Implementação e Avaliação Experimental de um Agente de Rejuvenescimento de Software para Servidores Web

Rivalino Matias Jr.[1], Eliete Marchioro[2], Elizabeth Specialski[2], Paulo J. F. Filho[2]

[1] PPGEP/PLAB-PPGCC – Universidade Federal de Santa Catarina
[2] Pós-graduação em Ciência da Computação – Universidade Federal de Santa Catarina

rivalino@eps.ufsc.br, eliete@superip.com.br,
{beth, freitas}@inf.ufsc.br

Abstract. *The phenomenon of software aging has been a target of many recent researches mainly focusing on its proving, modeling and experimental verification. In most of the cases, the approach adopted to managing this problem has been a technique named software rejuvenation. The first step to use this technique is the characterization of the software aging in order to built analytical models for the estimation of parameters used to apply the rejuvenation. This work presents the results of an experimental study whose objective was to evaluate a software rejuvenation agent. The agent developed was applied to a web server environment whose aging symptoms were identified and evaluated in a previous phase. The main result of this paper is the validation of a rejuvenation agent for Apache web servers which can be extended to accomplish others applications.*

Resumo. *O fenômeno do envelhecimento de software tem sido alvo de inúmeras pesquisas voltadas, principalmente, para sua comprovação, modelagem e verificação experimental. Na maioria dos casos, o enfoque adotado para tratar deste fenômeno tem sido a utilização de técnicas de rejuvenescimento de software. O primeiro passo no sentido de se utilizar tais técnicas é a caracterização do envelhecimento, onde modelos analíticos podem ser construídos para a estimação dos parâmetros a serem usados para fins de rejuvenescimento. Este trabalho apresenta os resultados de um estudo experimental, cujo objetivo foi de avaliar o desempenho de um agente de rejuvenescimento de software. Este agente foi aplicado a um servidor web, cujos sintomas de envelhecimento foram identificados e avaliados em uma fase anterior à construção do agente. Como resultado, implementou-se um agente de rejuvenescimento de software para servidores Apache, o qual pode ser estendido a fim de abranger novos tipos de aplicações.*

1. Introdução

Diversos estudos sobre confiabilidade e tolerância a faltas de software têm identificado e caracterizado o fenômeno do envelhecimento de software (*software aging*) [Huang et al. 1995]. Segundo Trivedi et al. (2000), o envelhecimento de software pode ser definido como uma degradação, contínua e crescente, do estado interno de um processo[1] de software durante a sua execução. As causas do envelhecimento de software tem sido atribuídas, principalmente, a faltas do tipo *Heisenbugs* [Gray 1986]. O acumulo destas faltas, durante longos períodos de execução do processo, provocam inconsistências de dados, erros numéricos, exaustão de recursos do sistema operacional, dentre outros problemas correlatos. Os seus principais efeitos vão desde a degradação progressiva da performance até o travamento completo (*crash*) do sistema [Avritzer et al. 1997].

O fenômeno do envelhecimento de software tem sido verificado, em maior intensidade, naqueles processos de software que executam continuamente e por um longo período de tempo. Neste caso, são candidatos naturais os processos do tipo servidores e também sistemas dedicados tais como roteadores de rede e controladores de processos industriais, dentre outros. Entretanto, aplicações de propósitos gerais (ex. editores de texto, navegadores, etc.) também estão suscetíveis aos efeitos provocados pelo envelhecimento de software [Trivedi et al. 2000].

A fim de mitigar os efeitos do envelhecimento, Huang et al. (1995) propõe uma técnica chamada rejuvenescimento de software. Esta técnica é um mecanismo pró-ativo para a contenção e remoção do acumulo de faltas que causam o envelhecimento de um processo de software, visando ampliar a sua longevidade operacional. Basicamente, esta técnica consiste em parar o processo envelhecido e reinicializar o seu estado interno. Maiores detalhes serão apresentados na seção 4.

