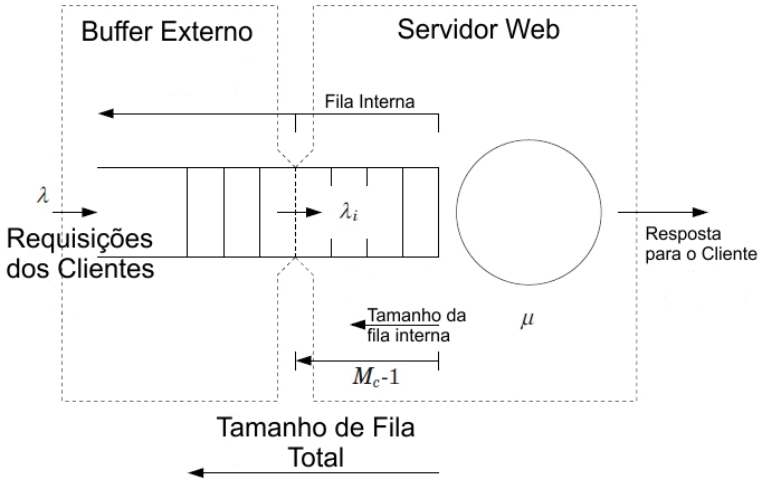
DS = Disciplina do serviço (FCFS/FIFO, LIFS/LIFO, SIRO, RD, GD)

M é a distribuição exponencial, chamada de "memoryless" (Tráfego Markoviano). Ek é "erlang", com parâmetro K. Hk é hiperexponencial, com parâmetro K. D é uma distribuição determinística significando que o tempo de chegada e o tempo de serviço são constantes ou não possuem variância.

FIFO é procedimento de tratamento de fila, pela qual a primeira requisição que chega será a primeira a ser processada; LIFO é a última requisição que chega e será a primeira a ser processada. SIRO é selecionada randomicamente a requisição que será processada. RD: a seleção da requisição a ser processada é baseada em prioridades. GD: é realizada a seleção por distribuição genérica.

As características da fila do sistema computacional "Apache" são M/M/1 (cadeia de Markov) e um esquema é mostrado na Figura 3.1 [1]. Quando são suprimidas as demais informações na denotação Kendall, os valores adotados são os padrões adotados (infinito, infinito, FCFS/FIFO).

Figura 3.1 – Esquema da fila do sistema computacional "Apache" (Fonte: [8])



Utilizando-se o modelo Tipper [1][20], que equaciona o fluxo dinâmico de uma fila tendo um único provedor de serviço, tem-se;

$$\frac{d}{dt}n = \lambda(t) - \mu H(n(t)) \qquad \textbf{(Modelo Tipper)}$$

Onde n é a dinâmica do estado, que representa a expectativa de número de trabalhos no sistema computacional, e H(.) é a média dos conjuntos de funções de estado utilizados. No caso estudado (características da fila), H é dado por:

$$H\big(n(t)\big) = \frac{n(t)}{1 + n(t)} \qquad \textbf{3.1}$$

A dinâmica desse modelo é representativa para a teoria de filas e no estado estacionário. Então:

$$\frac{d}{dt}n(t) = \lambda(t) - \mu\frac{n(t)}{1 + n(t)} \qquad \textbf{3.2}$$

O tempo de resposta de uma requisição começa a partir do momento de chegada ao sistema computacional "Apache". Ele pode ser modelado utilizando-se a lei de Little e será a média do tempo das requisições que foram concluídas em um determinado número de amostras,

$$\frac{d}{dt}d(t) = -\frac{1}{T_d}d(t) + \frac{1}{T_d}\frac{n(t)}{\lambda(t)} \qquad \textbf{3.3}$$

Onde T_d é o tempo de amostragem.

Para o número de requisições que estão dentro do sistema "web", será adotada a nomenclatura n_i, a qual é mensurável. A dinâmica da variável n_i está associada à fila e não à medição, já que a fila é composta por outras filas. Desta forma, podemos utilizar a relação estática para a dinâmica do processo[12]. Assim,

$$n_i(t) = n(t)\left[1 - \left(\frac{n(t)}{1 + n(t)}\right)^{M_c}\right] =: f_i(n(t)) \qquad \textbf{3.4}$$

Considere a taxa de chegada de requisições no servidor "web" λ_i como sendo o número de requisições que chegam ao servidor "Apache" em um intervalo de tempo. Agora, considere n_e como o número de processos que se encontram no buffer do servidor fora do "Apache". O número de processos externos pode ser escrito por um integrador simples, similar às leis de conservação da física. Então,

$$\boldsymbol{n_e(t) = \int_0^t \big(\lambda(\tau) - \lambda_i(\tau)\big) d\tau} \qquad \textbf{3.5}$$

Como as relações estáticas servem para a dinâmica em se tratando do modelo a ser estudado e como o número total de requisições no sistema é igual à soma das requisições que estão na lista externa com as requisições que estão na lista interna, pode-se fazer a diferença obtendo:

$$\boldsymbol{\lambda_i(t) = \lambda(t) - \frac{d}{dt}\{n(t) - n_i(t)\}} \qquad \textbf{3.6}$$

O tempo de resposta no servidor "Apache" $d_i(t)$ é o tempo desde a chegada da requisição na fila do servidor até o completo processamento da requisição. Fazendo-se a derivação do modelo da fila completa e usando-se a lei de Little e um filtro passa baixa, tem-se:

$$\boldsymbol{d_i(t) = -\frac{1}{T_d} d_i(t) + \frac{1}{T_d}\frac{n_i(t)}{\lambda_i(t)}} \qquad \textbf{3.7}$$

Para relacionar com a quantidade de memória sendo utilizado pelo servidor "Apache" utiliza-se a relação:

$$\boldsymbol{Mem = kn_i} \qquad \textbf{3.8}$$

Sendo k a quantidade de memória que cada processo ocupa na memória. Isso irá depender da página que será chamada pelo usuário. Nos testes reais, será utilizada uma única página e sendo obtido por teste o valor de 0,3% de uma quantidade de 2 Gbytes.

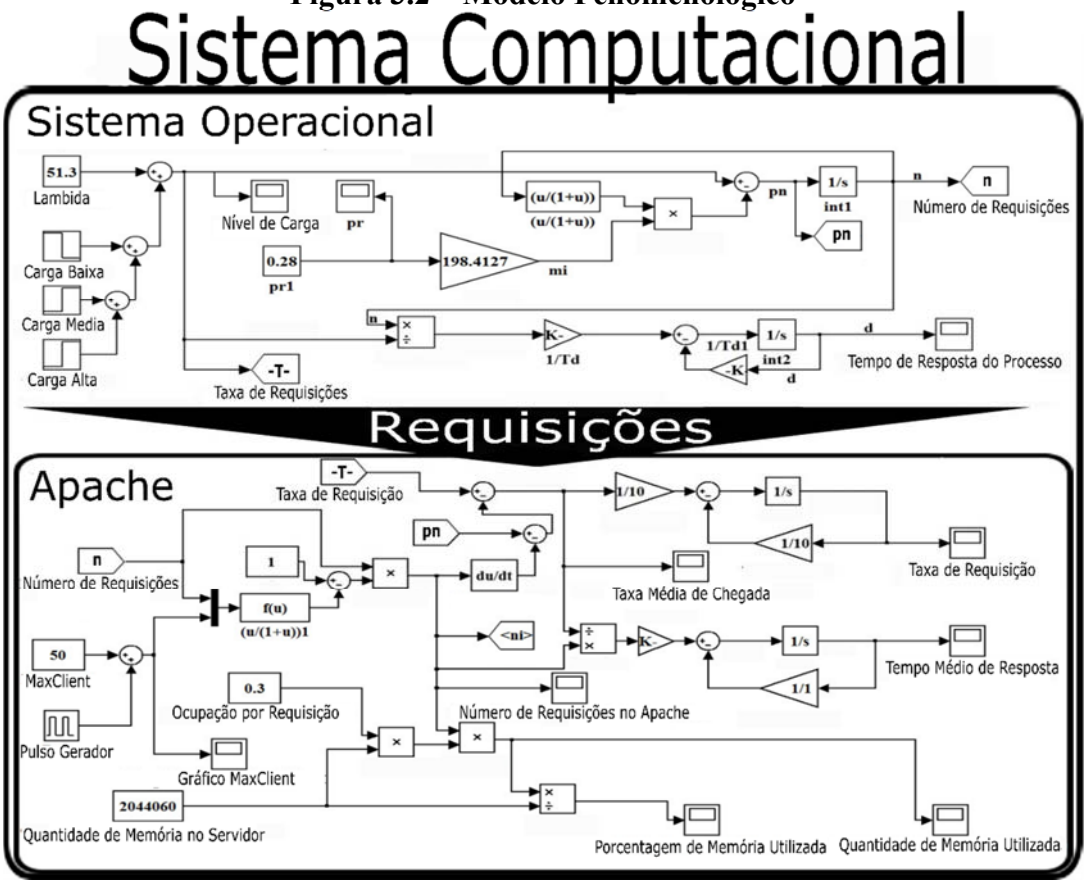
A construção do modelo no "SIMULINK" no "MATLAB" é visto na Figura 3.2. Nesse modelo, é possível observar o número de processos que estão esperando na fila para o processamento, o tempo de resposta de um processo (a partir da entrada da fila do servidor "Apache"), assim como realizar alterações em parâmetros de forma a provocar perturbações no sistema, como a taxa de processos por tempo (carga exercida no sistema), quantidade total disponível de memória no servidor hospedeiro etc. Variáveis que podem ser projetadas também estão incluídas no modelo construído, como o "MaxClient", que é foco de estudo deste trabalho.

Esse modelo proporciona a obtenção do entendimento do relacionamento entre as variáveis que podem ser projetadas (controladas por um processo com metodologia a ser desenvolvida através da utilização da informação automatizada) com as variáveis de resposta do sistema, incluindo-se no sistema perturbações controladas para um estudo da dinâmica do

processo. Para este trabalho, nosso foco é o relacionamento entre o número de processos concorrentes (“MaxClient”) com a quantidade de memória principal que está sendo consumida para o armazenamento desses processos concorrentes.

Ao desenvolver a Figura 3.2, optou-se por separar a fila externa de requisições “HTTP” pertencente ao sistema operacional, que é infinita, da fila interna de requisições “HTTP” do servidor “Apache”, que é finita. Desta forma, é possível visualizar que as requisições se acumulam na fila externa e de algum modo isso ocupara recursos computacionais (memória). Neste momento, não está no foco do problema realizar tais pesquisas, mas acredita-se que influenciará diretamente no buffer ou cache de memória do sistema operacional. Entretanto, para os estudos fenomenológicos, o tratamento a ser dado a estas duas filas será o de uma única fila de caráter infinito.

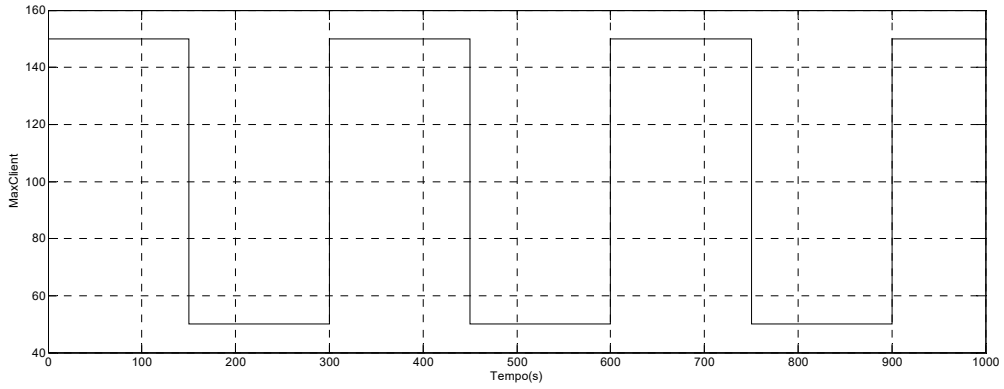
Figura 3.2 – Modelo Fenomenológico



Para os testes do modelo fenomenológico, foram utilizados, para λ, três níveis de carga (baixa=50, média = 100 e alta=150) e foi utilizado 28% do processador para as operações do servidor “Apache”. O tempo de amostragem $T_d = 1$ segundo para os filtros passa baixa. No “MaxClient”, foi parametrizado com 50 posições na fila interna do “Apache” para as

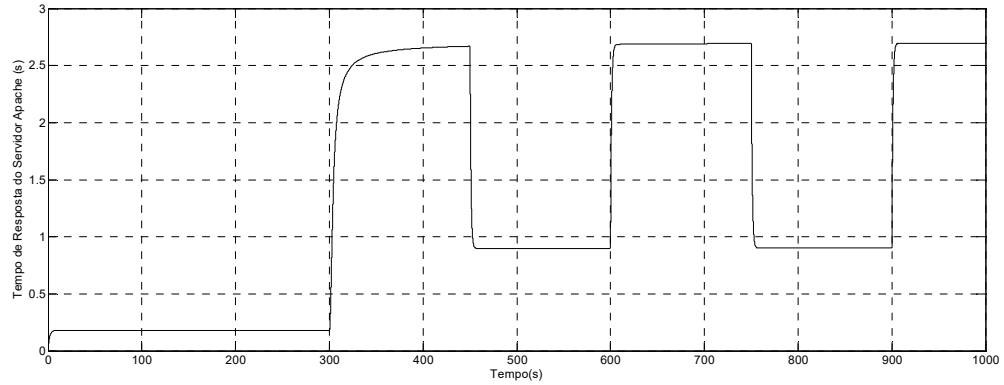
requisições e alterado com pulsos de amplitude 100 em período de 300 segundos com permanência de 50% (Figura 3.3).

Figura 3.3 – Variação do "MaxClient" no modelo fenomenológico



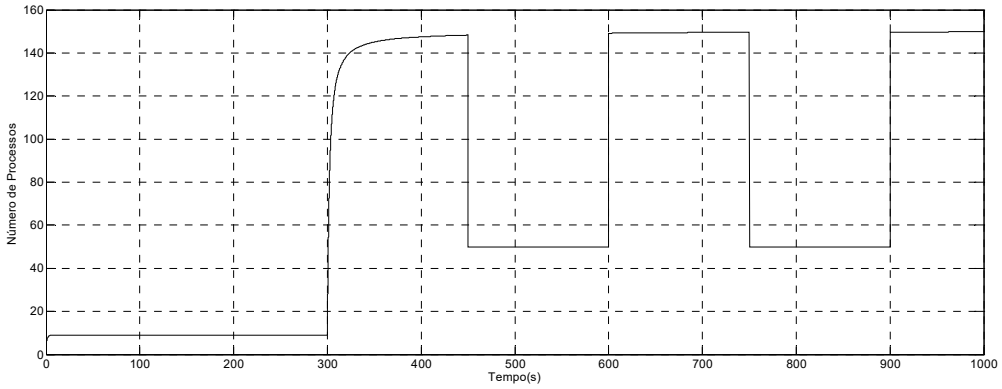
O tempo de resposta do servidor "Apache" irá depender do número de requisições que estão esperando para serem processadas. É importante lembrar que esse tempo inicia a contagem quando o processo já está na lista do servidor "Apache" e não na lista externa, ou seja, no sistema operacional. Até os 300 segundos, a variação de "MaxCliente" não interfere no tempo de resposta devido existir poucos processos nas filas, ou seja, a taxa de requisições é menor do que a taxa de processamento, permanecendo o tempo de reposta constante (Figura 3.4). Praticamente não existe tempo de latência para que os clientes tenham as respostas das requisições enviadas ao servidor.

Figura 3.4 – Tempo de reposta do servidor web



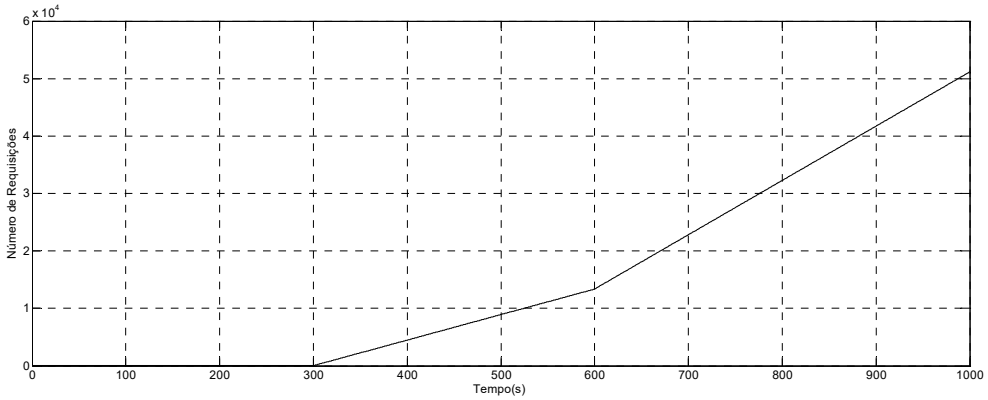
No tricentésimo segundo, há um aumento de carga (a taxa de processamento ultrapassa a taxa de requisição) no sistema computacional, fazendo aumentar o tempo de resposta do servidor "Apache" e tendo variações com o mesmo comportamento do "MaxClient", já que a fila interna do servidor será ocupada por processos que estão esperando sua vez de serem executados (Figura 3.5).

Figura 3.5 – Número de Processos na Fila do Servidor "Apache"



O aumento da carga provoca um aumento no número de processos na fila interna (servido "web") e na fila externa (sistema operacional). Como inicialmente foi definido que a fila externa seria infinita, a simulação não irá fazer limitações. Entretanto, o ângulo de inclinação da reta define a taxa de requisições (Figura 3.6).

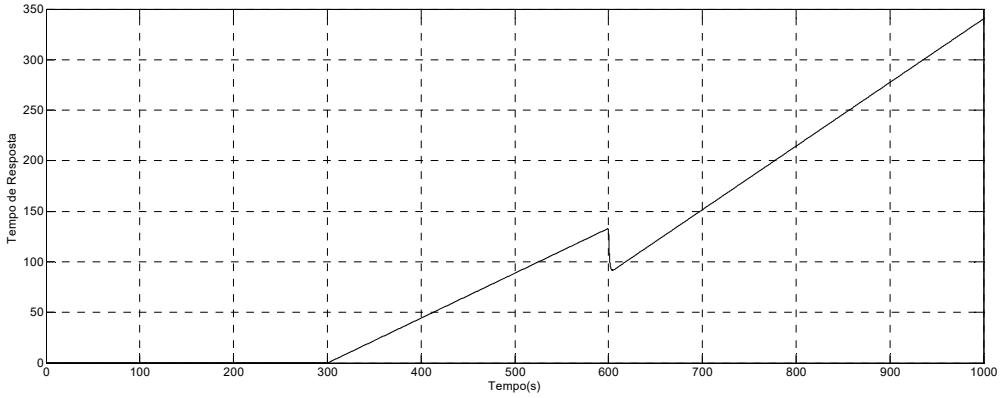
Figura 3.6 – Número de Requisições na Fila Externa



O número de requisições tem um relacionamento direto com o tempo de resposta. Ao observar as Figuras 3.6 e 3.7, nota-se a semelhança entre os gráficos. O coeficiente angular das retas está relacionado com a taxa de processamento ser menor do que a taxa de chegada de requisições "HTTP". Até o tempo de 300s, a carga empregada na simulação era menor do que o poder de processamento não tendo requisições acumuladas. A partir dos 300s, a carga é maior do que o poder de processamento começando a acumular processos nas filas externa e interna. De 0s a 600s, a fila interna tinha 50 posições. Desta forma, por não haver alteração na fila do "Apache", o tempo de resposta acompanhou de forma similar o número de requisições. No tempo de 600 segundos, quando houve a variação para uma taxa de 150 processos por segundo, a taxa de resposta tem um decremento, mas rapidamente volta acrescer. Isso é explicado por

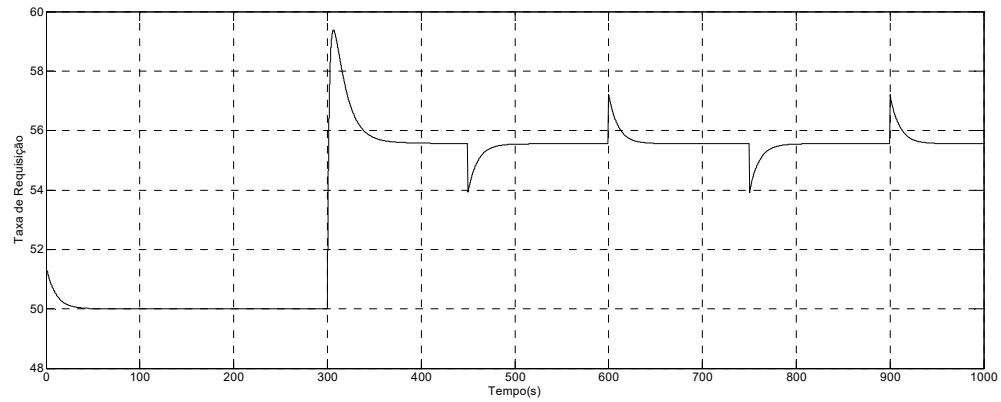
um aumento no limite do número de processos que estão na fila interna do servidor "web" (Figura 3.7).

Figura 3.7 – Tempo de Resposta do Sistema Computação



A taxa de requisição no servidor "web" irá ter pulsos de variações no momento das mudanças de valores "MaxClient", voltando à mesma taxa interna de requisições (intervalo de 300 a 1000 segundos). Isso é decorrente do aumento do limite da fila interna e de a taxa de requisições (λ) ser maior do que a taxa de processamento (μ) (Figura 3.8). O comportamento já era esperado, uma vez que nos gráficos anteriores a variação do número de processos depende da carga sendo empregada e do poder de processamento que está constante nestas amostras.

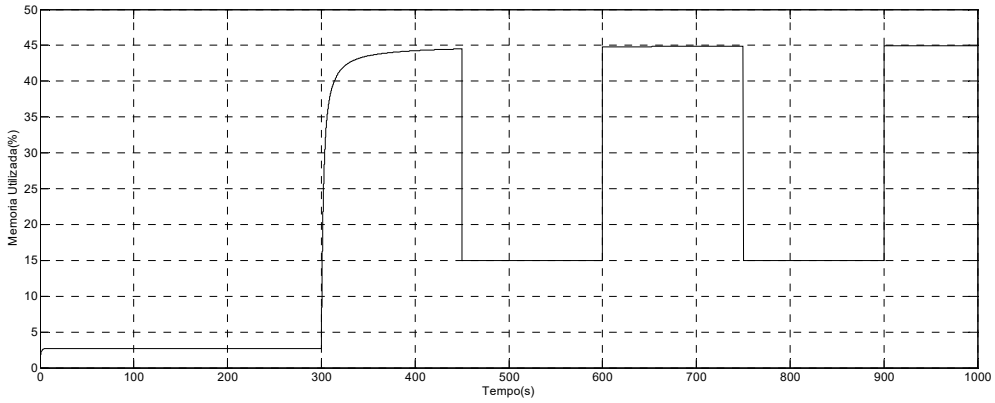
Figura 3.8 – Taxa de Requisição do Servidor "Apache"



Finalmente, a quantidade de memória que o servidor "Apache" estará utilizando. Primeiramente, a quantidade de memória utilizada pelo servidor será considera a que está dentro da fila interna. Assim, a quantidade de memória utilizada na fila externa não está sendo somada aos resultados da simulação (Figura 3.9) . No intervalo de 0 a 300 segundos, temos uma utilização constante da memória, mesmo havendo variação de "MaxClient". O ocorrido é explicado por não haver números de processo suficientes para serem inseridos na fila interna,

definido pela relação entre λ e μ. Entre o intervalo 300 a 1000 segundos, a taxa de requisição cresce e o número de processos também. Na primeira variação, intervalo de 300 a 450 segundos, há um tempo de acomodação do sistema devido à inclinação na reta de crescimento do número de processos. A partir da terceira variação, 600 segundos, o sistema não terá mais esse comportamento por já ter o número suficiente de processos na fila externa para ocuparem a fila interna, fazendo a variação da percentagem de memória ocupada se acomodar aproximadamente conforme o tempo estimado para o sistema T_d.

Figura 3.9 – Variação do Percentual de Memória Utilizada pelo Servidor "Apache"



3.1 - Linearização do Sistema Fenomenológico

O objetivo é construir uma função de transferência que relacione a quantidade de memória com o MaxClient. Então:

$$\boldsymbol{G(s) = \frac{\Delta y}{\Delta u} = \frac{\Delta Mem}{\Delta M_c} = K\frac{\Delta n_i}{\Delta M_c}} \qquad \textbf{3.9}$$

Sendo K a quantidade de memória que cada processo ocupa na fila interna. O valor foi mencionado anteriormente.

O ponto de operação para linearização será: $\lambda^0, \mu^0, {M_c}^0$. Então:

$$\Delta\dot{n} = \Delta\lambda - \Delta\left(\frac{n}{1+n}\mu\right) = -\mu^0\Delta\left(\frac{n}{1+n}\right)$$

$$\Delta\dot{n} = -\mu^0\left(\frac{\Delta n(1+n^0) - n^0\Delta(1+n)}{(1+n^0)^2}\right.$$

$$\Delta\dot{n} = -\mu^0\left(\left(\frac{1}{1+n^0}\right)\Delta n - \left(\frac{n^0}{(1+n^0)^2}\right)\Delta n\right)$$

$$\Delta\dot{n} = -\mu^0\left(\frac{1}{(1+n^0)^2}\right)\Delta n$$

$$\boldsymbol{\Delta\dot{n} = \frac{-\mu^0}{(1+n^0)^2}\Delta n} \qquad \textbf{3.10}$$

Agora,

$$\Delta\dot{d} = -\frac{1}{T_d}d + \frac{1}{T_d}\left(\frac{n}{\lambda^0}\right)$$

$$\boldsymbol{\Delta\dot{d} = -\frac{1}{T_d}\Delta d + \frac{1}{T_d}\left(\frac{\Delta n}{\lambda^0}\right)} \qquad \textbf{3.11}$$

Para

$$\Delta n_i = \Delta n\left[1 - \left(\frac{n^0}{1+n^0}\right)^{M_c{}^0}\right] + n^0\Delta\left[1 - \left(\frac{n}{1+n}\right)^{M_c}\right]$$

$$\Delta n_i = \Delta n\left[1 - \left(\frac{n^0}{1+n^0}\right)^{M_c{}^0}\right] - n^0\Delta\left[\left(\frac{n}{1+n}\right)^{M_c}\right]$$

$$\Delta n_i = \Delta n\left[1 - \left(\frac{n^0}{1+n^0}\right)^{M_c{}^0}\right] - n^0\left[M_c{}^0\left(\frac{n^0}{1+n^0}\right)^{M_c{}^0-1}\Delta n + \left(\frac{n^0}{1+n^0}\right)^{M_c{}^0} ln\left(\frac{n^0}{1+n^0}\right)\Delta M_c\right]$$

$$\boldsymbol{\Delta n_i = \left[1 - \left(\frac{n^0}{1+n^0}\right)^{M_c{}^0} - n^0 M^0\left(\frac{n^0}{1+n^0}\right)^{M_c{}^0-1}\right]\Delta n} - \left[n^0\left(\frac{n^0}{1+n^0}\right)^{M_c{}^0} ln\left(\frac{n^0}{1+n^0}\right)\right]\Delta M_c \qquad \textbf{3.12}$$

Para a equação $\lambda_i = \lambda - (\dot{n} - \dot{n_l})$ temos:

$$\boldsymbol{\Delta\lambda_i = -\Delta(\dot{n} - \dot{n_l}) = -\Delta\dot{n} + \Delta\dot{n_l}} \qquad \textbf{3.13}$$

E para $\Delta\dot{\lambda}_f = -1\frac{1}{T_\lambda}\lambda_f + \frac{1}{T_\lambda}\lambda_i$ temos:

$$\Delta\dot{\lambda}_f = -1\frac{1}{T_\lambda}\Delta\lambda_f + \frac{1}{T_\lambda}\Delta\lambda_i \qquad 3.14$$

Fazendo

$$d_1 = -\frac{1}{T_d}\dot{d}_1 + \frac{1}{T_d}\left(\frac{n_1}{\lambda_1}\right)$$

$$\Delta\dot{d}_1 = -\frac{1}{T_d}\Delta d_i + \frac{1}{T_d}\left(\frac{\Delta n_1 \lambda_i^{\,0} - n_i^{\,0}\Delta\lambda_i}{(\lambda_1^{\,0})^2}\right) \qquad 3.15$$

Finalmente,

$$\frac{\Delta n_i}{\Delta M_c} = \left[1 - \left(\frac{n_0}{1+n_0}\right)^{M_c^{\,0}} - n_0 M_c^{\,0}\left(\frac{n_0}{1+n_0}\right)^{M_c^{\,0}-1}\right]\frac{\Delta n}{\Delta M_c} - \left[n^0\left(\frac{n_0}{1+n_0}\right)^{M_c^{\,0}} ln\left(\frac{n_0}{1+n_0}\right)\right] \qquad 3.16$$

3.2 - Modelo Não Linear Simplicado

Para simplificar a análise de projeto, visando ganhar uma intuição preliminar acerca do comportamento dinâmico da planta, supõem-se que a taxa de crescimento do número médio de requisições seja bastante lenta ($\dot{n} \approx 0$) e que o mecanismo de atuação via M_c(MaxClient) não é imediata e tem uma constante de tempo T, o sistema pode ser modelado pelo seguinte sistema não linear simplificado:

$$M_c = -\frac{1}{T}\dot{M}_c + \frac{1}{T}M_c^{\,*}$$

$n_i = n_0\left[1 - \left(\frac{n_0}{1+n_0}\right)^{M_c}\right]$, sendo $n_0 = \frac{\lambda^0}{\mu^0 - \lambda^0}$ onde $M_c^{\,*}$é o valor de referência de M_c.

O correspondente modelo linearizado em torno de um ponto de operação $(\lambda^0, \mu^0, M_c^{\,0})$ é dado por:

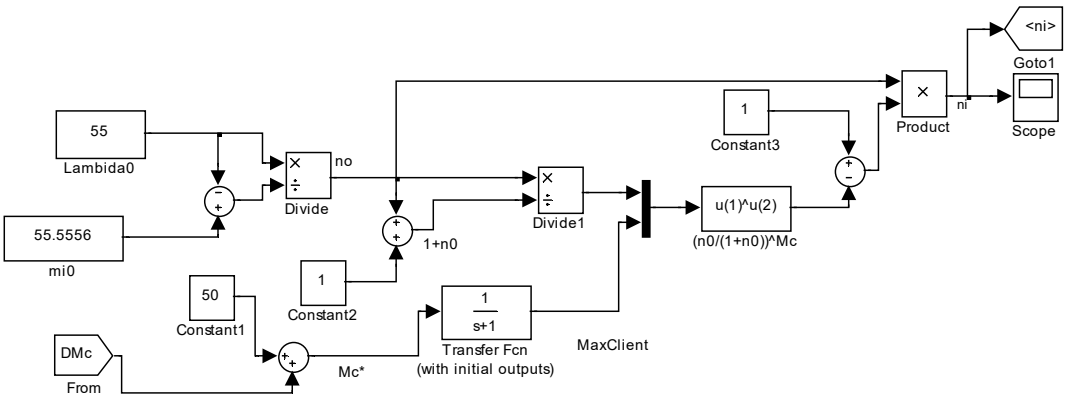
$$\Delta n_i = \frac{G_0}{Ts+1}\Delta M_c$$

Onde

$$G_0 = -n_0\left(\frac{n_0}{1+n_0}\right)^{M_c^{\,0}} ln\left(\frac{n_0}{1+n_0}\right) \qquad 3.17$$

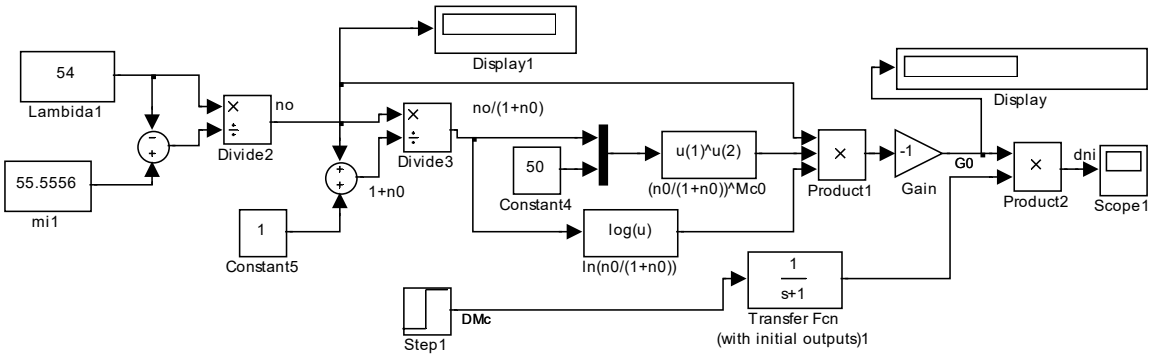
Os modelos foram implementados no “SIMULINK” (Figuras 3.10 e 3.11).

Figura 3.10 – Modelo Fenomenológico Simplificado Não Linear



Em [21], para os modelos não lineares, se pelo menos uma das derivadas parciais da função em relação aos parâmetros depender de pelo menos um dos parâmetros do modelo, pode-se dizer que o modelo é não linear, desde que os parâmetros sejam não lineares. Observa-se, na Figura 3.10, o descrito em [21] quando a quantidade de processos existentes no sistema computacional "Apache" interfere no valor n_{i+1}, ou seja, nele mesmo. Assim caracterizando um modelo não linear. O componente "Scope" do "SIMULINK" armazena o histórico temporal contínuo do número de processos. Com isso, é possível observar a dinâmica do modelo nas aproximações das instabilidades, que é a aproximação de $\boldsymbol{\lambda^0}$de $\boldsymbol{\mu^0}$.

Figura 3.11 – Modelo Fenomenológico Simplificado Linear



Realizando-se alguns testes no modelo fenomenológico simplificado linear para verificação do comportamento do sistema, quando o sistema se aproxima da instabilidade, temos os resultados conforme a Tabela 3.1 . Lembrando que a instabilidade é quando ocorrer $\lambda^0 > \mu^0$(o número de requisições que estão chegando é próximo ao poder de processamento da máquina hospedeira simulada).

Tabela 3.1 – Tabela de ensaios de aproximação da instabilidade do sistema

VARIÁVEIS	ENSAIO 1	ENSAIO 2	ENSAIO 3
$\boldsymbol{\mu^0}$	55,5556	55,5556	55,5556
$\boldsymbol{M_c}$	50	50	50
$\boldsymbol{\lambda^0}$	50	54	55
$\boldsymbol{n_0}$	9	34,71	99
$\boldsymbol{n_i{}^0}$	8,9536	26,323	39,104
$\boldsymbol{G_0}$	0,004887	0,238300	0,602000
ADMISSÃO	99,4%	75,84%	39,5%

Fica confirmado, a partir dos ensaios, que a aproximação λ^0 ao valor de μ^0 irá provocar uma instabilidade no sistema. Primeiro, é o esforço do ganho que se eleva consideravelmente. Segundo, a porcentagem de processos que estão fora da lista interna cresce muito. No ensaio 1, é de 99,4% de inclusão na lista interna; no ensaio 2, já diminui para 75,84% e no ensaio 3 somente 39,5% das requisições estão na lista interna do servidor. E essa queda abrupta ocorreu com uma variação de carga de apenas 5 unidades da variável - indícios de um sistema não linear no intervalo de estudo.

3.3 – PI Clássico

Para que o sistema em malha fechada tenha erro nulo de regime, ou seja, $n_i = n_i{}^{ref}$ (ganho unitário em malha fechada), deve-se projetar um controlador que possua uma ação integradora (controlador PI).

Entretanto, para o sistema em estudo, a não linearidade está presente. Assim, controladores que são projetados em um determinado ponto de operação podem não satisfazer os requisitos do projeto em outro ponto. Inicialmente, será projetado um controlador PI nominal e posteriormente um controlador intervalar.

O controlador nominal utiliza o ponto médio (ensaio 2) dos pontos de operação. Serão utilizados os valores dos ensaios da Tabela 3.1 para projetar este controlador, sendo utilizados valores aceitáveis inicialmente para os requisitos da maioria dos projetos, tais como: $\xi_d = 0{,}7 \; e \; t_{set} = 2T$.

Para projetar o controlador, utilizam-se as relações:

$$\boldsymbol{t_{set} = \frac{4}{\xi_d \omega_n}} \quad \textbf{3.18}$$

$$\boldsymbol{a_0 + b_0 c_1 = 2\xi_d \omega_n} \quad \textbf{3.19}$$

$$\boldsymbol{b_0 c_0 = {\omega_n}^2} \quad \textbf{3.20}$$

Assim,

$$\omega_n = \frac{4}{\xi_d t_{set}} = \frac{4}{\xi_d 2T} = \frac{4}{0{,}7 * 2 * 1} = 2{,}8571$$

Para o sistema estudado, tem-se:

$$b_0 = G_0$$

$${G_0}^{Nominal} = 0{,}2383$$

Então,

$$c_0 = \frac{{\omega_n}^2}{b_0} = \frac{{\omega_n}^2}{{G_0}^{Nominal}} = \frac{2{,}8571^2}{0{,}2383} = \mathbf{32{,}8997}$$

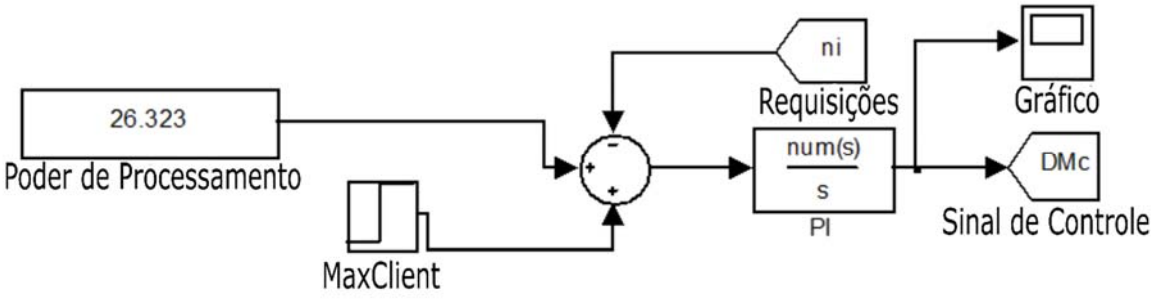
$$c_1 = \frac{2\xi_d \omega_n - a_0}{b_0} = \frac{(2 * 0{,}7 * 2{,}8571) - 1}{0{,}3034} = \mathbf{12{,}2535}$$

Obtêm-se o controlador PI nominal projeto, com a forma:

$$PI(s) = \frac{c_1 s + c_0}{s} = \frac{12{,}2535s + 32{,}8997}{s}$$

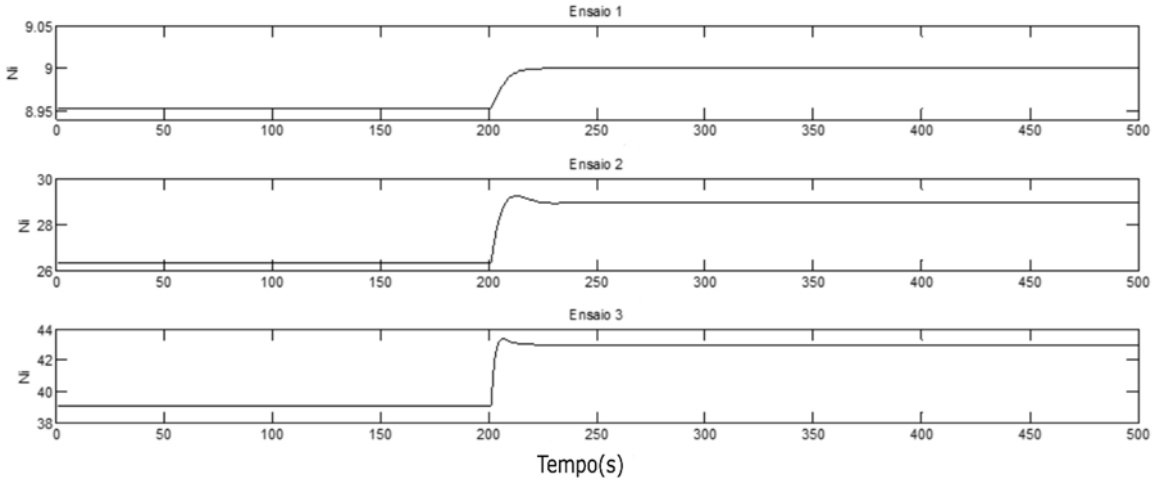
A construção no "SIMULINK" do controlador PI ocorre conforme a Figura 3.12.

Figura 3.12 – Controlador PI nominal do modelo Fenomenológico



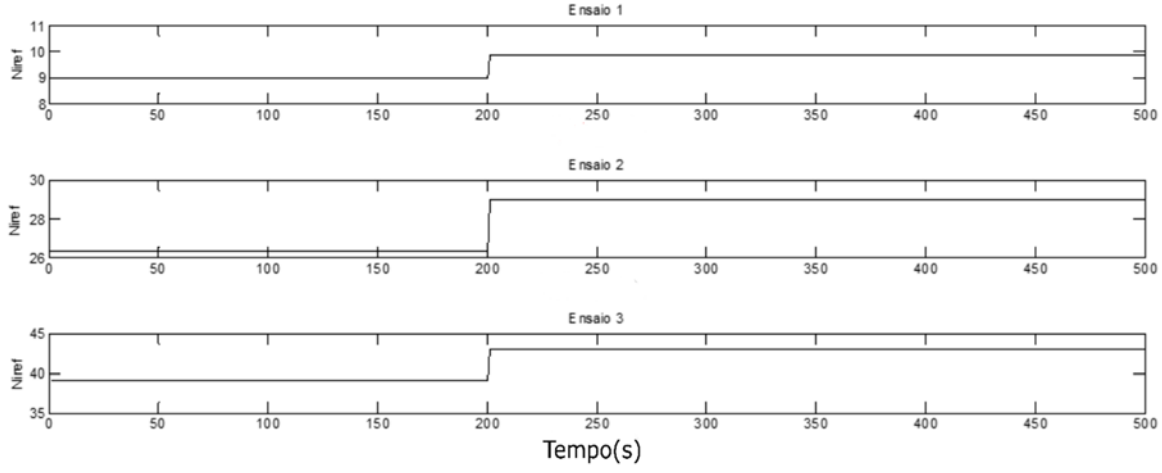
A Figura 3.13 mostra a atuação do controlador PI nominal nos três pontos de operações (ensaio 1, ensaio 2 e ensaio 3). É importante lembrar que o controlador foi projetado para o ponto de operação do médio (ensaio 2). Então, os requisitos do projeto desejados, sobressinal de 5% e o tempo de acomodação, foram respeitados. Entretanto, para os outros dois pontos de operação, o sistema não teve o comportamento esperado.

Figura 3.13 – Atuação do Controlado PI nominal nos três pontos de operação



O degrau (*step*) aplicado em cada ensaio de malha fechada pode ser visto na Figura 3.14. Não esquecer que Niref é a referência de número de processos no sistema de resposta às requisições “HTTP”, ou seja, o parâmetro “MaxClient”. Note que existe um pequeno atraso para o sistema entrar em regime. Isso ocorre porque os processos não são criados ou mortos de forma imediata no sistema de resposta às requisições “HTTP”, obedecendo à singularidade do sistema-alvo real. No modelo fenomenológico esse atraso é implementado pelas funções de transferência e não são variantes no tempo.

Figura 3.14 – Degrau na referência no sistema fenomenológico em malha fechada com controlador PI nominal



3.4 – Controlador Intervalar

Fica evidente nos testes realizados que o controlador projetado no ponto de operação médio não possui robustez [5]para os demais pontos. Técnicas de controle intervalar são indicadas para que o sistema seja robustamente estável na faixa de operação desde a carga baixa até a alta. A técnica a ser aplicada neste trabalho foi publicada em [26], na qual se utiliza o intervalo entre os pontos de operação máximo e mínimo em que o sistema irá trabalhar.

Adotando-se como intervalo de trabalho os ensaios 1 e 3 (mínimo e máximo), pode-se escrever:

$$G(s) = \frac{G_0}{Ts+1}, para\ G_0 = \left[{G_0}^-, {G_0}^+\right]$$

sendo ${G_0}^- = 0{,}004887$ o menor valor do intervalo de trabalho e ${G_0}^+ = 0{,}602000$ o maior valor do intervalo.

Utilizando-se as equações 18, 19, 20 e definindo-se as especificações desejadas do projeto $\xi = [0{,}6\ ;\ 0{,}8\ ]$ e $\omega_n = [2\ ;\ 6{,}6667\ ]$, utilizou-se $1 < t_{set} < 2{,}5$ e, continuando os cálculos teremos:

$$2{\xi_d}^-{\omega_n}^- < a_0 + b_0c_1 < 2{\xi_d}^+{\omega_n}^+;$$

que é igual a

$$2{,}4 < a_0 + b_0c_1 < 10{,}6667$$

Continuando,

$${\omega_n}^{2^-} < b_0c_0 < {\omega_n}^{2^+};$$

que é igual a

$$4 < b_0c_0 < 44{,}4444$$

Novamente, $b_0 = G_0$. Entretanto, G_0 é intervalar. E $a_0 = \frac{1}{T} = 1$ devido o tempo resposta do sistema ser de 1 segundo. Escrevendo as equações, temos:

$$2{,}4 < 1 + {G_0}^-c_1 < 10{,}6667$$

$$2{,}4 < 1 + {G_0}^+c_1 < 10.6667$$

[5] Permanência das características dinâmicas e de regime projetadas em pontos de operação e linearidades diferentes.

$$4 < G_0^{-} c_0 < 44{,}4444$$

$$4 < G_0^{+} c_0 < 44{,}4444$$

Então:

$$1{,}4 < 0{,}004887 c_1 < 9.6667$$

$$1{,}4 < 0{,}602000 c_1 < 9{,}6667$$

$$4 < 0{,}004887 c_0 < 44.4444$$

$$4 < 0{,}602000 c_0 < 44{,}4444$$

Finalmente

$$\frac{1{,}4}{0{,}004887} < c_1 < \frac{9{,}6667}{0{,}004887}$$

$$\frac{1{,}4}{0{,}602000} < c_1 < \frac{9{,}6667}{0{,}602000}$$

$$\frac{4}{0{,}004887} < c_0 < \frac{44{,}4444}{0{,}004887}$$

$$\frac{4}{0{,}602000} < c_0 < \frac{44{,}4444}{0{,}602000}$$

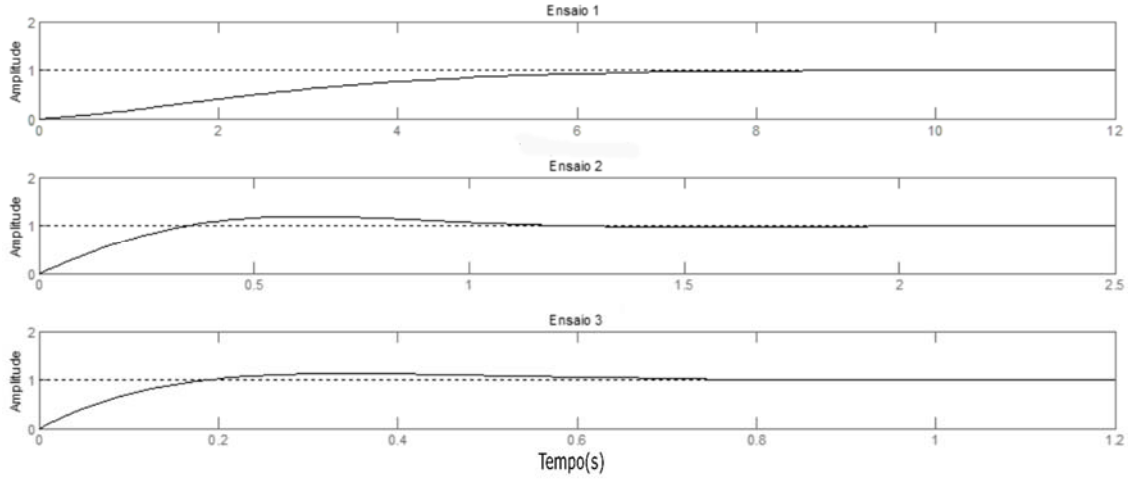
Resolvendo este sistema de inequações, um dos possíveis pontos que satisfazem as relações é: $c_1 = 16{,}0576\ e\ c_0 = 73{,}8280$. Então, o controlador intervalar terá a forma da Equação 3.21.

$$\boldsymbol{Crobusto = \frac{16{,}0576s + 73{,}8280}{s}} \qquad \textbf{3.21}$$

A Figura 3.15 mostra a ação do controlador intervalar na planta simulada. A sintonização intervalar dos parâmetros do controlador PI procura minimizar as diferenças de comportamento entre os pontos de operação. As resoluções das inequações são realizadas a partir de programação linear usando o teorema de Chebyshev. A programação linear é utilizada para executar a otimização convexa de uma função objetivo baseada em um conjunto de

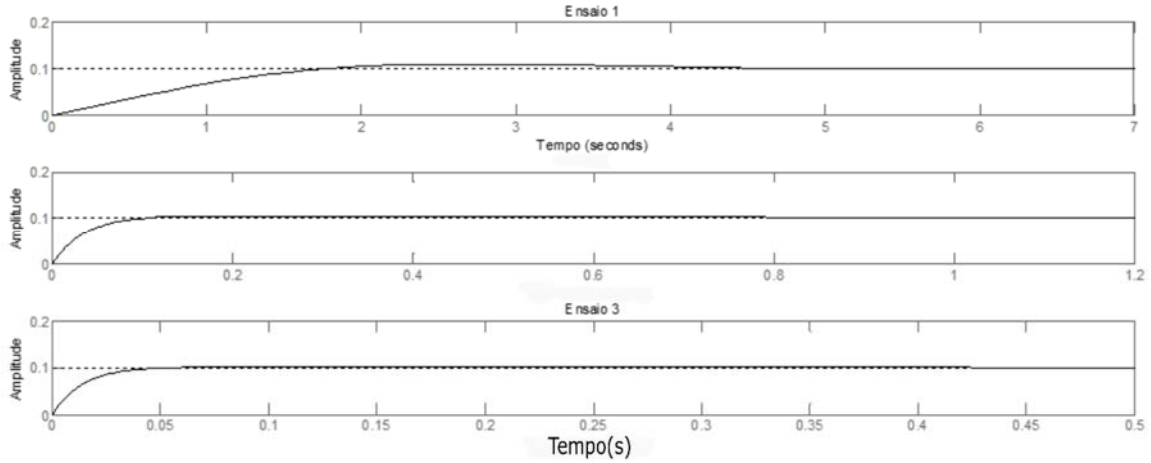
inequações lineares, que abrange incertezas politópicas e intervalares do sistema e calcula os valores ótimos dos parâmetros do controlador, de acordo com as especificações de desempenho do sistema encontrando um ponto, geralmente um vértice do poliedro. Assim, pontos importantes no desenvolvimento, como o *overshoot* e tempo de assentamento, são minimizados nas diferenças existentes nos pontos de operação definidos nos ensaios, causados pela não linearidade do sistema.