Este trabalho avalia os efeitos causados pelo envelhecimento de software em servidores web Apache, bem como os benefícios na utilização de uma técnica de rejuvenescimento sobre estes servidores. Um estudo, com base em resultados de experimentos controlados, foi conduzido para se caracterizar o comportamento do envelhecimento. A partir destes resultados, foi implementado um agente de rejuvenescimento de software (*ARS*), com objetivo de avaliar sua aplicabilidade junto aos processos com sintomas de envelhecimento.

A escolha da aplicação do *ARS* voltada para servidores web deve-se, sobretudo, a dois importantes aspectos. Primeiro, pela relevante função que os servidores web exercem na infra-estrutura atual da Internet. Segundo, pelo fato de terem tipicamente uma execução contínua, de longa duração, e passível de flutuações diversas em suas cargas de trabalho (*workloads*). Estas aplicações são candidatas naturais a experimentarem os efeitos do envelhecimento citados anteriormente. A escolha do servidor web Apache justifica-se pela sua ampla utilização, encontrado em aproximadamente 67% dos *sites* pesquisados pela Netcraft (2004). Aliado a isso, a disponibilidade de seu código fonte oferece maior flexibilidade para a experimentação.

As demais seções deste artigo estão descritas a seguir. A seção 2 apresenta outros trabalhos desenvolvidos nesta área, buscando ressaltar seus principais resultados.

---

[1] Neste trabalho o termo processo de software se refere à instância de um programa sendo executada na memória principal.

Na seção 3, um projeto de experimentos é realizado, visando identificar e caracterizar a presença do envelhecimento na instalação do Apache utilizada neste estudo. A partir da análise dos resultados obtidos na seção 3, uma implementação de *ARS* é apresentada na seção 4, a fim de demonstrar a sua aplicabilidade para o caso abordado (servidores Apache). Finalmente, a seção 5 apresenta as conclusões da pesquisa.

2. Trabalhos Relacionados

Em [Avritzer et al. 1997] um estudo sobre o envelhecimento de software em sistemas de telecomunicação é apresentado, considerando a reinicialização programada de uma central de comutação, a fim de restaurar o seu estado interno à condição inicial (capacidade nominal) após um determinado período de execução ininterrupta. Para isso foi proposto um modelo estocástico, baseado em cadeias de Markov, a fim de estimar o melhor momento para executar a reinicialização da central. Em Huang et al. (1995), foi utilizada uma variável de custo de paradas (*downtime*), programadas ou não, como parâmetro para um modelo utilizado na estimação do momento de execução do mecanismo de rejuvenescimento de software. Esta variável representa tanto custo financeiro quanto qualquer outra medida significativa para apoiar a decisão de realizar ou não as reinicializações do sistema. Seguindo um enfoque diferente, Trivedi et al. (2000) propõem um modelo probabilístico que considera não só o tempo de execução, mas também a carga de trabalho submetida contra o sistema estudado. O estudo teve como ambiente um sistema operacional Unix. Foram derivadas séries temporais, a partir dos dados monitorados, a fim de se qualificar e estimar o grau de envelhecimento em cada recurso avaliado. As conclusões do trabalho apontam que o modelo parametrizado, tanto com o tempo de execução quanto com a carga de trabalho, oferece maior precisão em relação àqueles que desconsideram a carga de trabalho em suas estimativas. Uma limitação do modelo é a estimação do MTTF (*Mean Time To Failure*), baseando-se em apenas um único recurso por vez. Os autores sugerem que o trabalho seja estendido no sentido de se ter uma predição global, considerando os efeitos do envelhecimento combinados em diversos recursos. Utilizando uma abordagem analítica diferente, o trabalho de Shereshevsky et al. (2003) critica o enfoque adotado por Vaidyanathane Trivedi (1998), que se baseia em modelos lineares para estimar as tendências de envelhecimento nos recursos monitorados. Em sua análise, Shereshevsky et al. argumenta que, a abordagem linear adotada pelos autores, não é adequada quando se tem um grande aumento do intervalo de confiança, que ocorre em períodos que as variáveis estudadas demonstram comportamentos não lineares. Sua proposta para contornar esta limitação é a utilização da teoria de fractais para estimar os tempos de exaustão de recursos.