Figura 3.15 – Atuação do controlador intervalar na planta fenomenológica



A Figura 3.16 mostra a atuação do controlador clássico nominal nas mesmas condições do controlador intervalar.

Figura 3.16 – Atuação do Controlador Clássico Nominal



Em comparação entre as Figuras 3.15 e 3.16, nota-se que o controlador clássico não garante a semelhança de comportamento da atuação do controlador. O sobressinal de 5% não é obtido nos ensaios 1 e 3, e o tempo de acomodação também varia. Já no controlador intervalar, há certa semelhança, mesmo que o estado transitório não seja igual.

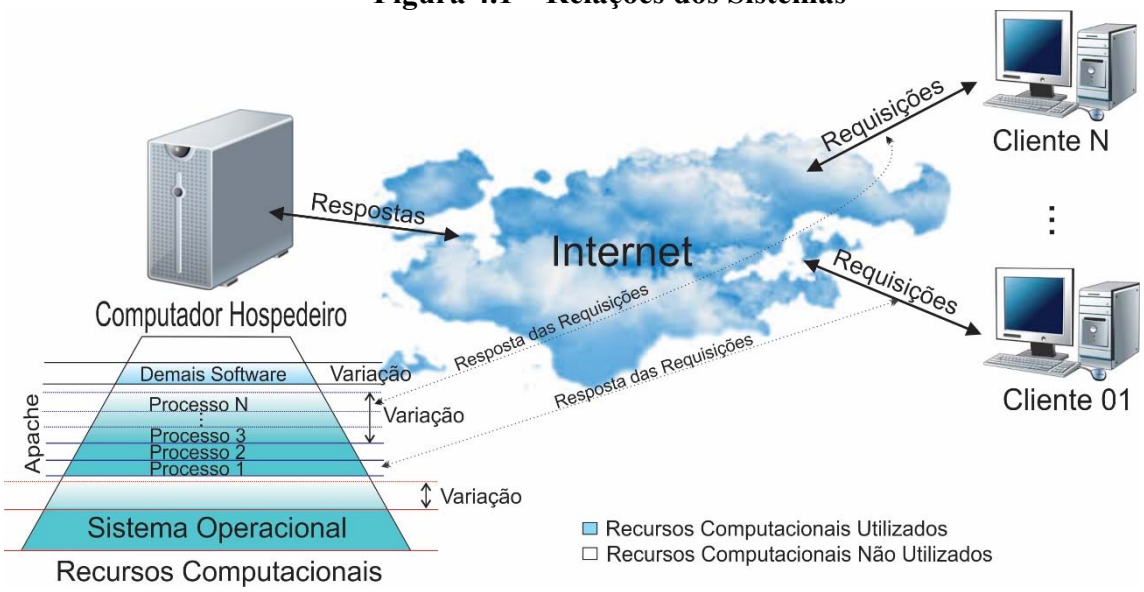
4 – DESENVOLVIMENTO DO AMBIENTE DE TRABALHO

A preocupação de entender e controlar as forças e materiais da natureza é o foco do estudo da engenharia. Os estudantes aplicam as técnicas de controle preocupando-se em entender e em controlar o segmento do ambiente-alvo, denominado geralmente de sistema, para proporcionar produtos economicamente úteis à sociedade. O foco deste capítulo é mostrar como a teoria de controle é aplicada de forma prática em um sistema de computador que provê serviços de resposta a requisições HTTP.

Entretanto, não existe perfeição em sistemas reais. Sempre haverá uma diferença entre o valor desejado e o valor real que o sistema irá produzir. Essa diferença é denominada de erro, de maneira que a aplicação de controle tenta minimizar o erro. É essa diferença que será utilizada para controlar o processo, resultando em uma sequência de operações em malha fechada, que é chamada de sistema com realimentação.

Como um servidor possui vários programas que consomem quantidades de recursos computacionais ao longo do tempo de sua execução, deve-se ter a preocupação em realizar um controle no qual não faltem tais recursos, já que a indisponibilidade de qualquer recurso provoca a interrupção ou atraso do serviço oferecido ao *software* solicitante. A Figura 4.1 mostra a relação dos sistemas e dos recursos computacionais.

Figura 4.1 – Relações dos Sistemas



O estado do sistema mostrado na Figura 4.1 pode ser considerado “bem comportado”, já que mesmo se todos os softwares mostrados ocuparem o máximo da variação de recursos computacionais, ainda existirão recursos a serem utilizados, ou seja, o ideal para o sistema.

Entretanto, essas relações nem sempre são bem comportadas, principalmente em se tratando do serviço de resposta às requisições HTTP.

Como já mencionado neste trabalho, a quantidade de solicitações de uma página "web" não possui um comportamento definido ao longo do tempo para que possam ser realizadas gestões preventivas de alocação de recurso. E mesmo que o gestor de Tecnologia da Informação esteja *full time* em seu local de trabalho realizando um controle de malha aberta (o que encareceria o serviço), não há como equacionar o impacto de uma informação na sociedade com os recursos computacionais necessários. Informações com grande impacto causam, em geral, rajadas de solicitações ao servidor "web", podendo provocar *overload* no sistema computacional.

Para a realização da coletada de dados e controle do sistema de resposta "HTTP", deve-se pensar no ambiente de trabalho (quantidade de recursos computacionais e *softwares* de gerenciamento de recursos da máquina hospedeira) e no método de desenvolvimento do *software* destinado à extração e armazenamento dos parâmetros temporais desejados e controle do sistema "Apache".

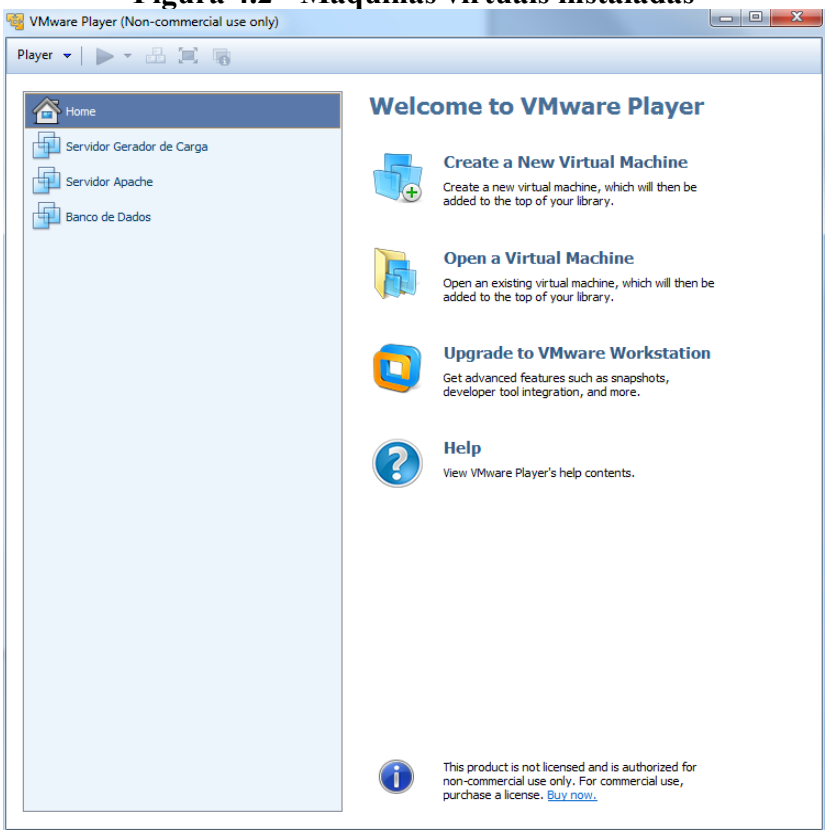
4.1 AMBIENTE DE TRABALHO

O ambiente projetado e desenvolvido para a realização desta tese de doutorado utiliza uma máquina física, com as características descritas na Tabela 4.1, que será a máquina física hospedeira de três máquinas virtuais. Cada máquina virtual possui uma funcionalidade bem definida para o sistema, quais sejam: Servidor "Apache" (resposta das requisições "HTTP" e local de armazenamento e instalação do software de coleta de dados e controle do "Apache"), Servidor de Banco de Dados (repositório dos valores dos parâmetros endógenos e exógenos desejados, relacionando-os com o tempo) e Servidor Gerador de Carga (solicitação de requisição ao servidor "web") (Figura 4.2).

Tabela 4.1 – Máquina utilizada na experimentação

Características	Valor
Processador	Intel i7 CPU Q 740
Memória RAM	6 GB
Sistema Operacional	Windows 7 Home Premium 64Bits
Conexão Ethernet	10/100Mbps

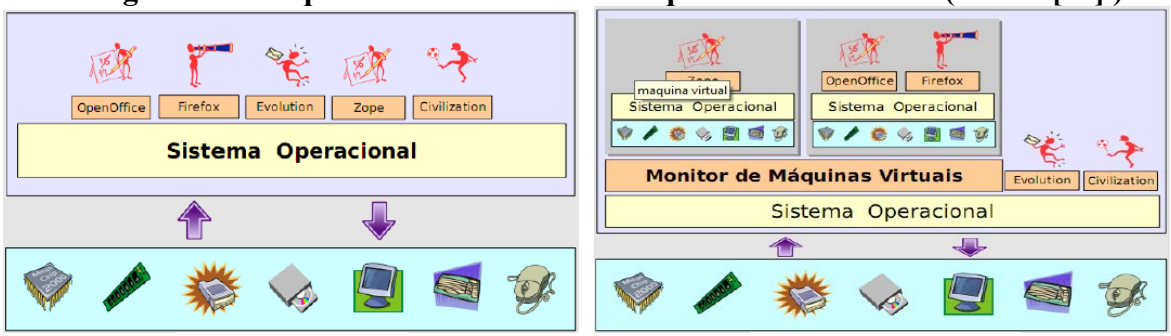
Figura 4.2 - Máquinas virtuais instaladas



A máquina virtual utiliza o conceito de virtualização, que é a técnica para a criação de uma ou mais estações de trabalho ou servidores em um único computador físico, que assume o papel de vários computadores lógicos. Cada máquina virtual oferece um ambiente completo similar a uma máquina física. Com isso, cada máquina virtual pode ter seu próprio sistema operacional, aplicativos e serviços de rede [27].

A Figura 4.3 mostra a arquitetura de uma máquina real e de máquinas virtuais.

Figura 4.3 - Esquema de estrutura de Máquina Real e Virtual (Fonte: [29])



Máquina Real Máquina Virtual

Os recursos computacionais da máquina física são divididos conforme a política adotada pelo gestor de tecnologia da informação no monitor de máquinas virtuais e, uma vez alocado o recurso computacional para uma máquina virtual, este não será utilizado por outra máquina

virtual. Isso impede que um determinado serviço que está disponível na máquina virtual A interfira na execução de outro servido instalado na máquina virtual B, além de impedir também que a máquina física utilize seus recursos na totalidade, causando a parada das instâncias de programas existentes na máquina física.

A eficiente utilização do poder de processamento é comumente chamada de consolidação de servidores. É utilizada principalmente em *data centers*, que geralmente possuem heterogeneidade de plataforma devido aos negócios existes. Outro aspecto que a utilização da virtualização favorece ao *data center* é a redução de custos nos sistemas de refrigeração, cabeamento de rede, energia elétrica, infraestrutura física, suporte e na manutenção a vários sistemas.

Os conceitos de flexibilidade[6] e portabilidade[7] estão implícitos na virtualização, sendo interessante para estudos de Tecnologia da Informação o uso da virtualização em "desktops". Para a pesquisa, é de grande valia poder utilizar várias plataformas em uma única máquina, já que se pode desenvolver produtos para vários sistemas operacionais com uma única plataforma física. Dessa forma, as instâncias das máquinas virtuais são ambientes experimentais, onde as alterações pertinentes não surtirão efeito no sistema operacional original pertencente à máquina física, ou ainda para compor plataformas distribuídas como *clusters* e grades computacionais.

No ambiente desenvolvido para a execução desta tese, houve a divisão dos recursos computacionais entre as máquinas virtuais, conforme as Figuras 4.4, 4.5 e 4.6.

Tal divisão de recursos computacionais se deve ao esforço computacional que cada máquina virtual terá para responder às requisições e para o armazenamento dos dados do sistema de monitoramento, ficando 2GB de memória RAM e 5 núcleos de processamento existentes na máquina física para a execução do sistema operacional da máquina real. Desta forma, há a garantia de que um sistema não interferirá no outro na hipótese de que haja a utilização exagerada ou total dos recursos por qualquer um dos sistemas envolvidos no desenvolvimento deste trabalho.

Na máquina virtual Banco de Dados, está instalado o sistema de gerenciamento de banco de dados "MYSQL" com o banco de dados projetado, que será responsável por armazenar as informações provenientes do monitoramento de recursos da máquina virtual Servidor "Apache" e armazenar as informações de alterações dos parâmetros de configuração do servidor "web".

[6] Escalabilidade facilitada (atendimento de demanda de hoje e futuras).
[7] Facilidade de se adaptar às diferentes plataformas (Sistemas Operacionais).

Com o agrupamento das informações, pode-se realizar um estudo aprofundado de causa e efeito no sistema que está alocado na máquina virtual Servidor “Apache”.

Figura 4.4 - Recursos Computacionais da Máquina Virtual Servidor Gerador de Carga

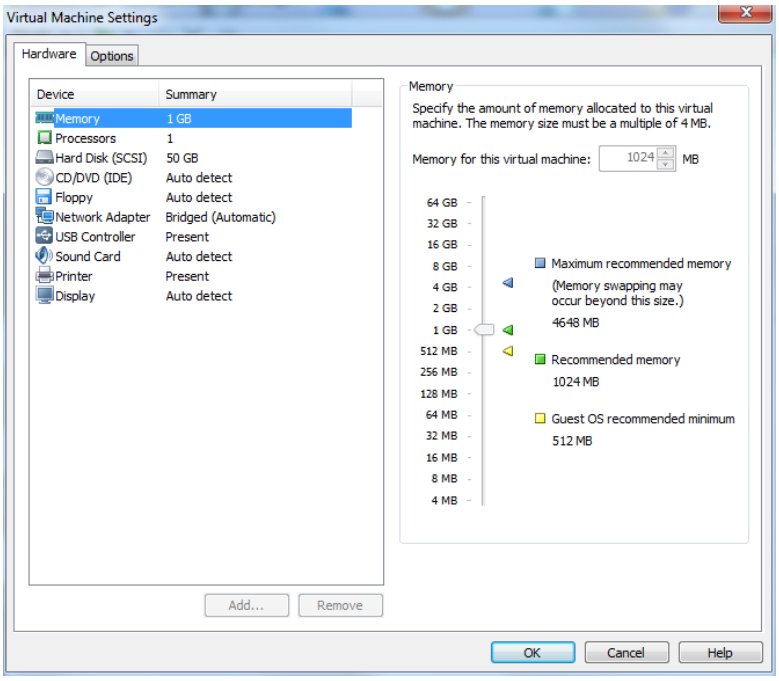
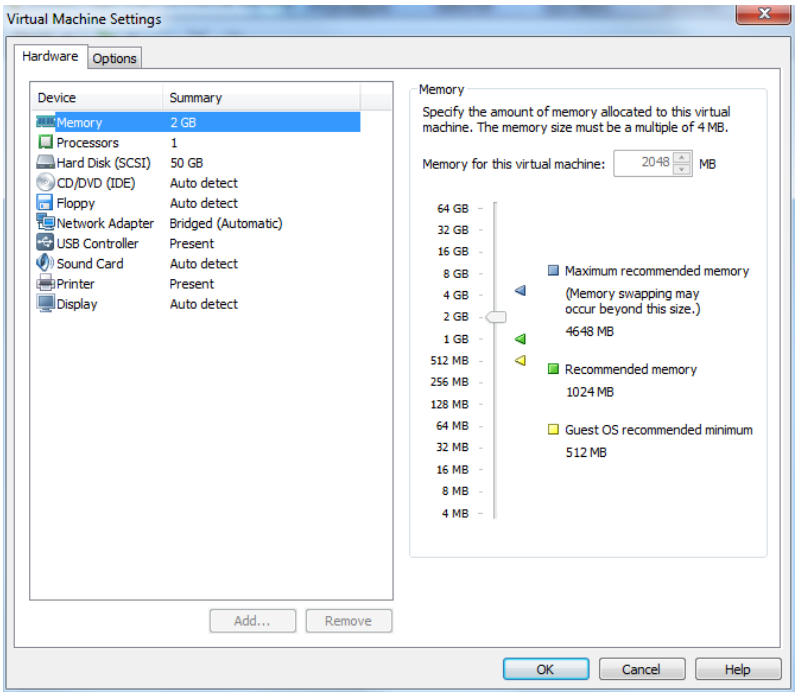
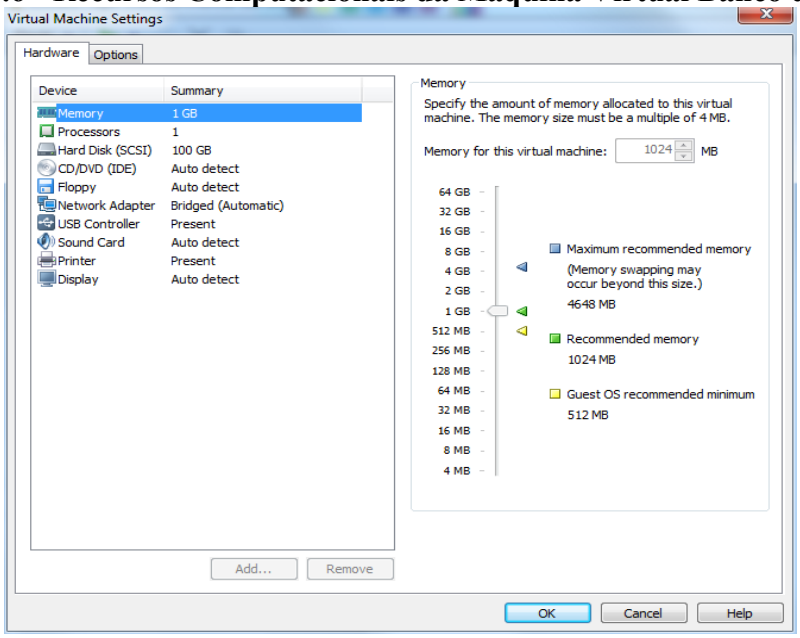


Figura 4.5 - Recursos Computacional da Máquina Virtual Servidor “Apache”



As máquinas virtuais “Servidor Gerador de Carga”, “Servidor Apache” e “Banco de Dados” possuem o “Ubuntu” como sistema operacional. O “Ubuntu” é uma versão do Linux cujo escalonador está baseado em tarefas e prioridades.

Figura 4.6 - Recursos Computacionais da Máquina Virtual Banco de Dados



Na máquina virtual Servidor Gerador de Carga está instalado o software “httperf”, que é responsável por realizar as chamadas de requisições de forma planejada. O “httperf” é uma ferramenta para medir o desempenho do servidor “web”, fornecendo um mecanismo flexível para gerar cargas de trabalho HTTP e para medir o desempenho do servidor. Uma característica desse *software* é, ainda, a de sustentar a sobre carga no servidor.

Dessa forma, portanto, cria-se um ambiente com três máquinas, cada uma com seu propósito, e completamente isoladas entre si. A comunicação entre elas é realizada através do sistema de rede. Entretanto, não há saída dos dados da máquina real, diminuindo o tempo de latência, já que não se utiliza equipamento de distribuição de redes e cabos para a comunicação entre as máquinas virtuais.

4.2 O *SOFTWARE*

O *software* desenvolvido tem a principal característica de ser não intrusivo (Anexo I) ao sistema “Apache”, ou seja, não é necessário fazer alterações no código fonte do “Apache” para automatizar a funcionalidade desejada. Ele irá armazenar dados dos instantes de testes e fará o controle do valor de “MaxClient” para que o servidor tenha sempre memória disponível para a execução de rotinas do sistema operacional.

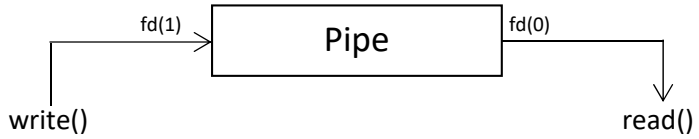
O *software* proposto é um gerenciador de processos que trabalha em uma camada superior ao gerenciador de processos do servidor “Apache”, promovendo uma dinâmica nos valores descritos no arquivo “httpd-mpm.conf” no “Apache 2.4.6” sem a interrupção do serviço.

O *software* possui uma gestão de multiprocessos, idênticas ao módulo de multiprocessamento "prefork" do "Apache", o qual utiliza a criação de processos filhos a partir do processo pai. Os processos filhos utilizam as variáveis de forma independente a partir de sua criação, sem alterar as variáveis dos outros processos filhos ou do processo pai. O processamento das rotinas é realizado mediante a fila no sistema operacional e o nível de prioridade existente para cada processo.

Em linguagem de programação, a primitiva de criação para método de multiprocessamento pretendido é "fork()". Essa primitiva é a única chamada de sistema que possibilita a criação de um processo em UNIX. Os processos pai e filho partilham o mesmo código. O segmento de dados do usuário do novo processo (filho) é uma cópia exata do segmento correspondente ao processo antigo (pai). Por outro lado, a cópia do segmento de dados do filho do sistema pode diferir do segmento do pai em alguns atributos específicos (como por exemplo, o "pid", o tempo de execução etc.). Os filhos herdam uma duplicata de todos os descritores dos arquivos abertos do pai (se o filho fecha um deles, a cópia do pai não será modificada). Mais ainda, os ponteiros para os arquivos associados são divididos (se o filho movimenta o ponteiro dentro de um arquivo, a próxima manipulação do pai será feita a partir desta nova posição do ponteiro). Essa noção é muito importante para a implementação dos *pipes* (tubos) entre processos.

Os *pipes* servem para fazer a comunicação entre dois processos, sendo que um processo escreve e outro processo lê o *pipe*. Dessa forma, os *pipes* possuem sentido para escrever e ler. Para identificar a escrita ("write()") e a leitura("read()"), utilizou-se os "file descriptors", que são dois números inteiros que vão definir a entrada padrão e a saída padrão do *pipe*, ou seja, um vetor de inteiros de duas posições. A Figura 4.7 mostra o esquema do *pipe*.

Figura 4.7 - Esquema de escrita e leitura no *pipe*



A estrutura do software foi projetada para que as execuções de coleta de dados e a execução de controle fossem independentes. A coleta de dados é responsável por obter os dados do sistema "LINUX" e enviá-los para um banco de dados. A execução de controle é responsável por realizar os cálculos de controle e aplicar o sistema de resposta às requisições "HTTP".

Para adquirir as informações necessárias do sistema “LINUX”, utiliza-se a função “free” para obter a quantidade de memória usada.

Entretanto, as informações não estão tabuladas e nem formatadas para a utilização do *software*. Inicialmente, muda-se o dispositivo principal de saída (vídeo) para um local na memória endereçável através da estrutura de *pipe*. Com os dados em forma de texto, deve-se realizar a busca (saber o exato ponto do texto para a extração da informação a partir da padronização utilizada) e preparar as informações em formato compreensível pela máquina.