Um estudo voltado para o envelhecimento de servidores web é apresentado em Li et al. (2002). Neste, são realizados experimentos com o objetivo de identificar, caracterizar e estimar o envelhecimento em servidores web apache. A partir dos dados coletados do sistema operacional, modelos estatísticos ARMA são usados para a análise das séries temporais obtidas, buscando-se detectar e estimar o envelhecimento dos processos em questão. A proposta de Li et al. (2002) oferece menor complexidade computacional do que outras similares desenvolvidas em [Garg et. 1998] e [Vaidyanathan et al. 1998]. A periodicidade em que os parâmetros do modelo devem ser estimados é um ponto não abordado no trabalho, apesar de sua importância significativa para aplicabilidade do mesmo.

Em Huang et al. (1995) uma proposta de interface para a implementação de rejuvenescimento é apresentada, porém de forma bastante genérica. A partir de uma extensa pesquisa a cerca de implementações de *ARS*, encontrou-se no trabalho de Castelli et al. (2001) o de maior detalhamento dentre os pesquisados. Neste, o *ARS* é programado para executar o rejuvenescimento tanto baseado em um intervalo fixo de tempo quanto por meio de predição. No modo de predição, o *ARS* monitora diversos recursos do sistema (ex. memória virtual disponível) e, baseando-se em um histórico recente de seus comportamentos, são realizadas projeções da exaustão destes recursos. O mesmo trabalho apresenta uma validação numérica, de ambos os modos de operação, baseada em cadeias de Markov.

Buscando contribuir com a literatura atual, o presente trabalho visa avaliar, experimentalmente, os efeitos do envelhecimento em servidores web Apache, a fim de se verificar a aplicabilidade do rejuvenescimento nestes processos. Seguindo uma abordagem diferente de Li et al. (2002), o *ARS* proposto neste trabalho não considera o recurso *memória principal disponível* para efeitos de predição. A justificativa desta escolha, bem como os demais detalhes da implementação do agente, serão apresentados na seção 4.

3. Caracterização do Envelhecimento de Software no Apache

Antes de implementar o *ARS*, buscou-se verificar se o servidor web, utilizado durante os testes, estaria sofrendo ou não algum tipo de envelhecimento de software. Em Li et al. (2002), comprovou-se a presença de envelhecimento no Apache, contudo, aqueles resultados não podem ser utilizados neste trabalho a fim de alimentar o processo de implementação do agente. Esta impossibilidade deve-se, principalmente, ao fato que a reprodução ou utilização de resultados, obtidos em outros ensaios, só é possível quando se mantém exatamente o mesmo ambiente de testes. Esta exigência é justificada devido à grande sensibilidade que este fenômeno apresenta no tocante a mudanças ambientais (ex. sistema operacional, versões do software servidor, etc.). Portanto, fez-se necessário a realização de um projeto experimental prévio a construção do agente, a fim de comprovar a existência do envelhecimento no ambiente de testes criado para este estudo. A seguir serão apresentados os principais componentes utilizados durante os experimentos.

3.1 Ambiente de Teste

Basicamente, foram usadas três máquinas, sendo duas geradoras de tráfego (K6-II 350 MHz, 128 MBytes, Rede 10/100) e uma terceira como o servidor web (P4 1.4 Ghz, 512 MBytes, Rede 10/100). A interconexão das três máquinas foi realizada através de um *switch* 3COM 10/100 Mbps. A Figura 1 ilustra o ambiente utilizado.

Para a monitoração do ambiente servidor, criou-se um programa chamado *sysmon*. Como se pode observar na Figura 1, o *sysmon* realiza dois tipos básicos de monitoração. A primeira é referente aos recursos do sistema operacional, sendo realizada através da leitura e interpretação de diversos arquivos no diretório */proc* (ex. */proc/meminfo*) do Linux. Além disso, o *sysmon* utiliza programas adicionais (ex. *netstat*, *sar*) para capturar dados complementares sobre o ambiente operacional do servidor. No segundo tipo de monitoração, o *sysmon* coleta dados específicos sobre os

processos do Apache, como por exemplo, a quantidade de memória utilizada e compartilhada pelos servidores *httpd*.

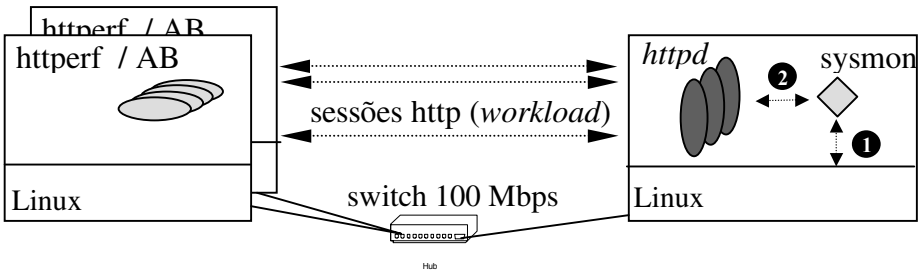

Figura 1. Laboratório de Testes

Para realizar a geração das cargas de trabalho, foram utilizadas as ferramentas *httperf* [Mosberger et al. 1998] e *ab* (*Apache Bench*)[2]. Apesar da grande flexibilidade disponível no *httperf*, principalmente no que tange à geração de cargas de trabalho (*workloads*), o *ab* foi usado de forma complementar para aqueles casos onde o número de conexões concorrentes deveria permanecer constante. Esta funcionalidade, apesar de não disponível na interface do *httperf*, também poderia ser atingida através do ajuste de outros parâmetros tais como a distribuição do tempo entre chegadas, taxa de requisições por segundo, dentre outros. Contudo, procurando manter uma padronização destes parâmetros, evitou-se tal artifício, utilizando o *ab* em tais cenários. Outra finalidade do *ab* foi a de validar resultados obtidos com o *httperf*.

A fim de desempenhar o papel de servidor web, optou-se pelo Apache (versão 2.0.48)[3]. Foi padronizado que seriam utilizados 200 processos servidores *httpd*, suportando assim um número máximo de 200 conexões concorrentes. Todos os 200 processos foram criados durante a inicialização do Apache e permaneceram carregados por todo o período de cada experimento. Deste modo evitou-se a remoção de servidores *httpd* ociosos que poderiam estar acima do especificado na diretiva *MaxSpareServers*. Complementarmente, atribui-se o valor zero para as diretivas *MaxRequestperChild* e *MaxKeepAliveRequests*, garantindo assim a não reciclagem dos processos pelo número de requisições atendidas. De forma geral estas configurações objetivaram a maior exposição dos processos servidores aos efeitos do envelhecimento, já que se garante que cada processo servidor *httpd* executará durante todo o período do experimento. A vantagem desta abordagem, em relação ao que foi verificado nos trabalhos citados na seção 2, é a maximização do tempo de exposição dos processos ao fenômeno sob estudo e conseqüente redução no tempo para se detectar problemas desta natureza.

3.2 Projeto de Experimentos

O objetivo desta etapa foi verificar se o servidor web, instalado no ambiente de teste, estava sofrendo ou não de envelhecimento de software como reportado em Li et al. (2002). A seleção dos fatores e seus respectivos níveis foram, em parte, baseados nos trabalhos de Trivedi et al. (2000) e Li et al. (2002). Os seguintes fatores foram

[2] Disponível com a distribuição do Apache (http://www.apache.org).