As informações devem ser armazenadas em um banco de dados motivado pela quantidade de informação que será gravada no dispositivo secundário, já que o tempo de amostragem é de 1 segundo. Para o armazenamento, utiliza-se o módulo “mysql/mysql.h”, que deve ser inserido no código do *software* e, no momento da compilação, utilizar o parâmetro “-lmysqlclient” para a inserção/ligação com o módulo.

O nome do módulo indica qual é o sistema gerenciador de banco de dados (SGBD) utilizado: “MYSQL V 5.5.32”. A parte lógica do banco tem a seguinte estrutura escrita em linguagem de busca estruturada (Structured Query Language - SQL). O script está disponível no Anexo II.

O banco de dados possui uma tabela denominada “HTTP01” e um procedimento denominado de Status. A tabela possui como chave primária simples o campo id e uma configuração de autoincremento, possibilitando a gerência do valor a ser inserido no campo id. Problemas de reuso do valor gerado pelo autoincremento não existem devido não haver motivação para alterar ou excluir tuplas desta relação. O procedimento Status serve para ter o mínimo de segurança nas informações, já que informações como o nome da relação, onde se encontram os dados do servidor “web”, são abstraídas no programa que chama esse procedimento. Além de facilitar a programação no sentido de diminuir erros de sintaxe e semântica.

As tuplas da relação fazem a ligação entre os valores dos parâmetros do aplicativo de resposta “HTTP” com os valores dos recursos computacionais. Essa estrutura de armazenamento foi utilizada para a coleta de dados em malha aberta e fechada.

O modelo do *software* foi criado na concepção da análise estruturada moderna, motivado por ser um *software* de caráter específico. Segundo [28], a metodologia de desenvolvimento para um *software* específico deve ser “botton-up”, já que o objetivo principal é satisfazer os requisitos específicos encontrados no ambiente em questão.

O uso da implementação “botton-up” é uma das formas para análise do problema em análise de sistemas. Para o ciclo de vida do *software* foi escolhida a espiral prototipação. Espera-se que o programa seja desenvolvido em módulos e cada módulo devidamente testado, e, em seguida, os testes dos subsistemas e, por fim, o teste do sistema.

As ferramentas mais comuns na Análise Estruturada Moderna são: Diagrama de Fluxo de Dados, Diagrama de Entidade e Relacionamento, Diagrama de Transição de Estado e o Diagrama Estrutural.

Para ser realizado um desenvolvimento de sistema, deve-se envolver os dados e os processos, já que sem a participação dos dois componentes é impossível construir um sistema. Como será realizado o processamento é um aspecto importante para ser modelado e examinado com o usuário. A modelagem é descrita de várias maneiras:

- Que funções deve o sistema executar? Quais são as interações entre as funções?
- Que transformações deve executar o sistema? Que entradas são transformadas em que saída?
- Que espécie de trabalho faz o sistema? Onde ele obtém a informação para fazê-lo? Para onde ele remete os resultados do trabalho?

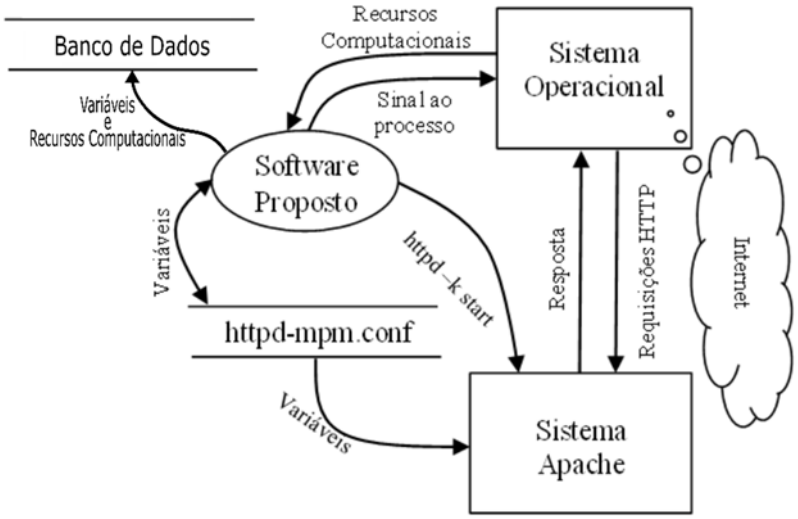
A ferramenta que descreve a transformação de entradas em saída, nesta modelagem, é o diagrama de fluxo de dados, que possui as seguintes partes: processos, depósitos de dados, fluxos e terminais.

1. Processos são escritos em forma de círculo ou “bolhas” e representam as diversas funções individuais que o sistema executa. Funções que transformam entradas em saídas.
2. Fluxos são mostrados por setas (podem ser curvas). Eles são as conexões entre os processos (funções do sistema) e representam a informação que os processos exigem como entrada e/ou as informações que eles geram como saída.
3. Depósito de Dados é representado por duas linhas paralelas ou por uma elipse. Mostra a coleção de dados (agregados) que o sistema deve manter na memória por um período de tempo. Após a conclusão do sistema, o depósito de dados existirá como um banco de dados ou arquivos.
4. Terminadores são representados por quadrados e mostram as entidades externas ao sistema com as quais ele se comunica (e.g. indivíduos, grupos de pessoas, sistemas externos de computadores e organizações externas).

Para sistemas em tempo real, há a necessidade de modelar fluxos e processos de controle. Os fluxos de controle são sinais ou interrupções para “stop” ou “start” de um processo comum. Os processos de controle são elipses cuja tarefa é coordenar ou sincronizar as atividades de outras bolhas do DFD. O fluxo de controle e o processo de controle são representados pelas mesmas formas já mencionadas, mas com suas linhas tracejadas.

O diagrama de fluxo de dados de alto nível do software proposto é mostrado na Figura 4.8. Este diagrama mostra como é o relacionamento do software com os componentes externos (Sistema Operacional, Sistema “Apache”, arquivo de configuração e o banco de dados) através dos fluxos de dados existentes com indicação de sentido de envio da informação. A internet não possui relação direta com o software proposto uma vez que o sistema operacional é responsável por fazer tal tarefa.

Figura 4.8 - Diagrama de Fluxo de dados de alto nível



Na Figura 4.8, deve-se observar quais são os fluxos de dados que saem do software proposto, quais estão entrando e quais têm duplo sentido, pois estes são os dados que influenciaram no processamento do sistema computacional.

O fluxo “sinal de processo” é responsável pelo envio, ao sistema operacional, das solicitações de interrupções aos processos “httpd” necessárias para o gerenciamento do sistema computacional.

O fluxo de dados “variável”, como tem duplo sentido, irá ler e escrever no arquivo repositório (httpd-mpm.conf) os valores de parâmetros de configuração do sistema “Apache”. Como já mencionado, são estes os valores utilizados para a inicialização do sistema “Apache”.

O fluxo de dados “httpd –k start” é uma chamada ao sistema para inicialização do serviço de resposta às requisições “HTTP”, ou seja, obedecendo às características de não

intrusivo. Esse fluxo será ativado após a alteração do arquivo "httpd-mpm.conf", pois os novos valores é que irão compor o sistema "Apache".

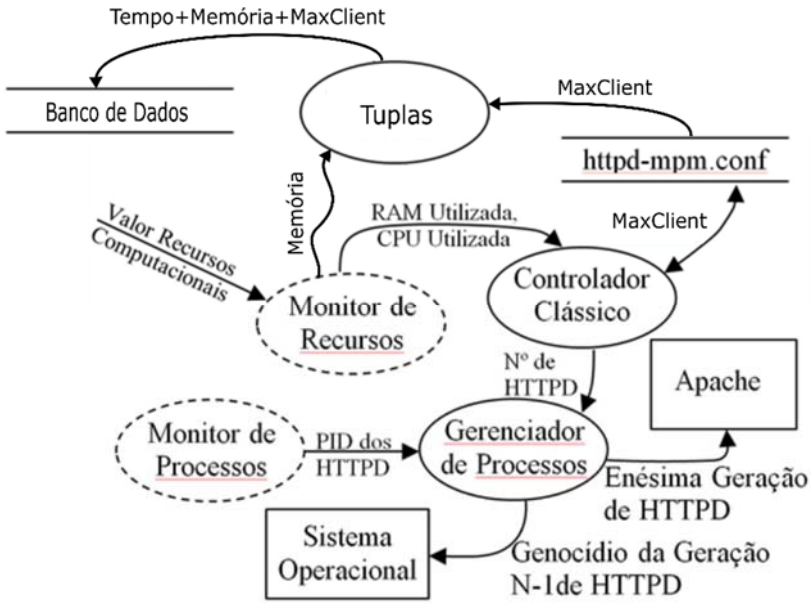
O sistema "Apache" continua sua comunicação livremente com o sistema operacional sem influência do *software* proposto. O fluxo de dados "recursos computacionais" envia ao *software* os valores dos recursos computacionais que servirão para o cálculo de controle.

O fluxo de dados "variáveis e recursos computacionais" envia para o banco de dados os valores dos parâmetros de configuração e os valores de recursos computacionais com dependência funcional (subconjunto escolhido entre as superchaves mínimas, ou seja, chave primária) no tempo que foram extraídas da máquina virtual hospedeira. Esses dados relacionados serão utilizados para a identificação do sistema computacional.

Na Figura 4.9 o *software* proposto é expandido, mostrando como os dados que entram no processo "*software* proposto" são processados (metodologia para resolução do problema de forma automática), mandando sinais ao sistema operacional, comandos de execução e solicitação de armazenamento de informações.

Existem dois tipos de processos: os que estão no "background" e os que estão ativos por intermédio de outro processo ou por um terminador. Os processos de "background" são ativados conforme a sua programação, ativado de tempos em tempos, ou por eventos, ativados por uma chamada do sistema operacional (processos de controle). Os processos que não estão em "background" são ativados por usuários e permanecerão ativos até que o usuário do termine o processo.

Figura 4.9 - Diagrama de Fluxo de dados de baixo nível



O processo “monitor de recursos” é responsável por receber informações do sistema operacional em formato de texto e é executado em *background* por ter sido criado a partir da função “fork”. O “monitor de recursos” deve extrair do texto, enviado pelo sistema operacional, a informação desejada (quantidade de memória ocupada). Essa informação chega ao “monitor de recursos” devido a alteração realizada de direcionamento de saída do “pipe” do monitor do sistema operacional, ou seja, o resultado da solicitação do comando “free” não irá para a “buffer” de saída padrão, mas para o “monitor de recursos”. O valor utilizado para a contagem da memória utilizada será o valor “used”-“buffers”-“cached”, eliminando o “drift” do comportamento o sistema computacional (Figura 4.10). Após receber e preparar a informação é enviado para os processos “tuplas” e “controlador clássico”.

Figura 4.10 - Comando “free”

```
root@ServerApache:/usr/local/httpd-2.4.6# free
             total       used       free     shared    buffers     cached
Mem:       2044060    1339436     704624          0     240480     427740
-/+ buffers/cache:     671216    1372844
Swap:      2094076          0    2094076
```

O processo “controlador clássico” é onde o controlador PI está e não é executado em “background”, modo em que o início e término da rotina programada sofrem interferência do usuário. O gerente de tecnologia e informação informa a quantidade de memória que pode ser ocupada na máquina hospedeira virtual e o controlador irá adaptar o número de processos “httpd” ao valor de referência informado automaticamente no arquivo “httpd-mpm.conf”. O fluxograma principal do processo “controlador clássico” é mostrado na Figura 4.11, sendo “R0” e “R1” os parâmetros do controlador retroalimentado a serem sintonizados, “Ref” sendo o valor de referência definida pelo administrador do sistema computacional, PU_Usada é a memória usada por unidade. A alteração no arquivo de configuração do “Apache” é realizada por uma chamada ao procedimento “gerenciador de processos”.

Figura 4.11 - Linha de código controlador PI

```
PU_Usada = (float)MemoriaUsada/MemoriaTotal;
Yk = PU_Usada;
Ek_1 = Ek;
Ek = Ref - Yk;
Uk_1 = Uk;
Uk = Uk_1 + floor(R0*Ek) + floor(R1*Ek_1);
MaxRequestWorkers = Uk;
```

O parâmetro “MaxRequestWorks” é o mesmo do “MaxClient”, a denominação foi alterada na versão do “Apache 4.12” utilizada para o teste do software, conforme exposto

anteriormente. Os demais parâmetros serão projetados no próximo capítulo a partir da coleta de dados do sistema real.

O processo “monitor de processo” relaciona os processos “httpd” existentes com a geração de processo correspondente, ou seja, a cada alteração do parâmetro “MaxClient” no servidor “Apache” os processos criados a partir da efetivação dos novos valores, serão de uma nova geração. O relacionamento é armazenado em uma fila com estrutura de registro tendo o “PID” como chave primária. O funcionamento do “monitor de processo” é em “background”.

O processo “gerenciador de processos” indica e solicita ao sistema operacional qual processo “httpd” do usuário “Daemon” deve ser morto e solicita ao servidor “Apache” a criação de uma nova geração de processos com as novas configurações, com base na informação enviada pelo processo “controlador clássico”. A Figura 4.12 mostra o resultado da execução desse módulo, onde duas gerações coexistem no sistema hospedeiro.

Figura 4.12 - Gerações Diferentes em Execução

```
root@ServerApache:/usr/local/httpd-2.4.6/bin# ps aux | grep -i httpd
daemon    4118  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4119  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4120  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4121  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
daemon    4122  0.0  0.1 106644  3892 ?        S    10:29   0:00 ./httpd -k start
root      4168  0.0  0.2 106644  5636 ?        Ss   10:33   0:00 ./httpd -k start
daemon    4169  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4170  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4171  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4172  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
daemon    4173  0.0  0.1 106644  3892 ?        S    10:33   0:00 ./httpd -k start
root      4175  0.0  0.0  13628   964 pts/0    S+   10:33   0:00 grep --color=auto -i httpd
root@ServerApache:/usr/local/httpd-2.4.6/bin#
```

A Figura 4.12 mostra a primeira ação do processo “gerenciador de processos”. Os processos de “PID” 4118, 4119, 4120, 4121 e 4122 são da geração anterior, ou seja, parâmetros não mais adequados para a condição da máquina hospedeira e serão extintos quando não mais responderem às solicitações. Os processos de “PID” 4168, 4169, 4170, 4171,4172 e 4173 são da nova geração com os parâmetros já ajustados. O processo que possui o “owner” igual a “root” fará o gerenciamento da nova geração na metodologia do “Apache” até uma nova intervenção do *software* proposto.

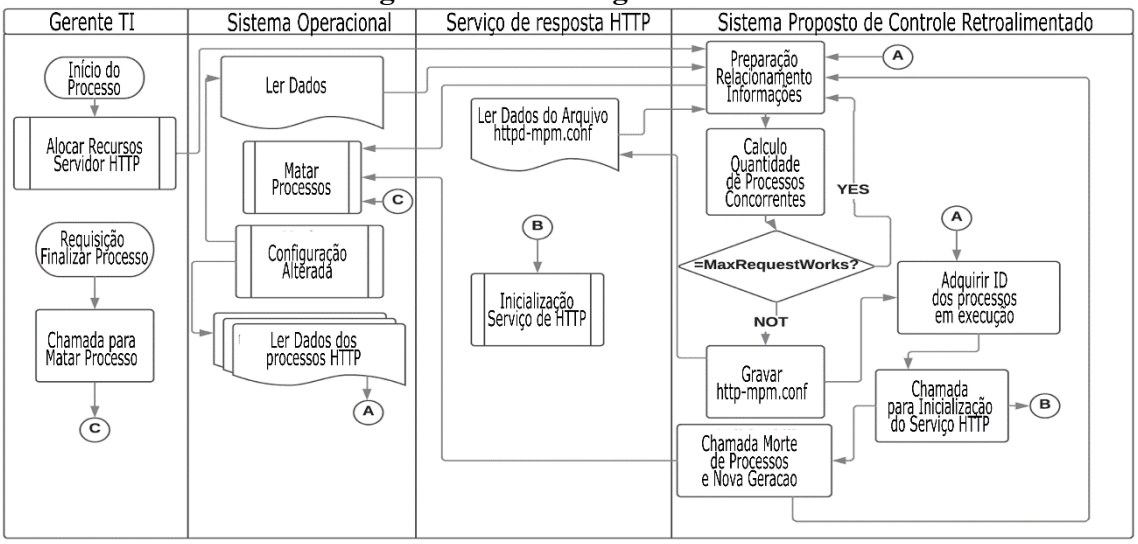
O processo “tupla” é responsável pela relação dos dados memória utilizada, “MaxClient” e tempo. Como pode ser observado, o processo não é executado em “background”, sendo necessária a interferência do usuário. As amostras coletadas dos ensaios do sistema computacional são extraídas por este módulo e enviadas ao sistema gerenciador de banco de dados para o armazenamento. A extração do banco dados para outro sistema como “MATLAB” é simples.

O fluxograma é uma representação gráfica do algoritmo através de formas geométricas, facilitando a compreensão da lógica utilizada pelo profissional e permitindo a descrição clara e precisa do fluxo, ou sequência, de um processo, bem como sua análise e redesenho.

O fluxograma é utilizado também de várias outras maneiras, não somente como representação gráfica de um algoritmo. É muito utilizado também nas empresas para demonstrar os processos de entrada, processamento e saída de uma produção ou de um projeto. Os principais aspectos nessa forma de utilização do fluxograma são: padronizar a representação dos métodos e os procedimentos administrativos, maior rapidez na descrição dos métodos administrativos, facilitar a leitura e o entendimento, facilitar a localização e a identificação dos aspectos mais importantes, maior flexibilidade e melhor grau de análise.

A fim de enfatizar as interações entre o sistema de controle proposto e o ambiente de *software* subjacente, ou seja, o sistema operacional e o serviço "HTTP", a Figura 4.13 mostra o fluxograma de processo correspondente envolvendo o sistema computacional por completo. Como pode ser visto, o controlador digital PI é executado como uma tarefa pertencente ao processo chamado "Cálculo da quantidade atual de processo HTTP". Se um evento de perturbação for detectado por uma tarefa de supervisão, o sistema de controle proposto realiza o ajuste em "turn on" das configurações do servidor de resposta "HTTP".

Figura 4.13 - Fluxograma Geral



Apesar de o DFD e o Fluxograma proporcionarem uma prática visão geral dos componentes funcionais do sistema, os detalhes são suprimidos. As ferramentas que são utilizadas para dirimir os detalhes são: dicionário de dados e as especificações dos processos (parte realizada ao longo do capítulo informando o que cada processo faz, faltando os diagramas de estados dos processos que mostrará os pré-requisitos necessários ao processamento).

Esquemas de notação de dicionário de dados devem ser utilizados para normatizar a leitura e interpretação. Uma das notações mais utilizadas para esta ferramenta utiliza os seguintes símbolos mostrados na Tabela 4.2.

Tabela 4.2 – Símbolos de notação do dicionário de dados

Símbolo	Notação
=	é composto de
+	E
()	opcional (pode estar presente ou ausente)
{}	Interação
[]	escolha uma das opções alternativas
**	Comentário
@	identificação (campo chave de um depósito)
|	separa opções alternativas na condição []

O dicionário de dados do *software* proposto segue conforme a Tabela 4.3. Aqui será apresentada a composição de cada fluxo existente e como cada dado é formado. Não são expostos os tipos de variável primitiva de cada dado nesse momento, estas informações estão expostas no Anexo I.

Tabela 4.3 – Dicionário de Dados

Fluxo de dados	Descrição
Controle do Sistema Apache =	**memória total do computador + memória usada + memória livre + %CPU usada pelo Usuário + %CPU usada pelo sistema + % CPU livre + StartServers + MinSpareServers + CMaxSpareServers + MaxRequestWorkers + MaxConnectionsPerChild + ServerLimit + KeepAliveTimeOut**
Tela do prompt =	** Todos os dados visualizados em tela pelo usuário ao solicitar o comando *top* e *men***
MaxClient e *KeepAliveTimeOut* =	**Valor do StartServers + MinSpareServers + CMaxSpareServers + MaxRequestWorkers + MaxConnectionsPerChild + ServerLimit + KeepAliveTimeOut**

Arquivo de Configuração =	**Todos os dados contidos no arquivo de configuração do "Apache" "*httpd-mpm.conf*" **
Novo Arquivo de Configuração =	**Alteração do arquivo de configuração "*httpd-mpm.conf" no valores de* StartServers + MinSpareServers + CMaxSpareServers + MaxRequestWorkers + MaxConnectionsPerChild + ServerLimit + KeepAliveTimeOut**
MaxClient =	** Valor do MaxRequestWorkers que define o número de requisições que são atendidas em paralelo; intervalo: 1-600**
Sinal de Controle =	**Valor de MaxRequestWorkers calculado pelo controlador PI observados os recursos computacionais e o valor anterior de MaxRequestWorkers; intervalo: 1-600**
Sinal de Início =	**Comando que informa uma nova configuração do sistema "Apache" a ser aplicada**
Sinal *MaxClient* SBPA =	**Valor do "MaxRequestWorkers" gerado a partir de um sinal pseudoaleatório; intervalo de 50-150
Memória =	** quantificado em sistema por unidade (PU) sendo a resultado da divisão de memória usada por memória total**
Memória e %CPU =	** memória total do computador + memória usada + memória livre + %CPU usada pelo Usuário + %CPU usada pelo sistema + % CPU livre**
Informações S.O. =	**informações em formato de texto contendo os dados de memória total do computador + memória usada + memória livre + %CPU usada pelo Usuário + %CPU usada pelo sistema + % CPU livre**

Memória total do computador	**Valor numérico inteiro positivo em bytes correspondente à memória RAM, que depende do recurso computacional alocado para máquina virtual**
Memória Usada	**Valor numérico inteiro positivo em bytes correspondente à memória RAM sendo utilizada no computador, incluindo sistema operacional, demais aplicações e o sistema de resposta às requisições HTTP **
Memória livre	**Valor numérico inteiro positivo em bytes correspondente à memória RAM que não está sendo utilizada no computador, incluindo sistema operacional, demais aplicações e o sistema de resposta às requisições HTTP **
%CPU usada pelo Usuário	**Valor numérico em porcentagem correspondente ao uso da CPU pelo usuário; intervalo de 0-100**
%CPU usada pelo sistema	**Valor numérico em porcentagem correspondente ao uso da CPU pelo sistema operacional; intervalo de 0-100**
% CPU livre	**Valor numérico em porcentagem correspondente ao não uso da CPU; intervalo de 0-100**
StartServers	**Quantitativo, inteiro positivo, de processos filhos que devem ser inicializados, independentemente de haver requisições de HTTP,**
MinSpareServers	**Quantitativo inteiro positivo mínimo de processos filhos pode ficar no estado de espera**
CMaxSpareServers	**Quantitativo inteiro positivo máximo de processos filhos pode ficar no estado de espera**

MaxRequestWorkers	**Quantitativo inteiro positivo máximo de processos filhos pode ser criado para responder às solicitações de HTTP**
MaxConnectionsPerChild	**Número inteiro positivo máximo de conexões por processo filho**
ServerLimit	**Idêntico ao MaxRequestWorkers por limitar o número de processos filhos**
KeepAliveTimeOut	**Tempo em segundos que limita o tempo do estado de espera de um processo filho**

Até o presente momento, foram utilizadas ferramentas de modelagem que realçam as funções desempenhadas (Diagrama de Fluxo de Dados e fluxogramas) por um sistema e os dados que o sistema deve memorizar (Dicionário de Dados). O Diagrama de Transição de Estado (DTE) é a ferramenta de modelagem que mostra o comportamento tempo-dependente do sistema.