[3] Ultima versão disponível em 18/04/2004.

definidos: *taxa de requisições por segundo*, *tamanho de página* e *tipo de página*. Para simplificação, estes serão chamados de *txReq*, *tamPág* e *tipoPág* respectivamente.

Para cada fator foram definidos dois níveis, resultando em um projeto do tipo fatorial completo 2^k [Jain 1991]. O número de fatores é representado por , sendo a quantidade de níveis por fator. O número total de experimentos é obtido através da equação (1):

$$\eta = \prod_{i=1}^{\kappa} \tau_i \qquad (1)$$

onde =3 e $_i$ = 2, tem-se um total de 8 experimentos.

A definição dos níveis para cada fator considerou dois cenários de operação. O primeiro, caracterizado como *normal*, estabeleceu cargas de trabalho que consomem 50% da capacidade efetiva do servidor web. O segundo, caracterizado como *limite*, utiliza 90% desta capacidade. Em [Li et al. 2002], sugere-se o uso de cargas iguais ou superiores a 90% da capacidade efetiva do sistema a fim de verificar a presença de envelhecimento de software. A Tabela 1 apresenta os valores utilizados para cada um dos fatores e seus respectivos níveis.

Tabela 1. Fatores e Níveis Selecionados

Normal (50%)	**Limite (90%)**
tamPág = 197 Kbytes	*TamPág* = 2 Mbytes
tipoPág = Estática	*TipoPág*= Dinâmica
txReq (req/s) = 30(E1), 2,75(E3), 15(E5), 2,75(E7).	*txReq* (req./s) = 54(E2), 4,95(E4), 27(E6), 4,95 (E8).

A seguir serão descritos os procedimentos usados na determinação destes valores.

3.2.1 Especificação das Cargas de Trabalho

Para a definição do valor associado ao nível *normal*, do fator *tamPágI*, foi realizada uma pesquisa com 6.000 páginas de um provedor de serviços Internet. Foram selecionadas as 10 mais acessadas em um período de 6 meses, de onde extraiu-se um tamanho de página médio igual 198 KBytes. Este valor foi confrontado com o tamanho de outras páginas obtidas de *sites* na Internet, selecionados de forma *adhoc*, para verificar a sua representatividade, que se comprovou adequada. Quanto ao nível *limite* deste fator, buscou-se um tamanho representativo daqueles casos em que o servidor web transfere objetos de dados maiores do que páginas *html* convencionais. Atualmente, verifica-se como uma prática comum, a distribuição de arquivos de programas, manuais, atualizações de antivírus, dentre outros, através de servidores web. A fim de determinar um valor que representasse estes casos, foram obtidas três amostras, sendo elas: Arquivo de atualização de antivírus (2,5 MBytes)[4], arquivos de instalação dos

[4] Atualização do antivírus da empresa McAffee. Disponível em http://download.mcafee.com/us/updates/ Acessado em 17/03/2004.

programas de declaração do IRRF (2,5 MBytes)[5] e currículo Lattes/CNPq (18,5MBytes)[6]. No caso do currículo Lattes, considerou-se a opção de transferência fragmentada em 13 arquivos de 1,4 KBytes. Baseado nestes valores, estimou-se um valor médio de 2,0 MBytes para representar o nível *limite* deste fator.

O fator *tipoPág* refere-se a forma como a página está disponível, ou seja, através de conteúdo estático ou criado dinamicamente. O nível *normal* representa páginas com conteúdo estático (arquivo.html + imagens). Já o nível *limite* representa páginas montadas dinamicamente. Para atingir este propósito, foi criado um programa em C para simular o acesso a um banco de dados e gerar as páginas requisitadas, de acordo com os tamanhos definidos pelo fator *tamPág*.

Na determinação dos níveis do fator *txReq*, foram necessários testes preliminares a fim de obter a capacidade efetiva do servidor. A medida utilizada, como métrica de performance, foi a taxa de respostas por segundo que o servidor suporta para cada carga de trabalho definida. Esta foi selecionada por ser representativa na avaliação da capacidade de servidores web [Li et al. 2002].