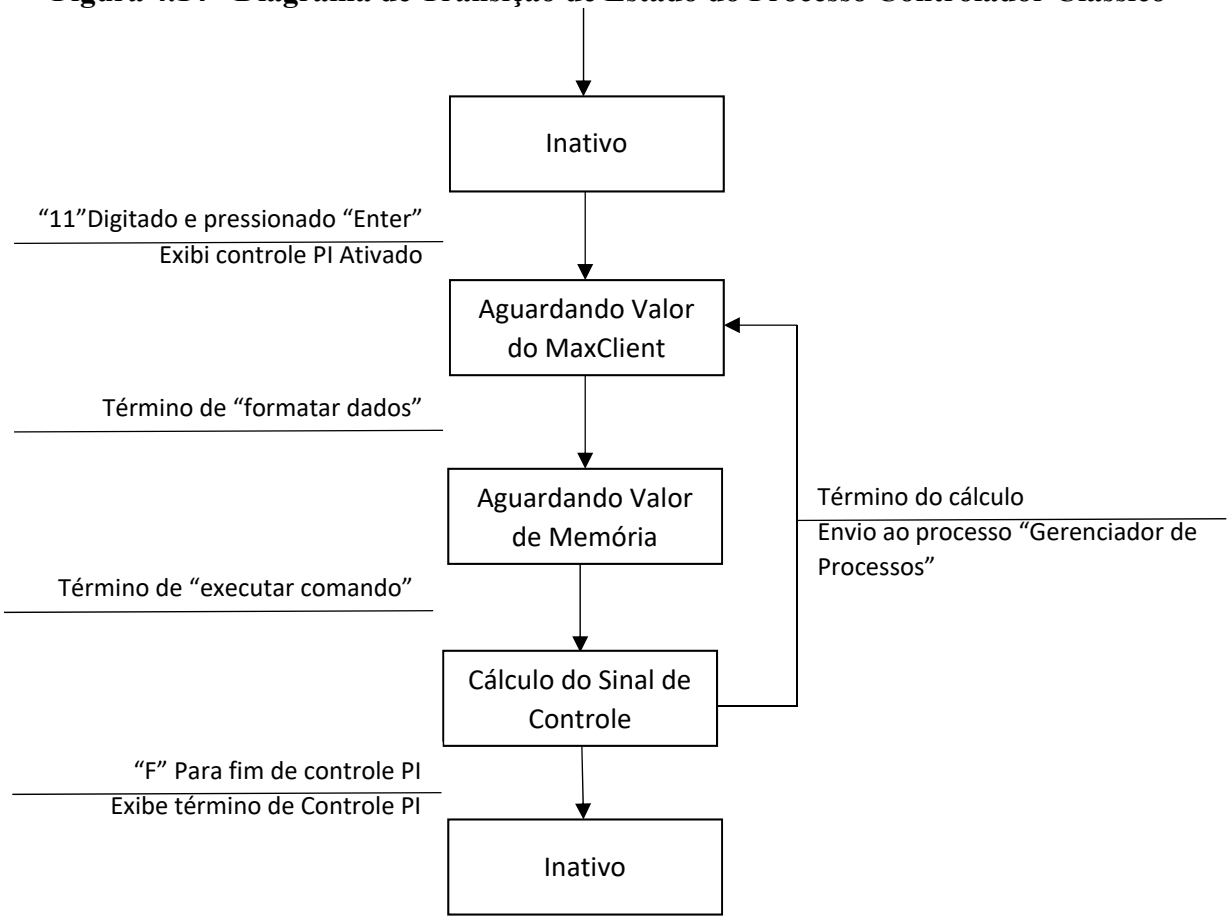
Os principais componentes do DTE são os estados e as setas, que mostram as alterações no estado do sistema. O estado do sistema é definido pelo "*Webster's New World Dictionary"* como sendo "o conjunto de circunstâncias ou atributos que caracterizam uma pessoa ou objeto em determinado momento, modo ou forma de ser; condição". As alterações no estado, setas, são as modificações válidas, ou seja, os possíveis arranjos de estados (e.g., sair do estado 1 para o estado 2 , sair do estado 2 para o estado 3, não poder sair do estado 1 e passar direto para o estado 3).

O diagrama de transição de estados pode ser utilizado de duas formas: a forma isolada e combinando com outras ferramentas de modelagem (DFD, casos mais comuns). Utilizando-se o DTE em combinação com o DFD, o diagrama de transição de estados representa a especificação de processo de uma bolha de controle do diagrama de fluxo de dados.

O diagrama da Figura 4.14 mostra que o processo controlador clássico possui 4 estados. Para iniciar o estado "Aguardando Valor do Maxclient", o usuário terá que solicitar ao *software*, pressionando o dígito 11 mais o botão "ENTER", ativando o estado. Iniciado o estado, fará o processamento para obter valor do "MaxClient" e somente sairá deste estado após formatar o arquivo "httpd-mpm_conf" e adquirir o valor de "MaxClient". Com valor armazenado em uma variável do processo "controlador clássico", é iniciado o estado de "Aguardando Valor de Memória". Nesse estado, é realizada uma chamada ao processo "monitor de recurso", o qual enviará a este processo o valor da quantidade de memória ocupada. Obtidos o valor de

"Maxclient" e o de memória ocupada, será realizado o cálculo de sinal de controle que envia para o processo de "Gerenciado de Processos". Caso o usuário permaneça utilizando o controlador clássico, o processo retornará ao estado de "aguardando valor de MaxClient". A negativa da permanência em *loop* do algoritmo, processo controlador entrando em estado de inativo.

Figura 4.14 - Diagrama de Transição de Estado do Processo Controlador Clássico

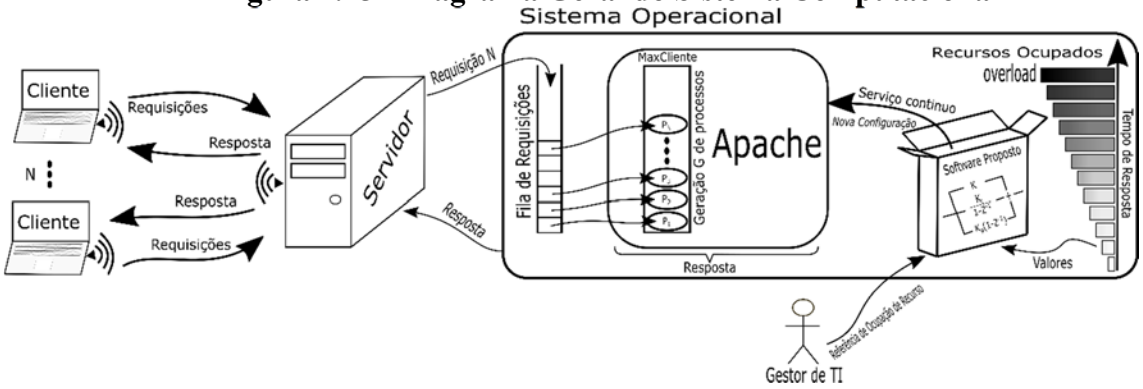


Os demais processos não necessitam de diagrama de transição de estado, pois as informações estão todas disponíveis a todo instante, não tendo especificações para que ocorra o processamento. A única exceção é o processo "Tuplas", que precisa aguardar o processamento dos demais processos. Entretanto, é bem intuitivo que primeiramente deve-se aguardar as informações para relacioná-las e somente depois enviar para o sistema gerenciador de banco de dados para armazenamento.

Para finalizar este capítulo, a Figura 4.15 é um esquema geral, sem utilização de ferramenta de desenvolvimento de *software*, que mostra a atuação do software proposto, seu relacionamento com outros sistemas, em que a teoria de filas se encontra no problema e a atuação simples do gesto de TI em uma atribuição complexa. Atuações sem conhecimento

específico podem provocar a parada completa do serviço de resposta a requisições “HTTP”, trazendo prejuízos à organização provedora do serviço.

Figura 4.15 - Diagrama Geral do Sistema Computacional



Com o ambiente de trabalho desenvolvido e implementado no servidor de respostas às requisições “HTTP” “Apache”, será possível armazenar amostras dos ensaios em malha aberta ou fechada, implementar controladores a partir das mais variadas teorias de controle, e.g., controlador nominal e intervalar, e verificar a efetividade do controlador na planta. Com os controladores implementados é possível impedir *overload* e prolongar a vida útil da máquina hospedeira do sistema de computação em foco.

5 – RESULTADOS

Com o *software* pronto, é possível retirar os dados do sistema real para o projeto do controlador clássico PI e fazer a implementação desse controlador. Assim, neste capítulo teremos os resultados da identificação do sistema real e da atuação do controlador projetado a partir dos dados da identificação.

Um sistema real possui características que devem ser estudas para proporcionar uma melhora no serviço ou produto pretendido. A modelagem matemática proporciona o desenvolvimento de uma equação que possua o mesmo comportamento das características desejadas de um sistema real.

O modelo matemático é uma representação aproximada das características do sistema-alvo. Aqui, deseja-se que o modelo represente a evolução temporal do sistema, ou seja, o comportamento dinâmico. Vale ressaltar que, por ser uma aproximação do sistema real, existe um conjunto de modelos com desempenho e características variados que representam o sistema real e a escolha de um desses modelos não é uma tarefa simples.

A simplicidade é sempre desejável no desenvolvimento de um modelo. Caso o projetista deseje agrupar várias características do sistema real em um único modelo, provavelmente a conclusão do trabalho não será alcançada. Assim, deve-se escolher apenas algumas características, simplificando o desenvolvimento do modelo matemático adequado [25].

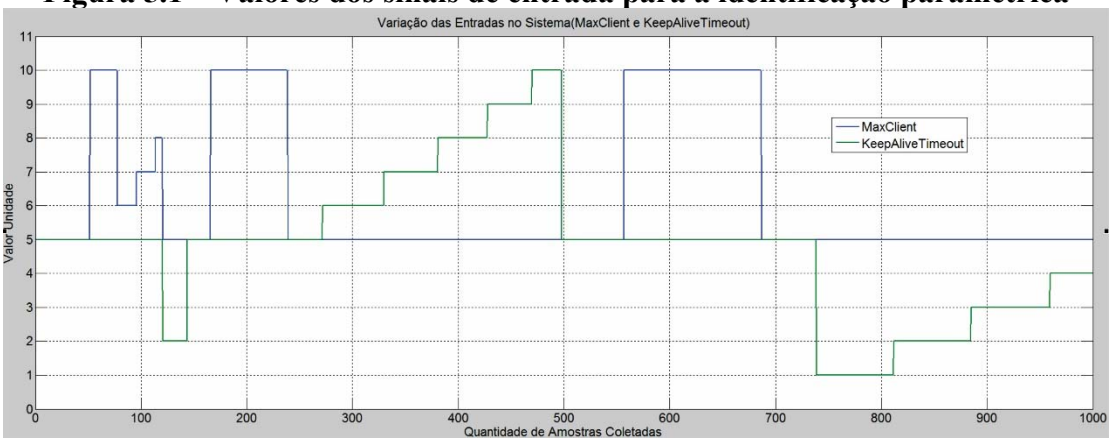
Será feita a coleta de dados relacionando a quantidade de memória utilizada com o número de processos "httpd" existentes no "background" do sistema operacional, ou seja, memória utilizada (retirando "buffer" e "cached") com o parâmetro "MaxCliente".

A variação no tempo é causada pela alteração de carga no servidor, sendo um fator externo ao sistema que provoca alterações na dinâmica que está regulando a evolução temporal do sistema. Caracteriza-se um sistema discreto invariante se a entrada u(k-ko) produzir uma saída y(k-ko), sendo os sinais de entrada e saída do sistema discreto u(k) e y(k). O modelo matemático a ser projetado para representar o sistema, inicialmente, será um modelo linear invariante no tempo baseado na simplicidade dos sistemas expostos [25].

A identificação paramétrica é uma boa metodologia a ser adotada para obter a função de transferência desses sistemas. Isto permite que se tenha uma primeira ideia das principais características dinâmicas do processo, como a presença de atraso puro, as constantes de tempo dominantes (que influenciam a escolha do intervalo de amostragem) e os ganhos estáticos.

Para obter dados de testes de funcionamento em malha aberta, foi utilizado o *software* projetado que varia os valores do sinal de entrada "MaxCliente" e "KeepAliveTimeout". Amostragens dos sinais são apresentadas na Figura 5.1.

Figura 5.1 – Valores dos sinais de entrada para a identificação paramétrica



A inserção do parâmetro "KeepAliveTimeout" ao *software* desenvolvido é de baixa complexidade, já que segue metodologia semelhante à do "MaxClient". O "KeepAliveTimeout" é o parâmetro que estipula quanto um processo pode ficar ocioso sem ser morto. Configuração muito elevada provoca consumo de memória e baixo valor provoca consumo do processador, ou seja, com valores altos, o processo fica em espera, consumindo a memória que está ocupando, e com valores altos, o processador precisa trabalhar mais, pois cria processo toda vez que chegar uma requisição, demandando um poder de processamento alto.

Junto do módulo de alteração do "KeepAliveTimeout" foi desenvolvido um módulo para extrair a porcentagem de utilização da CPU, seguindo os mesmos preceitos da quantidade de memória com alteração da comandando, que para este é o comando "top". Esse valor também é encaminhado ao processo "Tupla" para inserção no banco de dados, fazendo o relacionamento funcional entre os valores.

Assim, é possível extrair os valores do banco de dados e projetar estes valores em forma de gráficos. O sinal de quantidade de memória ocupada é visto na Figura 5.2 e o sinal da porcentagem de "UCP" (unidade central de processamento) é visto na Figura 5.3.

O módulo que relaciona o "KeepAliveTimeout" e a porcentagem de "UCP" já foi projetada para novos trabalhos. Neste, o foco é o relacionamento de quantidade de memória com o "MaxCliente".

Da análise dos gráficos, pode-se concluir que o sistema responde à excitação, em média, após a coleta da nonagésima amostra, ou seja, o tempo de resposta do sistema computacional

varia entre 4 a 100 segundos. Essa variação é decorrente do método utilizado de alteração do parâmetro.

Figura 5.2 – Quantidade de memória por unidade

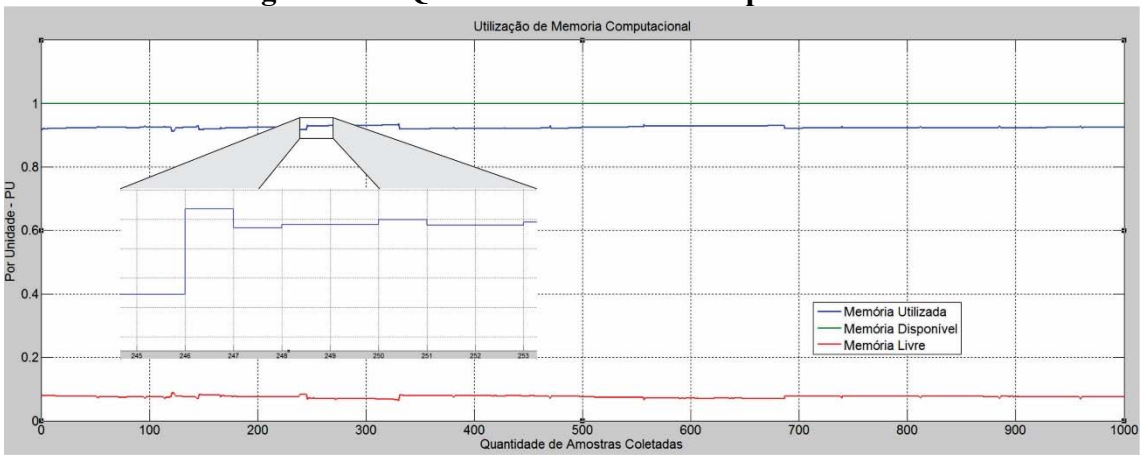
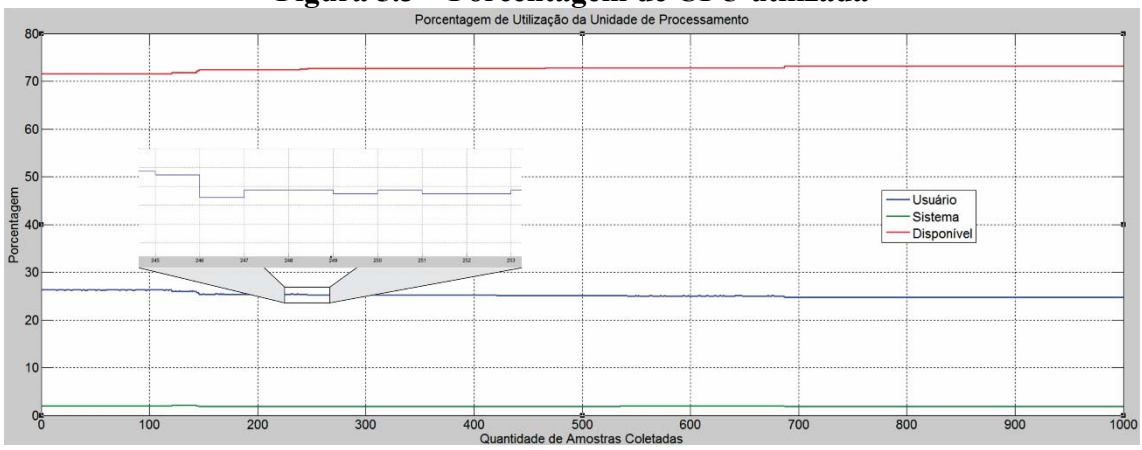


Figura 5.3 – Porcentagem de CPU utilizada

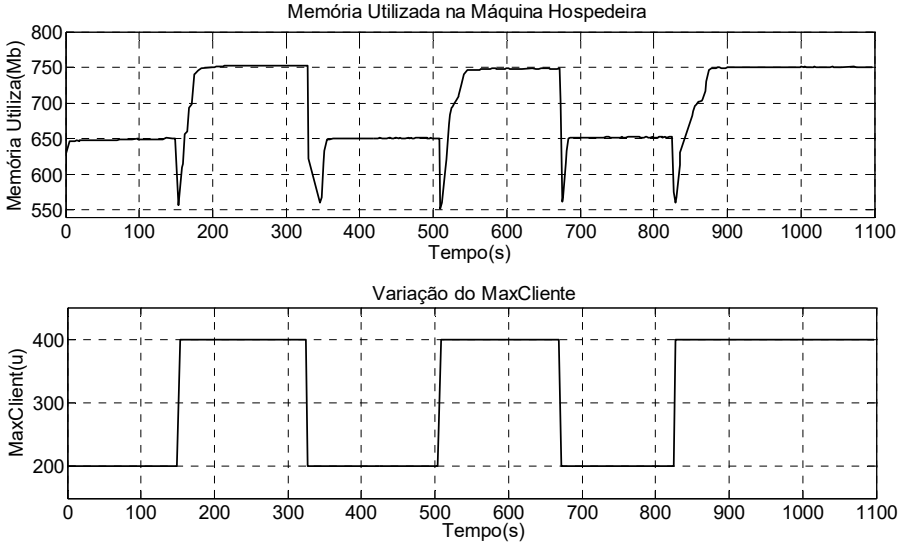


Em [8], faz-se referência à variação do tempo de resposta e justifica-se a variação no número, tipos ou composição das requisições que devem ser respondidas pelo sistema computacional. Entretanto, o principal componente de um sistema computacional para o tempo de resposta é o sistema operativo que está sustentando todas as aplicações hospedas na máquina. Neste trabalho foi utilizado, para as três máquinas virtuais, um sistema operativo que é comum quando se deseja ter o serviço de resposta "HTTP" baseado no "Apache", o "LINUX Ubuntu".

O "Ubuntu" não é um sistema operativo de tempo real. Assim, seu escalonador de processo não possui "deadline", realizando a multiplexação de processos sem ter como prioridade o tempo de conclusão, ocasionando a variação no tempo de assentamento do sistema computacional estudado.

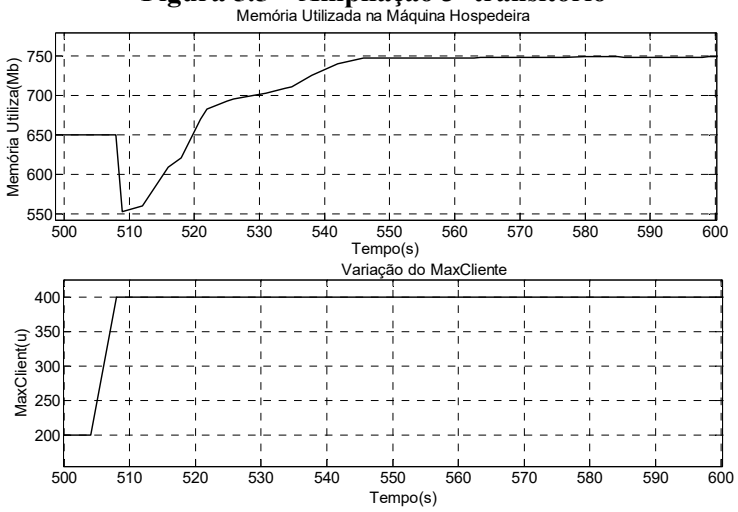
Devido à variação no tempo de resposta, foram realizados outros testes no sistema que confirmaram a variação (Figura 5.4). Ficando evidente que, conforme a máquina hospedeira possui outros processos em "background", o tempo de resposta irá aumentar. A diferença entre os testes realizados nas Figuras 5.1, 5.2 e 5.3 e nas Figuras 5.4, 5.5 e 5.6 foi a retirada do antivírus do "background" do sistema operativo.

Figura 5.4 – Variação e resposta do sistema "httpd"



Pode-se observar na Figura 5.4 que os maiores tempos para a acomodação do sistema são vistos nas terceira e quinta alterações do "MaxClient" (de 200 para 400 processos). A Figura 5.5 e a Figura 5.6 são as respectivas ampliações das alterações mencionadas para a retirada do maior tempo de acomodação do sistema.

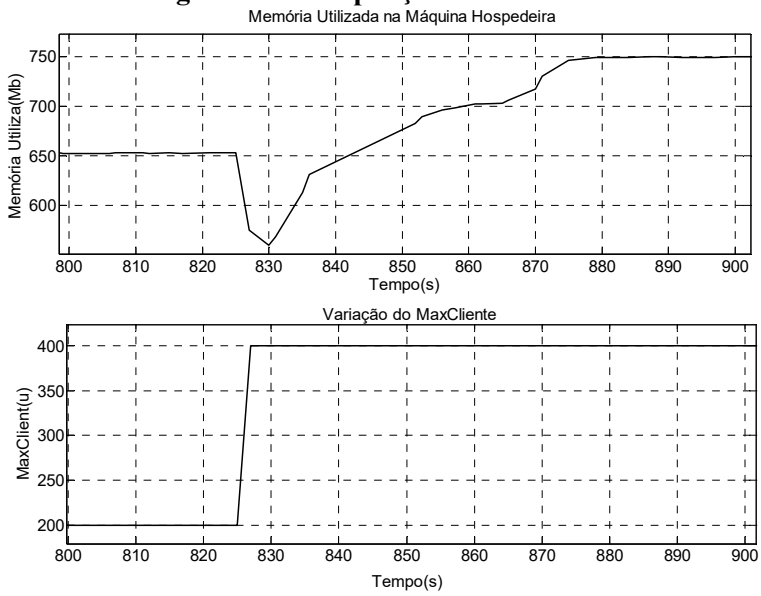
Figura 5.5 – Ampliação 3º transitório



Outro aspecto a ser mencionado é o declínio do valor de memória nos momentos iniciais da dinâmica do sistema. Em todas as alterações do "Maxclient", o declínio se faz presente,

induzindo ao entendimento de que este sistema é de fase não mínima. Entretanto, essa dinâmica recorrente se deve ao fato de o *software* matar todos os processos do "Apache" após a nova geração já estar pronta para o atendimento das requisições "HTTP". Esta forma de gerenciar os processos foi pensada com base no tipo de carga que está sendo gerada (vide final do capitulo). Outros métodos para a morte dos processos "httpd" da geração anterior podem ser estudas e implementados no *software*.

Figura 5.6 – Ampliação 5º transitório



Acredita-se que a variação do tempo de resposta é motivada pelo grande número de processos sendo executados em "background" nos sistemas operativos. É importante lembrar que o sistema computacional em foco não possui um *hardware* dedicado para cada processo, e para aumentar a complexidade e diminuir o "time-sharing" para os processos, existe a virtualização sendo utilizada para sustentar o sistema completo (serviço de resposta, armazenamento e gerador de carga).

A tecnologia de virtualização separa os recursos computacionais a serem utilizados por cada máquina virtual, não interferindo na execução de outro sistema operativo hospedado que utiliza a técnica de virtualização. Entretanto, existem componentes que não é possível fazer essa separação, e.g. o barramento. Consequentemente, haverá um momento na utilização desses recursos indivisíveis em que uma máquina virtual irá interferir na outra.

O tempo de 55 segundos é o maior tempo de acomodação do sistema que será adotado para projetar o sinal binário pseudoaleatório (SBPA). Para o menor tempo será adotado o de 24 segundos visando a garantir que o sistema responda às requisições "HTTP". Para o valor de

amplitude, será adotado o ponto de operação de 200 processos, variando em 200 processos para mais, uma vez que não houve impedimentos nos testes de variação do sistema em malha aberta.

Um sinal deve ser inserido na planta para estimular a dinâmica e a estática dominantes do sistema que será identificado. Esse sinal tem algumas características que, segundo [25], são:

> [...]tem potência espectral numa ampla faixa de frequência. O sistema a ser identificado terá os modos dominantes numa faixa finita de frequência. Do ponto de vista desse sistema, uma entrada será "branca" se ela contiver suficiente energia na faixa de frequências dominantes. Portanto, exigir que a entrada seja branca equivale a requerer que tal entrada excite a dinâmica do sistema.

De uma forma simples, as características dinâmicas e estáticas que não forem excitadas não aparecerão nos dados e o que não estiver nos dados não será identificado. Para obter os dados das características, aplica-se um sinal do tipo binário pseudoaleatório (SBPA) na planta.

Dessa forma, um sinal SBPA se faz necessário para excitar a dinâmica dominante da planta. Assim, neste trabalho, desenvolveu-se um módulo de alteração automática dos valores dos sinais que obedece às variações projetadas pelo SBPA. Geram-se, dessa forma, sinais que possuem semelhança de aleatoriedade e que a autocorrelação na variável de resíduo seja próxima de zero.