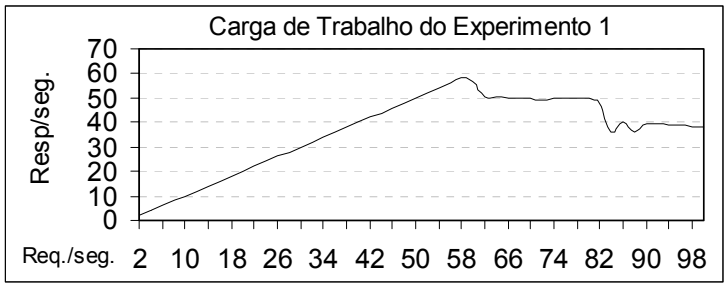

Figura 2. Capacidade efetiva em *txReq*

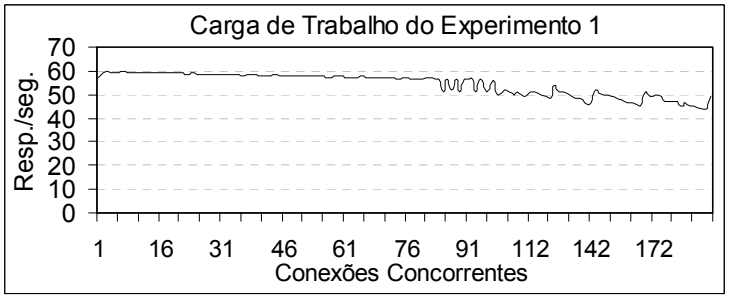

Figura 3. Validação através da *txReq* através da ferramenta *ab*

As Figuras 2 e 3 apresentam os resultados obtidos para a capacidade efetiva do servidor com os valores (*tamPág* = 197 KB e *tipoPág* = Estática). Como se pode verificar na Figura 2, valores acima de 58 requisições/seg. excedem a capacidade do servidor reduzindo sua taxa de respostas. A Figura 3 apresenta os resultados obtidos através da ferramenta *ab*, a fim de validar aqueles encontrados com o *httperf* (Figura 2). Neste caso, apesar da abordagem ter sido a variação do número de conexões

[5] Disponível em http://www.receita.fazenda.gov.br/PessoaFisica/IRPF/2004/progIRPF2004umdisco.htm Acessado em 17/03/2004.

[6] Disponível em http://lattes.cnpq.br/. Acessado em 17/03/2004.

concorrentes, o mesmo limite foi encontrado, ou seja, o melhor desempenho que o servidor web atingiu foram 58 respostas/seg. Este mesmo conjunto de testes foi realizado para cada carga de trabalho utilizada nos 8 experimentos, resultando nos valores listados na Tabela 1.

3.2.2 Realização dos Experimentos e Análise dos Resultados

Durante os testes preliminares para determinar a capacidade do servidor web, verificou-se que dentre todos os recursos monitorados (*cpu, memória virtual, conexões de tcp, processos httpd, processos do sistema*), a *memória virtual* foi o que sofreu maior influência. A mesma conclusão foi encontrada nos trabalhos de [Trivedi et al 2000] e [Li et al. 2002] durante seus testes. Naqueles, comprovou-se que a degradação da memória era conseqüência direta do envelhecimento de software, contudo não se demonstrou a origem deste envelhecimento.

A fim de se determinar qual o fator tem maior influência neste consumo de memória, o presente estudo considerou, inicialmente, a hipótese de que o envelhecimento do Apache teria sido a principal causa do maior consumo de memória durante os testes de capacidade. Deste modo, definiu-se como *variável resposta* (*VR*), para o projeto de experimentos, o consumo de memória dos processos *httpd.* Para a verificação da influência de cada fator sobre *VR*, foi utilizado o método da tabela de sinais [Jain 1991]. A matriz utilizada na computação dos efeitos está apresentada na Tabela 2.