Para excitar a dinâmica da planta com a finalidade de estimar um modelo paramétrico, projetou-se um sinal do tipo sequência binária pseudoaleatória (SBPA). Para a escolha dos parâmetros foram utilizados os seguintes critérios [25]:

1. Para os valores de amplitude do sinal SBPA, limita-se pela máxima excursão permitida ao sinal de excitação do processo, não provocando interrupção no serviço de resposta "HTTP" e nem alteração na faixa linear de operação.
2. O período do sinal não deve ser menor do que o tempo de acomodação do sistema. Caso contrário, reduz o caráter de aleatoriedade do teste.
3. O intervalo entre os bits deve ficar próximo do menor tempo do sistema dividido por 3, já que o sistema é não linear.

Assim, o sinal projetado ficou com um tempo entre bits de 8 segundos e com um número de bits igual a 3, totalizando o tempo de periodicidade de 56 segundos (Figura 5.7). Com esses parâmetros de projeto do sinal binário pseudoaleatório, são obedecidos os critérios adotados em [25].

A Figura 5.8 mostra o sinal binário pseudoaleatório (características conservadas do sinal SBPA projetado) obtido do chaveamento do "MaxClient" entre dois valores. Inicialmente, a planta se encontrava em estado estacionário, "MaxClient" igual a 250, e, a partir do 5º segundo, o sinal SBPA é inserido.

Figura 5.7 - Sinal SBPA projetado

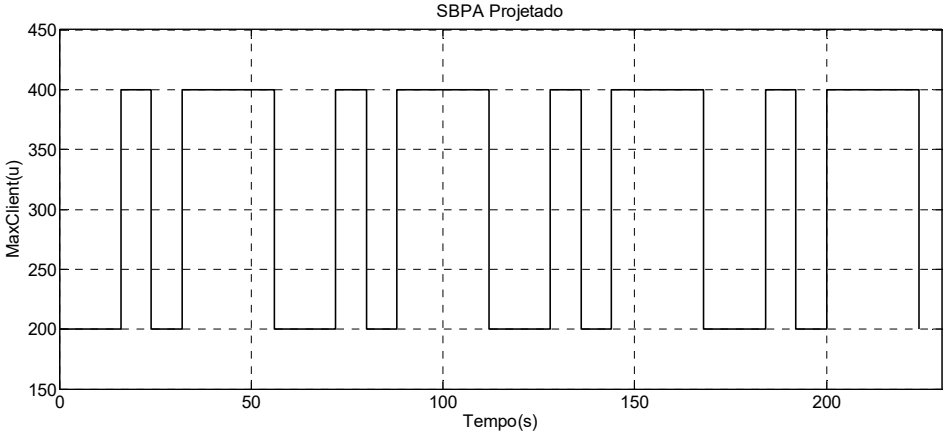
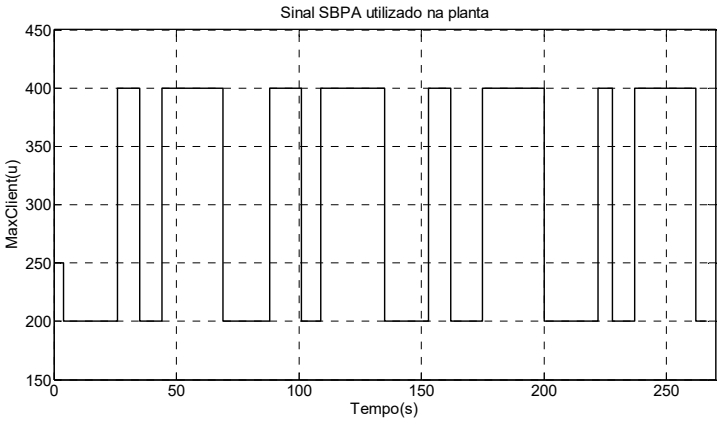


Figura 5.8 - Sinal SBPA projetado com estado inicial da planta

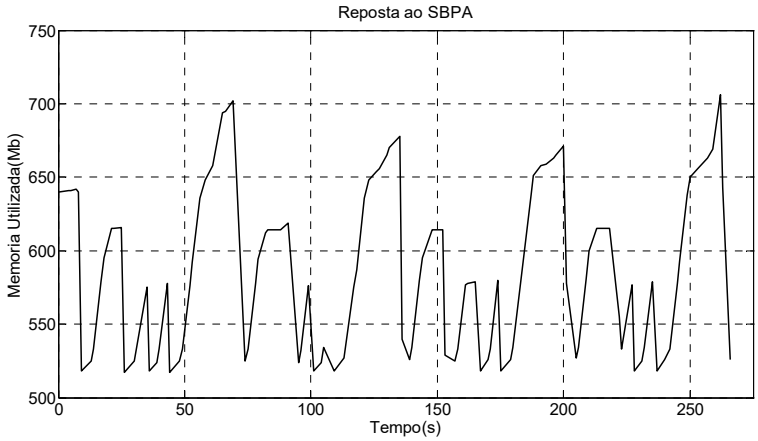


A resposta do sistema ao sinal SBPA é mostrada na Figura 5.9. Nota-se que, mesmo para os níveis mais largos do sinal SBPA, o sistema não permanece no estado estacionário, obedecendo à literatura[31]. O máximo de memória consumida foi de aproximadamente 700 MB e o mínimo foi de aproximadamente 524 MB. É importante sempre enfatizar que o valor de memória consumida não pertence somente aos processos clones criados pelo processo pai do sistema "Apache", mas de todos os outros processos que estão armazenados na memória principal, independentemente do estado dos processos (ativo, espera, bloqueado, zumbi etc.) ou forma de execução.

Se for realizada uma comparação com o teste anterior na quantidade de memória consumida, existirá uma diferença de aproximadamente de 50 MB. Não foi possível identificar o porquê dessa diferença, já que não foi realizado um monitoramento de todos os processos

armazenados na memória principal em execução e nem houve atividade de usuário de fechamento de *software* no momento do ensaio. A princípio, essa diferença foi atribuída ao sistema operacional por ter a capacidade de liberar memória se achar necessário sem a interferência de qualquer usuário.

Figura 5.9 – Resposta ao sinal SBPA projetado



Utilizando-se o "toolbox" de identificação do "MATLAB", chegou-se à equação 5.1, que possui o comportamento semelhante ao da planta em estudo. Foi utilizada a metodologia dos mínimos quadrados não-recursivos para otimizar a busca dos valores estimados dos parâmetros do modelo. Foi especificado um modelo de 1ª ordem que possui um polo, um zero e atraso discreto unitário. Obteve-se a seguinte função de transferência pulsada, tendo 1 segundo como tempo de amostragem:

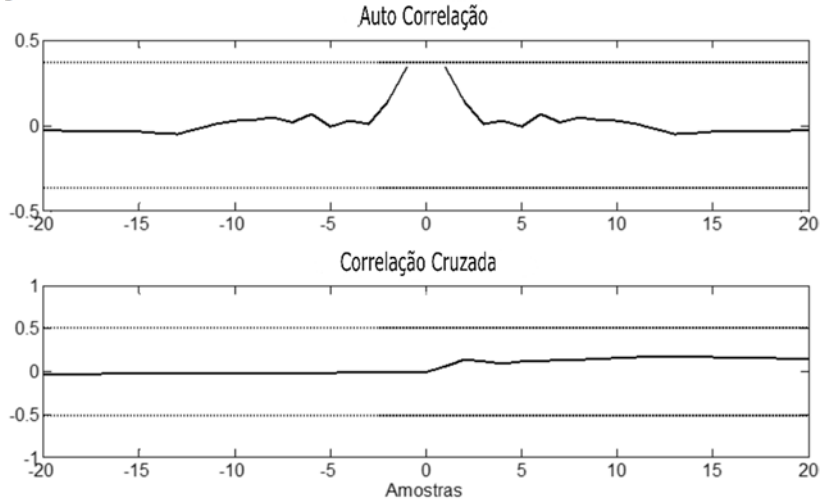
$$\boldsymbol{H(q^{-1}) = \frac{0,1521q^{-1}}{1 - 0,6732q^{-1}}} \qquad \textbf{5.1}$$

Uma das técnicas de validação dos dados de domínio do sistema é a função de auto correlação e a função de correlação cruzada, ou seja, para que o modelo do sistema seja válido, os resíduos estimados deverão ser aproximadamente não correlacionados. Foi utilizado o desvio padrão igual a 7 para o desenvolvimento da Figura 10, uma vez que a variação média de mais ou menos 7 não tem uma influência preocupante na planta.

O coeficiente de autocorrelação informa quanto uma determinada realização de uma variável aleatória influencia os seus vizinhos. Por exemplo, se a ocorrência de uma variável é considerada alta, o quanto isso influenciará em ter o próximo valor alto. Na Figura 5.10, observa-se que o valor residual de autocorrelação próximo ao instante 0 é aproximadamente um pulso unitário (não sendo representado no gráfico porque a sua amplitude modifica a

proporcionalidade da figura) e praticamente nulo para os demais atrasos. Pode-se concluir que o sinal de resíduo possui espectro de frequência quase branco.

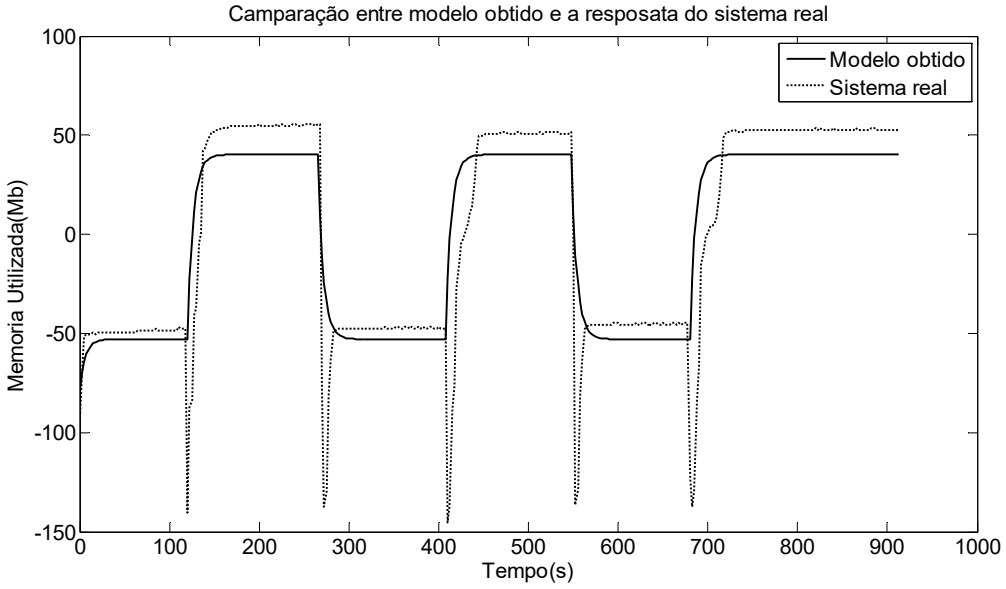
Figura 5.10 - Autocorrelação e coeficiente de correlação cruzada



O coeficiente de correlação cruzada é uma medida de similaridade entre dois sinais em função de um atraso aplicado a um deles. A Figura 10 mostra a correlação cruzada entre os resíduos estimados e a série temporal. Percebe-se que a linha do resíduo não passa da margem de segurança o que garante as características de espectro de frequência quase branco.

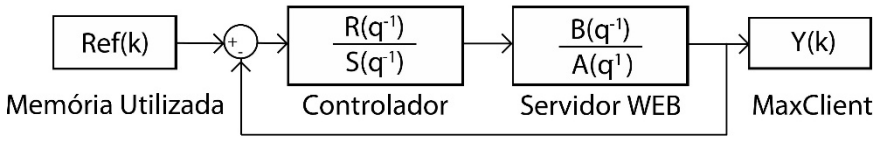
A Figura 5.11 mostra a comparação entre o modelo real e o modelo obtido. A percentagem de ajuste foi quantificada em 52,08%. Os dados de validação são da Figura 5.4, sendo que foram retiradas as suas médias.

Figura 5.11 - Comparação modelo obtido e sistema real



O controlador projetado para o sistema computacional "Apache" está baseado na teoria de controles PID, uma vez que já é possuída a equação que rege o sistema e também informações suficientes para o desenvolvimento de um controlador PI aplicando a metodologia de alocação de polos [30]. Foi escolhido um controlador do tipo PI devido às ações que as partes proporcional e integrativa fazem na planta. A parte proporcional foi utilizada para minimizar a ondulação do sistema e minimizar o erro. A parte integrativa suaviza o máximo sobressinal ("overshoot") e zera o erro. A parte derivativa não foi projetada devido às características de sua atuação na planta, que fornece uma correção antecipada do erro e é inversamente proporcional à velocidade de variação, indicando que não deve ser utilizada em sistema de resposta rápida a uma perturbação e que possua muito ruído no sinal medido[23]. A Figura 5.12 ilustra o diagrama de blocos do controlador digital implementado.

Figura 5.12 –Esquema do Controlador Retroalimentado



Sendo que [23]:

$$\frac{\boldsymbol{R(q^{-1})}}{\boldsymbol{S(q^{-1})}} = \frac{\boldsymbol{r_0 + r_1 q^{-1}}}{\boldsymbol{1 - q^{-1}}} \qquad \textbf{5.2}$$

Utilizando-se a técnica de posicionamento de polos, pode-se encontrar o valor de r_0 e r_1. Como o controlador é de 1º ordem e a planta é de 1º ordem, o polinômio da equação desejada $H_{des}(q^{-1})$ em malha fechada será de 2º ordem, tendo a seguinte forma:

$$\boldsymbol{A_{MF}^{des} = z^2 + a_1^{des} z + a_2^{des}} \qquad \textbf{5.3}$$

O polinômio de malha fechada é da forma:

$$\boldsymbol{A_{MF} = z^2 + (a_1 - 1 + b_0 r_0) z + (b_0 r_1 - a_1)} \qquad \textbf{5.4}$$

Igualando-se as equações 5.3 e 5.4 em forma matricial, temos:

$$\begin{bmatrix} \boldsymbol{b_0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{b_0} \end{bmatrix} \begin{bmatrix} \boldsymbol{r_0} \\ \boldsymbol{r_1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a_1^{des} + 1 - a_1} \\ \boldsymbol{a_2^{des} + a_1} \end{bmatrix} \qquad \textbf{5.5}$$

Para se obter o valor de a_1^{des} e a_2^{des}, utiliza-se o conhecimento da planta para impor o comportamento desejado. O tempo de assentamento (ts) é igual a 55s e, utilizando-o sobre um percentual de 5% ($\xi = 0{,}7$) , tem-se [25]:

$$\boldsymbol{ts = \frac{4}{\xi w_n} = \frac{4}{0,7 * 55} = w_n = 0,1039_{rad/s}} \qquad \textbf{5.6}$$

Entretanto, o sistema deste trabalho é discreto. Para transformar a_1^{des} e a_2^{des} no tempo discreto, utilizam-se as seguintes equações [22].

$$\boldsymbol{a_1^{des} = -2e^{-\xi\omega_n T_s} cos(\sqrt{1-\xi^2} w_n T_s)} \qquad \textbf{5.7}$$

e

$$\boldsymbol{a_2^{des} = e^{-2\omega_n T_s}} \qquad \textbf{5.8}$$

Aplicando-se os valores nas equações 5.7 e 5.8 com Ts = 8s, obtêm-se $a_1^{des} = -0{,}9266$ e $a_2^{des} = 0.1897$. Realizando-se as operações matemáticas na equação 5.5:

$$\boldsymbol{\begin{bmatrix} 0,1521 & 0 \\ 0 & 0,1521 \end{bmatrix} \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} -0,9266 + 1 - (-0,6732) \\ 0,1897 + (-0,6732) \end{bmatrix}} \qquad \textbf{5.9}$$

Assim, o valor de $r_0 = 4{,}9088$ e $r_1 = -3{,}1789$. A função de transferência do controlador PI sintonizado é vista na equação 5.10:

$$\boldsymbol{\frac{R(q^{-1})}{S(q^{-1})} = \frac{4,9088 - 3,1789q^{-1}}{1 - q^{-1}}} \qquad \textbf{5.10}$$

Elaborando-se o diagrama de blocos para a simulação do sistema em malha fechada, tem-se o esquema da Figura 5.13. A partir deste esquema, é possível extrair o sinal de erro, o sinal de controle e a resposta do sistema que são essenciais para se realizar uma análise do desempenho do controlador proporcional integrativo (PI) projetado.

Figura 5.13 – Esquema da Simulação do Sistema em Malha Fechada

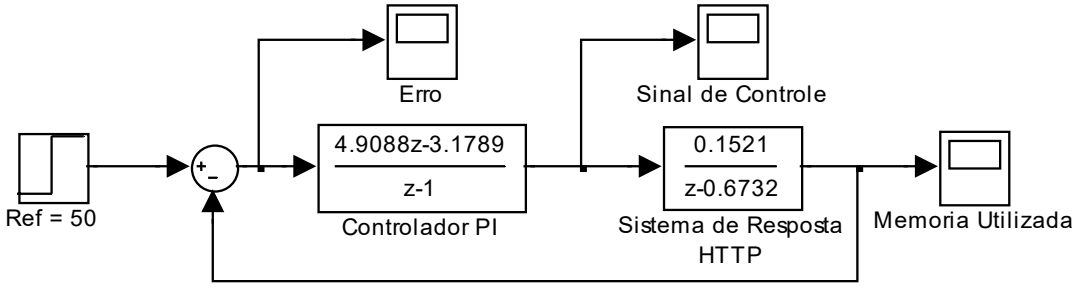
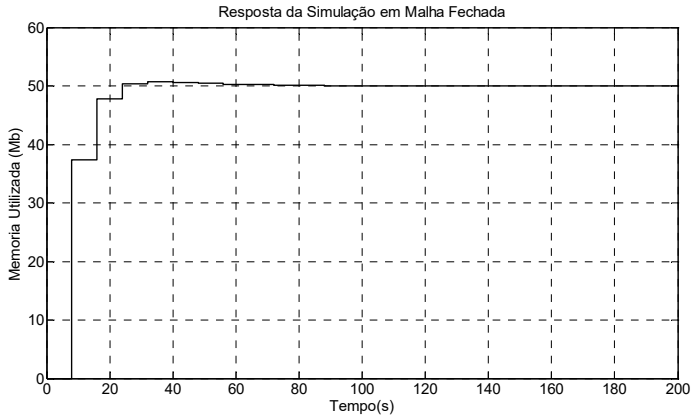
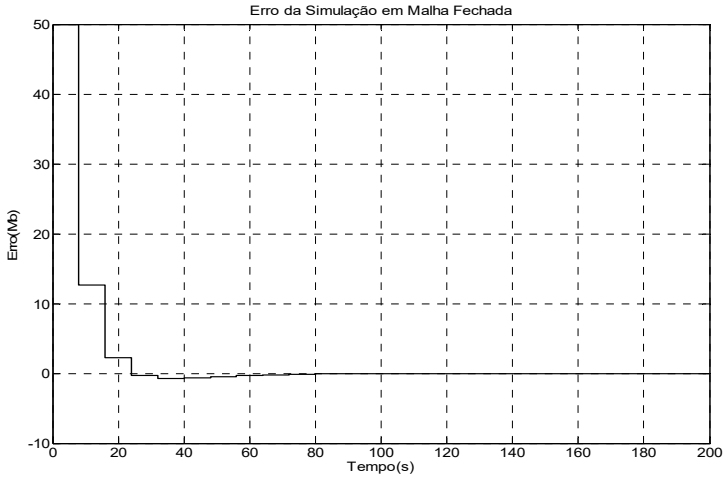


Figura 5.14 – Resposta da Simulação



A Figura 5.14 mostra a memória utilizada pelo sistema de resposta à requisição "HTTP". Observa-se que o tempo de assentamento (próximo ao valor de 55 segundos) está em conformidade com a planta e o máximo sobressinal ("overshoot") nos critérios estabelecidos (menor que 5% do valor de referência). Assim, o controlador projetado pode ser incluído na planta real, uma vez que houve a estabilização do sistema e o sobressinal máximo não é grande para provocar um "overload" no sistema em estudo. Esses dois aspectos são necessários para que o sistema de resposta às requisições "HTTP" não pare, seja por consumir toda a memória devido a um sobressinal máximo ou por excesso de processamento por uma instabilidade no sistema.

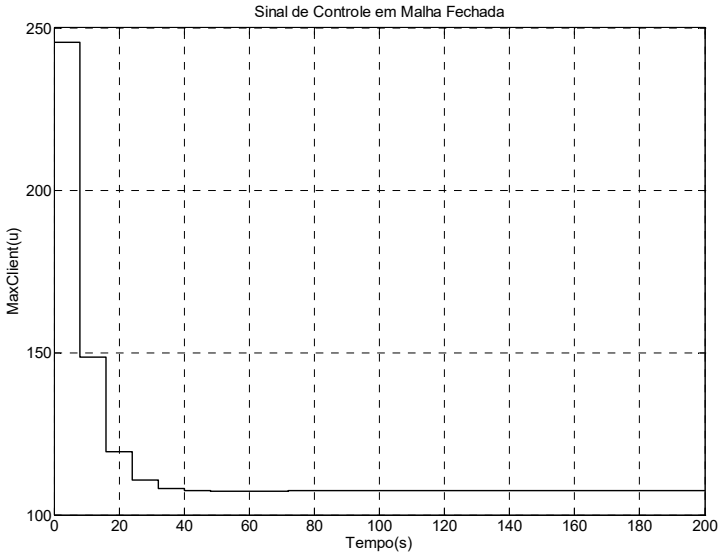
Figura 5.15 – Sinal de Erro da Simulação em Malha Fechada



A Figura 5.15 e 5.16 são, respectivamente, o gráfico do erro e do sinal de controle. Devido à utilização da parcela integrativa do controlador PID o erro do sistema foi anulado a partir do octogésimo segundo. Entretanto, no tempo de assentamento da planta, o erro é muito

próximo de zero, já não interferindo na planta devido a uma unidade de processo da variável "MaxClient" possuir um tamanho fixo, dependendo da página web sendo requerida. O sinal de controle não possui amplitude relativa alta, podendo-se afirmar que o esforço de controle não é elevado, sendo estabilizado antes do tempo de assentamento da planta.

Figura 5.16 – Sinal de controle da simulação em malha fechada

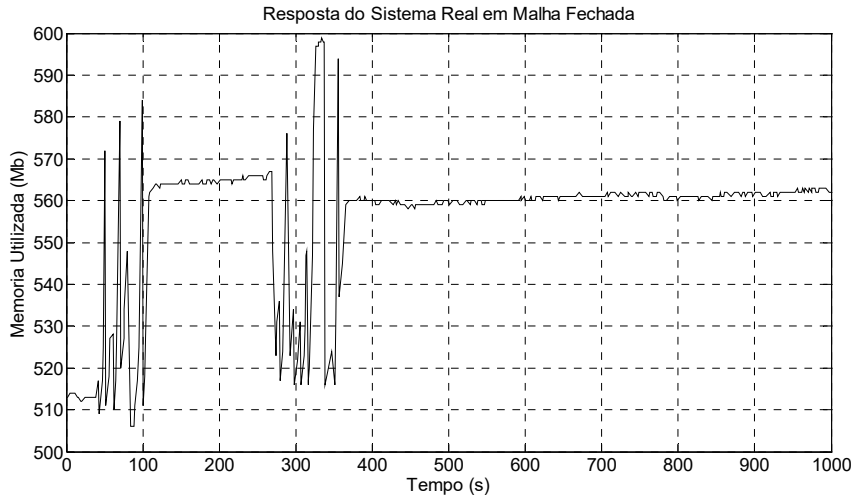


Assim, o controlador PI é suficiente para a planta nos pontos de operação do projeto do controlador e obedecendo a todos os requisitos do projeto. O próximo passo é inserir os parâmetros do controlador no *software* desenvolvido para verificar o comportamento do sistema computacional estudado.

Utilizando-se os valores projetados para o controlador PI no *software* desenvolvido, obteve-se a resposta do sistema (Figura 5.17) para uma referência de 562 megabytes de memória utilizada. Inicialmente, o servidor "HTTP" não está com carga. Desta forma, os processos existentes eram os 5 processos que são configurados no "StartServer" (intervalo de tempo de 0 a aproximadamente 50 segundos). A partir do instante 50 segundos, foi colocada carga no servidor "Apache" e iniciada a ação do controlador. Observa-se pela Figura 5.17 que o sistema teve sua estabilidade aproximadamente 55 segundos após o início da atuação do controlador e o erro é menor que 1% do valor de referência, podendo ser considerado nulo [25].

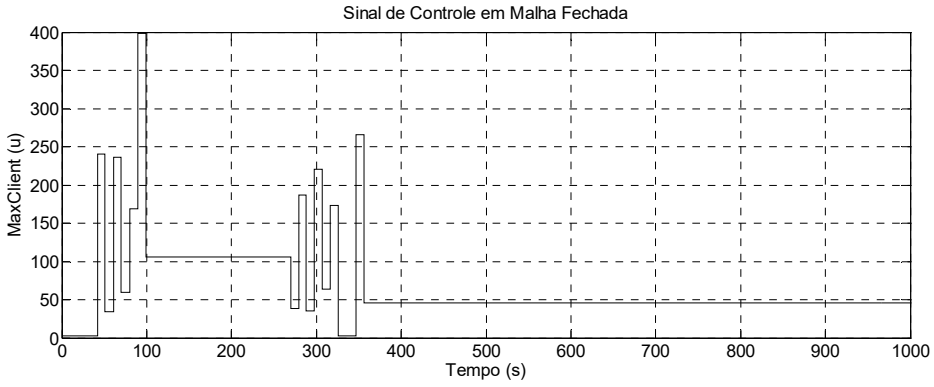
Aproximadamente no instante 270 segundos, foi provocada uma perturbação, a abertura de um *software* ("Gedit"). O controlador estabilizou o sistema aproximadamente 100 segundos após o início da perturbação. A demora na estabilização é ocasionada pelo tempo que os softwares levam para carregar por completo na memória ("Apache" e "Gedit"). Fato na simulação que considera o carregamento instantâneo. A estabilização tem um erro menor do que 1% do valor de referência, sendo considerado nulo.

Figura 5.17 – Resposta do sistema real em malha fechada



A Figura 5.18 mostra o sinal de controle que é utilizado na planta. No primeiro momento de atuação do controlador, período de 50-270 segundos, o valor de "MaxClient" foi estabilizado aproximadamente em 110 unidades de processo; no segundo momento de atuação do controlador, período de 270-1000 segundos, o valor de "MaxClient" foi estabilizado em aproximadamente 50 unidades de processo. A diminuição de processos para responder às requisições "HTTP" é ocasionada pela ocupação da memória por um software aberto, forçando o controlador a atuar de forma negativa no "MaxClient".

Figura 5.18 – Sinal de controle em malha fechada



A carga no servidor de resposta HTTP foi gerada a partir do *software httperf*, que é uma ferramenta para medir o desempenho do servidor WEB e foi desenvolvido pela empresa HP (parâmetros Tabela 5.1). A partir dos resultados encontrados, pode-se medir o tempo de resposta, as taxas de transferências, os níveis de utilização de recursos e o ponto de ruptura da aplicação que está implantada no servidor.

Tabela 5.1 – Parâmetros utilizados no httperf para a geração de carga no servidor "Apache"

Parâmetro	Descrição
Server	Especifica o endereço ou IP do servidor em que será feito o teste. Se essa opção não for especificada, o nome do host "localhost" é usado. Esta opção deve ser sempre especificada, já que não é uma boa ideia executar o cliente e o servidor na mesma máquina.
Rate	Especifica a taxa discreta na qual as conexões ou sessões são criadas. Caso não seja definida a taxa, as conexões ou sessões serão geradas assim que a anterior for concluída.
Hog	Esta opção permite ao *httperf* solicitar tantas portas TCP quanto precisar. Sem esta opção, *httperf* é geralmente limitado à utilização de portas efêmeras (na gama 1024-5000).
Num-conns	Específico para orientação de carga de trabalho. Ele especifica o número total de ligações para criar. Em cada ligação, as chamadas são emitidas conforme especificado pelas opções –num-calls e –burst-length. Um teste para assim que as conexões N concluírem ou falharem. Uma conexão falhará se nenhuma resposta for recebida depois de passado o tempo limite (soma de –timeout e –think-timeout). O valor padrão para esta opção é 1.
Num-calls	Específico para orientação de carga de trabalho. Corresponde ao número total de chamadas para a emissão em cada conexão. Se N for maior do que 1 (padrão do *httperf*), o servidor a ser testado deverá ter suporte a conexões persistentes.
Client	Especifica que a máquina *httperf* em execução é um cliente, com um total de N clientes, com N a variar normalmente de 0 a N-1. Este parâmetro também é usado para garantir que nem todos os clientes gerem cargas de trabalho perfeitamente idênticas.

Para este ensaio, foi utilizado o comando "/httperf --hog --server=172.16.15.15 --num-conns 720.000 --num-calls 100 --rate 600" provocando uma carga de 600 requisições por segundo, ou seja, 600 requisições em paralelo com 100 chamadas cada. As chamadas, "—num-call", servem para que os processos criados não fiquem ociosos, evitando a morte prematura deles pelo gerenciador de processos do "Apache". O número total de requisições enviadas será

720.000 requisições, e como são enviadas 600 requisições por segundo, o tempo em que o servidor "Apache" terá carga é de 1200 segundo.

6 – CONCLUSÃO

O propósito desta tese, de desenvolver um *software* não intrusivo ao servidor de requisições "HTTP", foi alcançado, possibilitando que o gestor de TI possa dimensionar a quantidade máxima de memória que o servidor poderá usar. Assim, um problema de falta de recurso, memória principal, para as rotinas básicas do sistema operacional, será automaticamente controlado, independentemente do conhecimento da pessoa responsável pelo sistema computacional, barateando os custos operacionais de pessoal.

O controlador relaciona a memória que está sendo utilizada por todo o sistema (sistema operacional, sistema "Apache" e outros programas ativados) com o número de requisições máximas que o "Apache" suporta simultaneamente. Desta forma, se houver a necessidade de executar um software ou módulo do sistema operacional que consuma uma determinada quantidade de memória e se o valor de referência (PU de memória utilizada) já tiver sido atingindo, o controlador irá reduzir o número máximo de requisições atendidas simultaneamente no "Apache", ou seja, o controlador perceberá qualquer perturbação na máquina hospedeira, realizando então a prevenção de "overload".

Entretanto, para sistemas que utilizam a memória virtualizada, o problema de falta de recursos de memória principal praticamente não existe, pois há uma extensão da memória principal para a memória secundária, não havendo limite para o uso de memória, com um aumento significativo no tempo de resposta por utilizar meio mecânico de armazenamento, disco rígido. Ao controlar o uso da memória, pode-se evitar o uso da memória virtual, proporcionando um controle do tempo de resposta do servidor.

Na identificação do sistema, nota-se que, dependendo do estado do sistema operacional, há uma variação no tempo de assentamento da planta. Essa variação no tempo é resultante da quantidade de recursos computacionais ocupados por processos que estão na fila do sistema operacional que hospeda o servidor de resposta à requisição "HTTP".

A fase não mínima do sistema é ocasionada pela execução do gerenciamento dos processos "httpd" da geração n-1 em forma de "batch" (processamento de dados em lote). Os processos da geração n são criados de acordo com o número de requisições (carga) na fila da porta "TCP" 80 do sistema operacional. Então, há uma liberação íngreme e uma ocupação, que demanda tempo, na memória.

Por se tratar de um sistema não linear, o projeto de controlador PI obteve êxito no ponto de operação no qual houve o ensaio do controlador, satisfazendo o principal objetivo de

permanecer a quantidade de memória ocupada, ponto de referência, com a introdução de perturbações de cargas e alteração no ambiente computacional como abertura de *software*.

Para o ponto de operação, são satisfatórios os resultados obtidos, já que a saída da planta não ultrapassa 1% do valor de referência. Entretanto, um refinamento no estado transitório deve ser estudado para evitar o aumento da percentagem de utilização da UCP, provocadas com as excessivas trocas de configuração do "Apache".

Esse refinamento pode ser feito com um projeto de controlador intervalar, minimizando os resultados provenientes da não linearização do sistema para outros pontos de operação, ou através de estratégias de controle multivariável para atacar a problemática da variação de tempo de assentamento. Outro ponto também para as próximas pesquisas é a influência da virtualização nos resultados obtidos e como outros serviços hospedados na máquina hospedeira influenciam o comportamento do controlador PI projetado.

7 – REFERÊNCIA BIBLIOGRÁFICA

[1] Kjaer,M. A., Artist, *Disturbance Rejection and Control in Web Servers.* [Art]. Lund University, 2009.

[2] Coar, K & Bowen, R. (2009). Apache Guia Prático. Alta Books, Rio de Janeiro.

[3] Abreu,T. W. M. & Barra, W., "Estratégias de Identificação Paramétrica Aplicadas à Modelagem Dinâmica de um servidor WEB Apache," *Revista Controle & Automação,* pp. 38-48, Janeiro e Fevereiro 2012.

[4]C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, S. Hyuk. Son. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers, IEEE Transactions on Parallel and Distributed Systems, pp 1014-1026, Vol. 17, NO. 9, September 2006.

[5] I. K. Gupta, J. Kumar, P. Rai. Optimization to Quality-of-Service-driven Web Service Composition using modified Genetic Algorithm, IEEE International Conference on Computer, communication and Control, IC4, 2015.

[6] R. Digital, "OlharDigital," Uol, 07 02 2013. [Online]. Available: httpd://olhardigital.uol.com.br/noticia/os-20-sites-mais-acessados-do-mundo/32477. [Acesso em 18 12 2013].

[7] Barreto, M. V., Bezerra Filho, F. F., Barra Junior, W., & Nogueira, F. G. (23 de outubro de 2015). Controle por realimentação para melhoria do desempenho dinâmico de requisições HTTP em máquinas virtuias. *Simposio Brasileio de Automação Inteligente – SBAI*

[8] Hellerstein, J. L., Diao, Y., Parekh, S., & Tilbury, D. M. (2004). Feedback Control of Computing Systems. New Jersey: John Wiley & Sons.

[9]Tanenbaum, A. S. (2000). Organização Estruturada de Computadores. Rio de Janeiro: LTC - Livros Técnicos e Ciêntificos Editora S.A.

[10] Ohlin, M., & Kjaer, M. A. (2007). Nice Resource Reservations in Linux. Second IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (pp. 1-7). Minich: IEEE.

[11]Ford, A(2009).Apache 2: Guia de Bolso. Alta Books, Rio de Janeiro.

[12]Gandhi, N., Tilbury, D. M., Diao, Y., Hellerstein, J., & Parekh, S. (2002). MIMO Control of an Apache Web Server: Modeling and Controller Design. Proceedings of the American Control Conference.

[13] Diao, Y., Gandhi, N., Hellerstein, J. L., Parekh, S., & Tilbury, D. M. (2008). Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Server. IEEE Xplore.

[14]Lindegren, E. Preparing the Apache HTTP Server for Feedback Control Application. Master Thesis. Lund University - Department of Automatic Control, Lund.

[15]Z. Zhu, X. GAO, C. Zhang. Research of an adaptive PI admission control of Web service. Chinese Control and Decision Conference(CCDC), IEEE, pp 2290 – 2293, 2014.

[16] Tanenbaum, S. Andrew (1999). Sistemas Operacionais Modernos. LTC – Livros Técnicos e Científicos Editora S.A. , Rio de Janeiro.

[17] Marques, J. A. , Ferreira, P. , Ribeiro, C. , Veiga, L. & Rodrigues, R. (2011). Sistemas Operacionais. LTC – Livros Técnicos e Científicos Editora LTDA, Rio de Janeiro.

[18] Murhammer, M. W., Atakan, O., Bretz, S., Pugh, L. R., Suzuki, K., & Wood, D. H. (2000). TCP/IP Tutorial e Técnico. São Paulo: Makron Books.

[19] Apache.org. (s.d.). *Apache Core Features*. Acesso em 31 de 01 de 2014, disponível em www.apache.org: http://httpd.apache.org/docs/2.2/mod/core.html#timeout

[20] N. H. Agnew, Transition metal complexes of poly(vinylpyridines). Journal of Polymer science, PartA, Polymer Chemistry, november, 1976

[21] BATES, D. M.; WATTS, D. G. Nonlinear regression: iterative estimation and linear approximations. [S.l.]: Wiley Online Library, 1988.

[22] Katsuhiko Ogata. Engenharia de Controle Moderno, 3º edição. Rio de Janeiro: LTC.

[23] Norman S. Nise. Engenharia de Sistemas de Controle, 5º edição. Rio de Janeiro: LTC

[24] Clainto M. Franchi. Controle de Processos Industriais, 1º edição. São Paulo: Érica Ltda.

[25] Luis A. Aguirre. Introdução à Identificação de Sistemas Técnicas Lineares e Não-Lineares Aplicadas a Sistemas Reais, 3º edição. Belo Horizonte: UFMG.

[26] Keel, L.H.; Bhattacharyya , S.P. . Robust stability and performance with "xed-order controllers. Automatica 35 (1999) 1717 -1724.

[27] Batista, B., Eutáquio, P., Leite, D., Estrella, j., Bruschi, S., Santana, M., et al. (2013). Avaliação de desempenho da migração de máquinas virtuais considerando diferentes configurações de segurança e de recursos computacionais em um ambiente de nuvem. SBRC Anais, pp. 15-25.

[28] Yourdon, E. (1990). Analise Estruturada Moderna. Rio de Janeiro : Campus.

[29] Portella, C.R; Vasconcelos, Á. B. (s.d). Redução do Consumo de Energia Utilizando Virtualização. Uniritter

[30] Astrom, K. j., & Wittenmark, B. (2014). *Computer - Controlled Systems.* New Jeraey: Prentice-Hall.

[31] Landau, I.D; Zito, G. (2006). Digital Control Systems; Design, Identification and Implementation, Springer

ANEXO I

CÓDIGO FONTE DO SOFTWARE PROPOSTO

```
/*
TITULO: Gerencimento do Servidor  Apache
DATA: 11/setembro/2013

Gerenciar as variaveis de controle do servidor apache que utiliza prefor

Autor: Marcos Vinicius Sadala Barreto

Funciona em LINUX!

Trabalho de Tese de Doutorado
Programa de Pós-Graduação em engenharia Eletrica - PPGEE
Universidade Federal do Pará - UFPA
Orientador: Walter Barra Junior


Linha de comando de copilação: gcc -o LucasV01 gerenciamento_apache.c -lmysqlclient -lm
IP de conecção com o banco: 172.16.15.15
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql/mysql.h>
#include <ctype.h>
#include <signal.h>
#include <math.h>
#include <unistd.h>



#define HOST "172.16.15.53"
#define USER "root"
#define PASS "Mal08092006"
#define DB "Status_Apache"
#define Config_Apache "//usr/local/httpd-2.4.6/conf/extra/httpd-mpm.conf"
#define Trans_Config_Apache "//usr/local/httpd-2.4.6/conf/extra/novo_httpd-mpm.conf"
#define ARQ_PRBS "//usr/local/httpd-2.4.6/htdocs/Lucas/prbs.dat"

// Variáveis Globais

FILE *ORIGINAL_MPM, *NOVO_MPM, *PRBS;
char linha[100];
char *result;
int StartServers, MinSpareServers, MaxSpareServers, MaxRequestWorkers,MaxConnectionsPerChild,
ServerLimit, KeepAliveTimeout;

//Contantes do Filtro Projetado
float Fa1=-0.9048;
float Fb0=0.0476;
float Fb1=0.0476;

int  Uk, Uk_1;
```

```
float Ek, Ek_1, Ref, Yk, R0, R1;


int flag=100;
int i;
int NAmostras[2];
int op, op1;
int amostra;

long int MemóriaTotal, MemóriaUsada, MemóriaLivre, MemóriaUsadaFiltrada;
float CPUUsuario, CPUSistema, CPULivre, CPUUsuarioFiltrada;
char *frase = "";

char retorno[100000]=""; // retorno do comando system
MYSQL  conexao; // variavel de conexão com mysql

pid_t filho_dados;
pid_t filho_carga;

// Executando o comando solicitado no sistema do servidor




void alterar_apache()

{
ORIGINAL_MPM = fopen(Config_Apache, "r");
NOVO_MPM=fopen(Trans_Config_Apache,"w");

system("clear");

if ((ORIGINAL_MPM==NULL)||(NOVO_MPM==NULL))
{
        printf("Problemas na abertura do arquivo\n");
        return;
}


while(!feof(ORIGINAL_MPM))
        {

        result=fgets(linha, 100, ORIGINAL_MPM);

  if    (!feof(ORIGINAL_MPM))
    {
          fputs(linha, NOVO_MPM );
      }

        if (strstr(linha,"<IfModule mpm_prefork_module>"))
                {
    fprintf(NOVO_MPM, "    StartServers            %d\n",StartServers);
    result=fgets(linha, 100, ORIGINAL_MPM);
    fprintf(NOVO_MPM, "    MinSpareServers         %d\n",MinSpareServers);
                        result=fgets(linha, 100, ORIGINAL_MPM);
    fprintf(NOVO_MPM, "    MaxSpareServers         %d\n",MaxSpareServers);
                        result=fgets(linha, 100, ORIGINAL_MPM);
    fprintf(NOVO_MPM, "    MaxRequestWorkers       %d\n",MaxRequestWorkers);
                        result=fgets(linha, 100, ORIGINAL_MPM);
    fprintf(NOVO_MPM, "    MaxConnectionsPerChild   %d\n",MaxConnectionsPerChild);
```

```
        result=fgets(linha, 100, ORIGINAL_MPM);
        fprintf(NOVO_MPM, "    ServerLimit           %d\n",ServerLimit);
        result=fgets(linha, 100, ORIGINAL_MPM);
        fprintf(NOVO_MPM, "    KeepAliveTimeout      %d\n",KeepAliveTimeout);
        result=fgets(linha, 100, ORIGINAL_MPM);
            }
            }
 fclose(ORIGINAL_MPM);
 fclose(NOVO_MPM);

 if (remove(Config_Apache))
     printf("Erro ao excluir arquivo antigo!\n");

 if(rename(Trans_Config_Apache,Config_Apache))
     printf("Erro ao renomear arquivo!\n");

system("//usr/local/httpd-2.4.6/bin/./apachectl -k graceful");
}



void exec_comando(char comando[50])
{
   int p[2], backup_saida_tela, backup_entrada_teclado;
   char buf[1000];


/* Descritor 1 é a SAIDA PADRAO, ou seja, por definicao a tela!
   Como vou mexer nele, estou salvando ele em backup_saida_tela
   Descritor 0 é a entrada padrão, ou seja, por definição o teclado
   Como vou mexer nele, estou salvando ele em backup_entrada_teclado
*/
    backup_saida_tela = dup(1);
    backup_entrada_teclado= dup(0);

/* Fecha a entrada padrao (0 = teclado) e a saida padrao (1 = tela) */
    close(0);
    close(1);

/* Pipe cria um canal bidirecional de comunicacao, sujos descritores
   serao armazenados no vetor p.

   NOTA: como 0 e 1 foram fechados, eles ocuparao a entrada e saida padrao
   respectivamente!

   Logo, ler do teclado significara ler do pipe, escrever na tela significara
   escrever no pipe
*/
    pipe(p);

/* Com a saida REDIRECIONADA, o system executado. Mensagens normais serao
   escritas no descritor 1 (saida padrao, que agora eh o pipe). Mensagens
   de erro continuam sendo enviadas para o 2 e sairao na tela imediatamente

   No caso a saida do comando (free exemplo) foi escrita NO PIPE!!!
*/
    system(comando);

/* Restaura a saida padrao, para que volte a ser a tela (salva logo no inicio
   do codigo)
```

```
*/
   dup2(backup_saida_tela, 1);

/* Veja, a ENTRADA PADRAO, 0, ainda continua sendo o pipe onde o system
  escreveu. Logo, ler da entrada padrao sera ler do pipe :-D
*/

   while (fgets(buf, 1000, stdin))
   {
          /* Lendo o que tem no pipe e ja jogando na tela. Mas agora buf pode
            ser processado de outra forma, como nao ser enviado para tela caso
            nao seja do interesse
           */
          strcat(retorno, buf);
           }
/*Restaura a entrada padrao, para que volte a ser o teclado (salva logo no inicio
  do codigo)
*/

dup2(backup_entrada_teclado,0);

}


//Extrai do retorno a quantidade de memória - total, Usada e livre o camando free

void quantidade_memória_servidor(char *Texto, long int *Vt,long int *Vu,long int *Vl)
{
int i=0;
char *t;
char TextoMemória[15];


    t = strchr(Texto,':');
    ++t;
//preparação da memória total
    i=0;
    while(*t==32)//testa espaço de separação dos valores do comando free
    {
      ++t;
    }

    while(*t!=32)
    {
       TextoMemória[i]=*t;
       ++i;
       ++t;
    }
    *Vt=atol(TextoMemória);
//Preparação a memória usada
    i=0;
    while(*t==32)//testa espaço de separação dos valores do comando free
    {
      ++t;
    }

    while(*t!=32)
    {
       TextoMemória[i]=*t;
       ++i;
```

```
        ++t;
    }
    *Vu=atol(TextoMemória);

//Preparação memória livre

    i=0;
    while(*t==32)//testa espaço de separação dos valores do comando free
    {
      ++t;
    }

    while(*t!=32)
    {
       TextoMemória[i]=*t;
       ++i;
       ++t;
    }
    *Vl=atol(TextoMemória);
}

void porcentagem_CPU(char *Texto, float *CPUU,float *CPUS,float *CPUF)
{

int i=0;
char *t;
char TextoMemória[15];
t = strchr(Texto,'%');
++t;
for(i=1; i<=7; i++)
{
++t;
}

while(*t==32)//testa espaço de separação dos valores do comando free
{
++t;
}

i=0;
while(*t!=32)
{
TextoMemória[i]=*t;
++i;
++t;
}
TextoMemória[i+1]=0;
*CPUU=atof(TextoMemória);
while(*t!=44)
{
++t;
}
++t;
while(*t==32)//testa espaço de separação dos valores do comando free
{
++t;
}
i=0;
while(*t!=32)
{
```

```
TextoMemória[i]=*t;
++i;
++t;
}
TextoMemória[i+1]=0;
*CPUS=atof(TextoMemória);

++t;
while(*t!=44)
{
++t;
}
++t;

while(*t!=44)
{
++t;
}
++t;
while(*t==32)//testa espaço de separação dos valores do comando free
{
++t;
}

i=0;
while(*t!=32)
{
TextoMemória[i]=*t;
++i;
++t;
}
TextoMemória[i+1]=0;
*CPUF=atof(TextoMemória);

}

void Conecta_Banco_Dados()
{
mysql_init(&conexao);
if ( mysql_real_connect(&conexao, HOST, USER, PASS, DB,0, NULL, 0) )
      {
          printf("Conectado com Sucesso!!!\n");
       }
       else
       {
          printf("Falha de conexao\n");
          printf("Erro %d : %s\n", mysql_errno(&conexao), mysql_error(&conexao));
       }
}


void Insercao_Banco_dados(long int MT, long int MU, long int ML, float CU, float CS, float CL, long int MUF,
float CUF)
{
char comando[100];
char variavel[20];

comando[0]=0;
strcat(comando, "CALL STATUS (");
sprintf(variavel, "%lu",MT);
```

```
strcat(comando, variavel);
strcat(comando, ",");


sprintf(variavel, "%lu",MU);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%lu",ML);
strcat(comando, variavel);
strcat(comando, ",");


sprintf(variavel, "%f",CU);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%f",CS);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%f",CL);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",StartServers);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",MinSpareServers);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",MaxSpareServers);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",MaxRequestWorkers);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",MaxConnectionsPerChild);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",ServerLimit);
strcat(comando, variavel);
strcat(comando, ",");

sprintf(variavel, "%d",KeepAliveTimeout);
strcat(comando, variavel);
strcat(comando, ",");


sprintf(variavel, "%lu",MUF);
strcat(comando, variavel);
strcat(comando, ",");


sprintf(variavel, "%f",CUF);
```

```
strcat(comando, variavel);
strcat(comando, ")");


//printf("\n");
//printf("%s\n", comando);

mysql_query(&conexao,comando);

}

void  Preparando_Filtro()  //motivo da utilização de arquivo e por estar utilizado multiprocessamento função fork()
{
//Preparação do arquivo para receber o valor de memória anterior
FILE *valor_anterior_memória;
valor_anterior_memória=fopen("penultimoValorMemória.txt", "w");
fprintf(valor_anterior_memória,"%lu",MemóriaUsada);
fclose(valor_anterior_memória);


//Preparação do arquivo para receber o valor de memória filtrada anterior

FILE *valor_anterior_memória_filtro;
valor_anterior_memória_filtro=fopen("penultimoValorMemóriaFiltrado.txt", "w");
fprintf(valor_anterior_memória_filtro,"0");
fclose(valor_anterior_memória_filtro);

//Preparação do arquivo para receber o valor de CPU anterior
FILE *valor_anterior_CPU;
valor_anterior_CPU=fopen("penultimoValorCPU.txt", "w");
fprintf(valor_anterior_CPU,"%f", CPUUsuario);
fclose(valor_anterior_CPU);


//Preparação do arquivo para receber o valor de CPU filtrada
FILE *valor_anterior_CPU_filtrada;
valor_anterior_CPU_filtrada=fopen("penultimoValorCPUFiltrada.txt", "w");
fprintf(valor_anterior_CPU_filtrada,"0");
fclose(valor_anterior_CPU_filtrada);

}
void filtro ()
{
long int AM,AMF;
float ACPU, ACPUf;


FILE *valor_anterior_memória;
FILE *valor_anterior_memória_filtro;
FILE *valor_anterior_CPU;
FILE *valor_anterior_CPU_filtrada;
valor_anterior_memória=fopen("penultimoValorMemória.txt", "r+");
valor_anterior_memória_filtro=fopen("penultimoValorMemóriaFiltrado.txt", "r+");
valor_anterior_CPU=fopen("penultimoValorCPU.txt", "r+");
valor_anterior_CPU_filtrada=fopen("penultimoValorCPUFiltrada.txt", "r+");

fscanf(valor_anterior_memória, "%lu", &AM );
fscanf(valor_anterior_memória_filtro, "%lu", &AMF );
fscanf(valor_anterior_CPU, "%f", &ACPU);
```

```
fscanf(valor_anterior_CPU_filtrada, "%f", &ACPUf );


//Filtro na Memória Usada

MemóriaUsadaFiltrada = (-Fa1*AMF)+(Fb0*MemóriaUsada)+(Fb1*AM);
AM=MemóriaUsada;
AMF=MemóriaUsadaFiltrada;


//Filtro na CPU Usuario

CPUUsuarioFiltrada= (-Fa1*ACPUf)+(Fb0*CPUUsuario)+(Fb1*ACPU);
ACPU= CPUUsuario;
ACPUf=CPUUsuarioFiltrada;



rewind(valor_anterior_memória);
rewind(valor_anterior_memória_filtro);
rewind(valor_anterior_CPU);
rewind(valor_anterior_CPU_filtrada);

fprintf(valor_anterior_memória,"%lu", AM);
fprintf(valor_anterior_memória_filtro,"%lu", AMF);
fprintf(valor_anterior_CPU,"%f", ACPU);
fprintf(valor_anterior_CPU_filtrada,"%f", ACPUf);


fclose(valor_anterior_memória);
fclose(valor_anterior_memória_filtro);

fclose(valor_anterior_CPU);
fclose(valor_anterior_CPU_filtrada);

}

void gravar_dados()
{

retorno[0]=0;
frase ="free";
exec_comando(frase);
quantidade_memória_servidor(retorno, &MemóriaTotal, &MemóriaUsada, &MemóriaLivre);

retorno[0]=0;
frase ="top -b -n 1";
exec_comando(frase);
porcentagem_CPU(retorno, &CPUUsuario, &CPUSistema, &CPULivre);

filtro();


Insercao_Banco_dados(MemóriaTotal, MemóriaUsada, MemóriaLivre, CPUUsuario, CPUSistema, CPULivre,
MemóriaUsadaFiltrada, CPUUsuarioFiltrada);
}

void inicializar_gerenciamento()
{
StartServers = 5;
```

```
MinSpareServers = 5;
MaxSpareServers = 10;
MaxRequestWorkers = 250;
MaxConnectionsPerChild =0;
ServerLimit = 256;
KeepAliveTimeout = 5;

alterar_apache();

MemóriaTotal = 0;
MemóriaUsada = 0;
MemóriaLivre = 0;
CPUUsuario = 0;
CPUSistema = 0;
CPULivre = 0;

Conecta_Banco_Dados();

}

void inserir_parametros_controle()
{
        system("clear");
  printf("**********************************Sistema de Gerenciamento do
Apache*********************************\n");
  printf("*                                                                    *\n");
  printf("* Modulo de Inserção de Parametros                                   *\n");
  printf("*                                                                    *\n");

printf("*************************************************************************************
***************\n");
  printf("\n");
  printf("*   StartServers:  "); scanf("%d",&StartServers);
  printf("*   MinSpareServers:  "); scanf("%d",&MinSpareServers);
  printf("*   MaxSpareServers:  "); scanf("%d",&MaxSpareServers);
  printf("*   MaxRequestWorkers:  "); scanf("%d",&MaxRequestWorkers);
  printf("*   MaxConnectionsPerChild:  "); scanf("%d",&MaxConnectionsPerChild);
  printf("*   ServerLimit:  "); scanf("%d",&ServerLimit);
  printf("*   KeepAliveTimeout:  "); scanf("%d",&KeepAliveTimeout);
  printf("\n");
  printf("                        Valores de Parametros Alterados!!!                        \n");
  sleep(2);
}


void pesquisa_parametros_atuais()
{
        system("clear");
  printf("**********************************Sistema de Gerenciamento do
Apache*********************************\n");
  printf("*                                                                    *\n");
  printf("* Modulo dos Valores Atuais do Sistema                                *\n");
  printf("*                                                                    *\n");

printf("*************************************************************************************
***************\n");
  printf("\n");
  printf("*   StartServers:%d\n", StartServers);
  printf("*   MinSpareServers:%d\n", MinSpareServers);
  printf("*   MaxSpareServers:%d\n", MaxSpareServers);
```

```
   printf("*   MaxRequestWorkers:%d\n", MaxRequestWorkers);
   printf("*   MaxConnectionsPerChild:%d\n", MaxConnectionsPerChild);
   printf("*   ServerLimit:%d\n", ServerLimit);
   printf("*   KeepAliveTimeout:%d\n", KeepAliveTimeout);
   printf("\n");
   printf("                    Aperte Qualquer Tecla Para Continuar!!!                    \n");
   sleep(4);
}


void iniciar_servidor_apache_e_coleta_de_dados()

{

int j;
j=0;
alterar_apache();
amostra=0;
filho_dados = fork(); //aqui criamos o processo filho

if ( filho_dados == 0 )
{
   close(0); //interrompe saida padrão
   close(NAmostras[0]);//Escrever no Pipe
   while (j==0)
   {

    sleep(1); //usleep(2500);
    gravar_dados();
    amostra++;
    //printf("Amostra Coletada: %d\n",amostra);
    write(NAmostras[1], &amostra, sizeof(int));

   }
}

else

{

close(NAmostras[1]);//Ler o Pipe

}
//esta alterando o apache e enviado para o banco ate aqui falta testar
}


void carga_servidor_apache()
{
filho_carga=fork();
if(filho_carga==0)
{
            close(1);
            while(1)
            {
    system("//usr/local/httpd-2.4.6/bin/./ab -r -q -k -n 1000000 -c 500  172.16.15.15/index.html");
             //usleep(100000);
            }
}
```

```
}

void entrada_kp_automatico()
{  int OldAmostra, acumulador, NewKeepAliveTimeout;

   acumulador=0;
   amostra=0;
   OldAmostra=0;
   //Inicio da Simulação
   KeepAliveTimeout = 11;
   MaxRequestWorkers = 256;
   ServerLimit = 256;
   op++;
   if(op>1)
   {
         kill( filho_dados, 9 );
         op--;
   }
   iniciar_servidor_apache_e_coleta_de_dados();
         //printf("EStou aqui\n");
   sleep(1);


   //Laço de Alteração de KeepAliveTimeout
   while (acumulador<2500)
   {
         //close(NAmostras[1]);
    OldAmostra=amostra;
         read(NAmostras[0], &amostra, sizeof(int));
         acumulador = acumulador + amostra - OldAmostra;

    NewKeepAliveTimeout = 11 + floor(9*sin(2*3.14*(acumulador/1200.0)));

    printf("Quantidade de Amostras Coletadas: %d\n",acumulador);
    printf("Valor do KeepAliveTimeout: %d\n",NewKeepAliveTimeout);


    if (NewKeepAliveTimeout != KeepAliveTimeout)
     {
         OldAmostra = 0;
         KeepAliveTimeout = NewKeepAliveTimeout;
      op++;
                  if(op>1)

                        {
                        kill( filho_dados, 9 );
                        op--;
                        }
                  pipe(NAmostras);
                  iniciar_servidor_apache_e_coleta_de_dados();
                  printf("!!!!!!!!!!!!!!Houve Variação de KeepAlive!!!!!!!!!!!!!!!!\n");
      sleep(1);
     }
     else
     {
         printf("Não Houve Variação de KeepAlive\n");
         sleep(1);
     }

   }
```

```
            amostra=acumulador;
            kill( filho_dados, 9 );
            op=0;

}


void entrada_mc_automatico()
{  int OldAmostra, acumulador, NewMaxClient;

   acumulador=0;
   amostra=0;
   OldAmostra=0;
   //Inicio da Simulação
   KeepAliveTimeout = 11;
   MaxRequestWorkers = 256;
   ServerLimit = 256;
   op++;
   if(op>1)
   {
          kill( filho_dados, 9 );
          op--;
   }
   iniciar_servidor_apache_e_coleta_de_dados();
          //printf("EStou aqui\n");
   sleep(1);


   //Laço de Alteração de KeepAliveTimeout
   while (acumulador<2500)
   {
          //close(NAmostras[1]);
    OldAmostra=amostra;
          read(NAmostras[0], &amostra, sizeof(int));
          acumulador = acumulador + amostra - OldAmostra;

    NewMaxClient = 150 + floor(105*sin(2*3.14*(acumulador/1200.0)));

    printf("Quantidade de Amostras Coletadas: %d\n",acumulador);
    printf("Valor do MaxClient: %d\n",NewMaxClient);


    if(abs((NewMaxClient - MaxRequestWorkers))>=20)
     {
          OldAmostra = 0;
          MaxRequestWorkers = NewMaxClient;
          ServerLimit = MaxRequestWorkers;
      op++;
                    if(op>1)

                              {
                              kill( filho_dados, 9 );
                              op--;
                              }
                    pipe(NAmostras);
                    iniciar_servidor_apache_e_coleta_de_dados();
                    printf("!!!!!!!!!!!!!!Houve Variação de MaxClient!!!!!!!!!!!!!!!!!\n");
      sleep(1);
     }
```

```
      else
      {
            printf("Não Houve Variação de MaxClient\n");
            sleep(1);
      }

   }

            amostra=acumulador;
            kill( filho_dados, 9 );
            op=0;

}

void Controle_Ref()
{
   int OldAmostra, acumulador, Old_MaxRequestWorkers;
   float PU_Usada;


   // Uk, Uk_1; Variaveis de controle inteiras
   //  Ek, Ek_1, Ref, Yk, R0, R1; Variaveis de controle de ponto flutuante

   Uk=0;
   Uk_1=0;
   Ek=0.0;
   Ek_1=0.0;
   Ref=0.86;
   Yk=0.0;
   R0 = 3737.3;
   R1=-2071.7;

   acumulador=0;
   amostra=0;
   OldAmostra=0;


   KeepAliveTimeout = 11;
   MaxRequestWorkers = 100 + Uk;

   Old_MaxRequestWorkers=0;

   op++;
   if(op>1)
   {
         kill( filho_dados, 9 );
         op--;
   }
   iniciar_servidor_apache_e_coleta_de_dados();
   sleep(1); // Saida do inicio do servidor


   while (acumulador<=3000)
   {

     OldAmostra=amostra;
          read(NAmostras[0], &amostra, sizeof(int));
     acumulador = acumulador + amostra - OldAmostra;
```

```
                retorno[0]=0;
                frase ="free";
                exec_comando(frase);
                quantidade_memória_servidor(retorno, &MemóriaTotal, &MemóriaUsada, &MemóriaLivre);

                PU_Usada = (float)MemóriaUsada/MemóriaTotal;
                Yk = PU_Usada;

                retorno[0]=0;
                frase ="top -b -n 1";
                exec_comando(frase);
                porcentagem_CPU(retorno, &CPUUsuario, &CPUSistema, &CPULivre);

                Ek_1 = Ek;
                Ek = Ref - Yk;
                Uk_1 = Uk;
Uk = Uk_1 + floor(R0*Ek) + floor(R1*Ek_1);


MaxRequestWorkers = 100 + Uk;


printf("O Numero Ek: %e\n", Ek);
printf("O Numero Uk: %d\n", Uk);
printf("O Numero MemóriaUsada: %ld\n", MemóriaUsada);
printf("O Numero MemóriaTotal: %ld\n", MemóriaTotal);
printf("O Numero PU_Usada: %.4f\n", PU_Usada);
printf("O Numero de Entrada na Planta é: %d\n", MaxRequestWorkers);

if ((MaxRequestWorkers>MaxSpareServers)&&(MaxRequestWorkers<ServerLimit))
{
                        Old_MaxRequestWorkers = 0;
                        op++;
                                if(op>1)

                                        {
                                        kill( filho_dados, 9 );
                                        op--;
                                        }
                                pipe(NAmostras);
                                iniciar_servidor_apache_e_coleta_de_dados();
                                printf("!!!!!!!!!!!!!!Houve Variação de MaxClient!!!!!!!!!!!!\n");
}
else
{
                        if ((MaxRequestWorkers>ServerLimit)&&(Old_MaxRequestWorkers!=1))
                        {
                         MaxRequestWorkers = ServerLimit;
        Old_MaxRequestWorkers = 1;

        op++;
                            if(op>1)
                                                {
                                                kill( filho_dados, 9 );
                                                op--;
                                                }
                                        pipe(NAmostras);
                                        iniciar_servidor_apache_e_coleta_de_dados();
                                        printf("!!!!!!!!!!!!!!Houve Variação de MaxClient!!!!!!!!!!!!!!!!\n");
```

```
                        }
                    else
                    {
                        if (Old_MaxRequestWorkers!=1)
                        {

                        MaxRequestWorkers = MaxSpareServers+1;
                        Old_MaxRequestWorkers = 1;

                op++;
                                        if(op>1)
                                                {
                                                        kill( filho_dados, 9 );
                                                        op--;
                                                }
                                        pipe(NAmostras);
                                        iniciar_servidor_apache_e_coleta_de_dados();
                                        printf("!!!!!!!!!!!!!!Houve Variação de MaxClient!!!!!!!!!!!!!!!!!\n");

                }
                  }
                 }
    sleep(2);
 }

 amostra=acumulador;
 kill( filho_dados, 9 );
 op=0;
}

void entrada_SBPA_automatico()
{
   int OldAmostra, acumulador, NewMaxClient;
   float Valor_PRBS, OldValor_PRBS;

   acumulador=0;
   amostra=0;
   OldAmostra=0;

   if ((PRBS= fopen(ARQ_PRBS, "r"))==NULL)
        {
                printf("Arquivo não pode ser aberto\n");
                sleep(1);
    }

   //Inicio da Simulação
   fscanf(PRBS, "%g\n",&Valor_PRBS);
   KeepAliveTimeout = 11;
   MaxRequestWorkers = 100 + (int)Valor_PRBS;
   NewMaxClient= 100 ;
   OldValor_PRBS=Valor_PRBS;

   op++;
   if(op>1)
   {
        kill( filho_dados, 9 );
        op--;
   }
   iniciar_servidor_apache_e_coleta_de_dados();
   sleep(1); // Saida do inicio do servidor
```

```
   while (acumulador<=3000)
   {
     fscanf(PRBS, "%g\n",&Valor_PRBS);
            //printf("Valor do PRBS é: %.2f\n", Valor_PRBS);
            //sleep(1);

     OldAmostra=amostra;
          read(NAmostras[0], &amostra, sizeof(int));
     acumulador = acumulador + amostra - OldAmostra;

     if (Valor_PRBS!=OldValor_PRBS)
     {

                    MaxRequestWorkers = NewMaxClient + (int)Valor_PRBS;


                    op++;
                          if(op>1)

                                  {
                                  kill( filho_dados, 9 );
                                  op--;
                                  }
                                  pipe(NAmostras);
                                  iniciar_servidor_apache_e_coleta_de_dados();
                                  printf("!!!!!!!!!!!!!!Houve Variação de MaxClient!!!!!!!!!!!!!!!!\n");


          OldValor_PRBS=Valor_PRBS;
            }
     sleep(1);
  }

  amostra=acumulador;
  kill( filho_dados, 9 );
  op=0;
}

void entrada_ss_automatico()
{  int OldAmostra, acumulador, NewMaxClient;

   acumulador=0;
   amostra=0;
   OldAmostra=0;

   //Inicio da Simulação
   KeepAliveTimeout = 11;
   MaxRequestWorkers = 100;
   ServerLimit = 100;
   op++;
   if(op>1)
   {
          kill( filho_dados, 9 );
          op--;
   }
   iniciar_servidor_apache_e_coleta_de_dados();
          //printf("EStou aqui\n");
```

```
sleep(1);


//Laço de Alteração de KeepAliveTimeout
while (acumulador<2500)
{
        //close(NAmostras[1]);
 OldAmostra=amostra;
        read(NAmostras[0], &amostra, sizeof(int));
        acumulador = acumulador + amostra - OldAmostra;

 //NewMaxClient = 150 + floor(105*sin(2*3.14*(acumulador/1200.0)));
 //NewMaxClient = acumulador; // Teste de amostragem com degrau constante

 if (acumulador >= 250)
 {
 NewMaxClient =250;

  if(acumulador >=500)
        {
                NewMaxClient = 50;
        }
 }
 else
 {
  NewMaxClient = 50;

 }




 printf("Quantidade de Amostras Coletadas: %d\n",acumulador);
 printf("Valor do MaxClient: %d\n",NewMaxClient);


 if(abs((NewMaxClient - MaxRequestWorkers))>=200)
        {OldAmostra = 0;
         MaxRequestWorkers = NewMaxClient;
         ServerLimit = MaxRequestWorkers;

    //StartServers= NewMaxClient-20;
    //MinSpareServers= NewMaxClient-20;
    //MaxSpareServers = NewMaxClient - 10;

    op++;
                if(op>1)

                        {
                        kill( filho_dados, 9 );
                        op--;
                        }
                pipe(NAmostras);
                iniciar_servidor_apache_e_coleta_de_dados();
                printf("!!!!!!!!!!!!!!Houve Variação de MaxClient!!!!!!!!!!!!!!!!\n");
   sleep(1);
  }
  else
  {
```

```
            printf("Não Houve Variação de MaxClient\n");
            sleep(1);
        }

    }

            amostra=acumulador;
            kill( filho_dados, 9 );
            op=0;

}



void main()
{


// Aumentando a prioridade de execução  do gerenciador
//Inicio
char prioridade[50];
sprintf(prioridade, "renice -n -20 -p %i", getpid());
system(prioridade);
//Fim

pipe(NAmostras);

inicializar_gerenciamento();
sleep(2);
op=0;
op1=0;
while(flag!=0)
{
            system("clear");
  printf("*********************Sistema de Gerenciamento do Apache***********************\n");
  printf("*                                                                            *\n");
  printf("*                                                                             *\n");
  printf("*                                                                             *\n");
  printf("*                    1 - INSERIR OS PARAMETROS DE CONTROLE                    *\n");
  printf("*                    2 - PESQUISA DE PARAMETROS ATUAIS                        *\n");
  printf("*                    3 - ALTERAR SERVIDOR APACHE E COLETAR DADOS              *\n");
  printf("*                    4 - PARAR COLETA DE DADOS                                *\n");
  printf("*                    5 - INICIAR CARGA NO SERVIDOR APACHE                     *\n");
  printf("*                    6 - RETIRAR CARGA NO SERVIDOR APACHE                     *\n");
  printf("*                    7 - ENTRADA AUTOMATICA PADRONIZADA(KP)                    *\n");
  printf("*                    8 - ENTRADA AUTOMATICA PADRONIZADA(MC)                   *\n");
  printf("*                    9 - ENTRADA AUTOMATICA PADRONIZADA(SS)                   *\n");
  printf("*                   10 - ENTRADA AUTOMATICA PADRONIZADA(SBPA)                  *\n");
  printf("*                   11 - CONTROLE PID(REF 0.86 Ultimo ensaio)                 *\n");
  printf("*                    0 - TERMINAR                                             *\n");
  printf("*                                                                             *\n");
  if (op>0)
  {
  printf("*    Coleta de Dados Sendo Executado                                        *\n");
  }
  if(op1>0)
  {
  printf("*    Carga Sendo Gerada no Servidor Apache                                    *\n");
  }
  printf("*                                        Número de Amostras no Ensaio: %d   *\n",amostra);
```

```
  printf("*                                                                    *\n");
  printf("**********************************************************************\n");
  printf("                    Digite a opção desejada: "); scanf("%d",&flag);
  printf("\n");

//Preparação dos valores dos recursos existentes
retorno[0]=0;
frase ="free";
exec_comando(frase);
quantidade_memória_servidor(retorno, &MemóriaTotal, &MemóriaUsada, &MemóriaLivre);

retorno[0]=0;
frase ="top -b -n 1";
exec_comando(frase);
porcentagem_CPU(retorno, &CPUUsuario, &CPUSistema, &CPULivre);
//Termino da preparação de recursos

//Preparar arquivos para execução do filtro
Preparando_Filtro();
//Final de preparação de arquivos para o filtro



switch(flag)
{
  case 1:
    inserir_parametros_controle();
    break;
  case 2:
    pesquisa_parametros_atuais();
    break;
  case 3:
    op++;
    if(op>1)
    {
          kill( filho_dados, 9 );
          }
    iniciar_servidor_apache_e_coleta_de_dados();
    break;
  case 4:
    op=0;
    kill( filho_dados, 9 );
    int a;
    a=1;
    while(a!=0) //laço para chegar na ultima informação inserida no pipe
    {
    a = read(NAmostras[0], &amostra, sizeof(int));

    }

    pipe(NAmostras);
    break;
  case 5:
          op1++;
          if(op1>1)
    {
          kill( filho_carga, 9 );
          }
          carga_servidor_apache();
          break;
```

```
  case 6:
    op1=0;
    kill( filho_carga, 9 );
    break;
  case 7:
    entrada_kp_automatico();
    break;

  case 8:
    entrada_mc_automatico();
    break;
  case 9:
    entrada_ss_automatico();
    break;
  case 10:
    entrada_SBPA_automatico();
    break;

  case 11:
    Controle_Ref();
    break;
  case 0:
    kill( filho_dados, 9 );
                    kill(filho_carga, 9 );
                    mysql_close(&conexao);
                    exit(0);
     break;

}



}

system("clear");
}
```

ANEXO II

SCRIPT DO BANCO DE DADOS

```
@OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;

SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL';

DROP SCHEMA IF EXISTS `Status_Apache` ;
CREATE SCHEMA IF NOT EXISTS `Status_Apache` DEFAULT CHARACTER SET latin1 ;
SHOW WARNINGS;
USE `Status_Apache` ;

-- -----------------------------------------------------
-- Table `HTTP01`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `HTTP01` ;

SHOW WARNINGS;
CREATE  TABLE IF NOT EXISTS `HTTP01` (
  `id` INT(11) NOT NULL AUTO_INCREMENT ,
  `Data` DATE NOT NULL ,
  `Hora` TIME NOT NULL ,
  `MTotal` BIGINT(20) NOT NULL ,
  `MUsada` BIGINT(20) NOT NULL ,
  `MLivre` BIGINT(20) NOT NULL ,
  `CUsuario` DOUBLE NOT NULL ,
  `CSistema` DOUBLE NOT NULL ,
  `CLivre` DOUBLE NOT NULL ,
  `StartServers` INT(11) NULL DEFAULT NULL ,
  `MinSpareServers` INT(11) NULL DEFAULT NULL ,
  `MaxSpareServers` INT(11) NULL DEFAULT NULL ,
  `MaxRequestWorkers` INT(11) NULL DEFAULT NULL ,
  `MaxConnectionsPerChild` INT(11) NULL DEFAULT NULL ,
  `ServerLimit` INT(11) NULL DEFAULT NULL ,
  `KeepAliveTimeout` INT(11) NULL DEFAULT NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB
AUTO_INCREMENT = 130068
DEFAULT CHARACTER SET = latin1;
```

```
SHOW WARNINGS;

-- -----------------------------------------------------
-- procedure STATUS
-- -----------------------------------------------------

USE `Status_Apache`;
DROP procedure IF EXISTS `STATUS`;
SHOW WARNINGS;
DELIMITER $$
USE `Status_Apache`$$
CREATE DEFINER=`root`@`%` PROCEDURE `STATUS`(IN MTOTAL BIGINT(20),
 IN MUSADA BIGINT(20),
 IN MLIVRE BIGINT(20),
 IN CUSUARIO DOUBLE,
 IN CSISTEMA DOUBLE,
 IN CLIVRE DOUBLE,
 In CStartServers INT,
 IN CMinSpareServers INT,
 IN CMaxSpareServers INT,
 IN CMaxRequestWorkers INT,
 IN CMaxConnectionsPerChild INT,
 IN CServerLimit INT,
 IN CKeepAliveTimeout INT)
BEGIN

INSERT INTO HTTP01
(Data, Hora, MTotal, MUsada, MLivre, CUsuario, CSistema, CLivre, StartServers, MinSpareServers, MaxSpareServers, MaxRequestWorkers,MaxConnectionsPerChild, ServerLimit, KeepAliveTimeout)
VALUES(curdate(), curtime(), MTOTAL, MUSADA, MLIVRE, CUSUARIO, CSISTEMA, CLIVRE, CStartServers, CMinSpareServers, CMaxSpareServers, CMaxRequestWorkers, CMaxConnectionsPerChild, CServerLimit, CKeepAliveTimeout);
COMMIT;
END$$

$$
DELIMITER ;
SHOW WARNINGS;
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
```

```
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```