Tabela 2. Tabela de sinais para computação dos efeitos do projeto 2^k

EXP.	I	A	B	C	AB	AC	BC	ABC	*VR*
E1	1	-1	-1	-1	1	1	1	-1	50
E2	1	1	-1	-1	-1	-1	1	1	53
E3	1	-1	1	-1	-1	1	-1	1	49
E4	1	1	1	-1	1	-1	-1	-1	52
E5	1	-1	-1	1	1	-1	-1	1	130
E6	1	1	-1	1	-1	1	-1	-1	141
E7	1	-1	1	1	-1	-1	1	-1	461
E8	1	1	1	1	1	1	1	1	470
A=*txReq*, B=*tamPag*, C=*tipoPag*									

Os valores –1 e 1 representam, respectivamente, os níveis *normal* e *limite*. A coluna *VR* refere-se aos resultados associados à *variável resposta* (consumo de memória dos processos *httpd*), os quais estão expressos em *MBytes*. Os valores de *VR* foram coletados no final de cada experimento, cuja duração foi de aproximadamente 8 horas, com um desvio padrão de ±45 minutos, dependendo do experimento. Durante cada experimento o *httperf* foi executado em diversas iterações, gerando um conjunto de 4000 requisições. O número de requisições foi definido a partir das recomendações descritas em [Mosberger et al. 1998] e no manual da ferramenta, a fim de prover um número mínimo de amostras geradas, que garantam estatísticas confiáveis.

As Figuras 4 e 5 apresentam o comportamento da memória nos quatro primeiros experimentos.

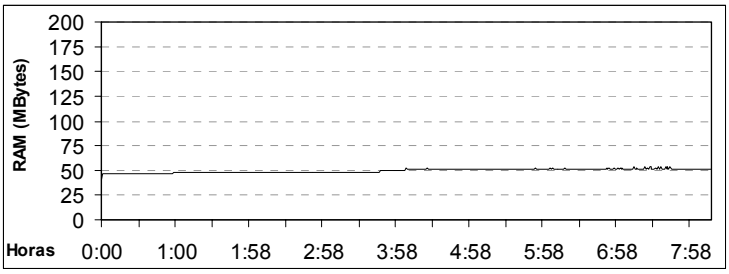

Figura 4. Consumo de Memória do Apache (E1,E2,E3,E4)

A Figura 4 foi gerada a partir dos dados coletados do experimento E1. Contudo os demais experimentos (E2, E3 e E4) apresentaram o mesmo comportamento, motivo pelo qual seus gráficos foram suprimidos. Destaca-se que em E3 e E4, apesar da utilização de páginas de 2,0 MBytes, o consumo de memória do Apache não se alterou em relação aos experimentos anteriores (196,0 KB). Entretanto, verificou-se que em E1 e E2 ocorreu uma maior alocação de memória pelo sub sistema de entrada e saída (*buffer-cache*) do Linux.

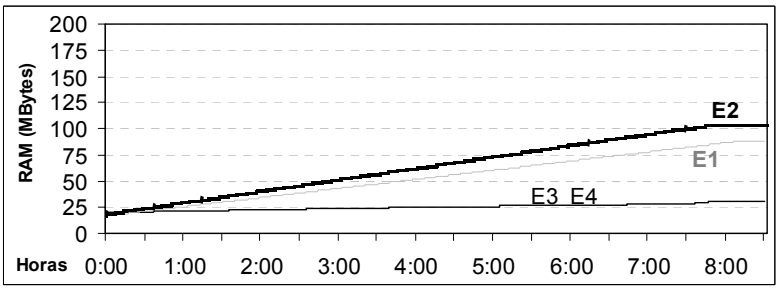

Figura 5. Comportamento do Buffer-Cache (E1,E2,E3,E4)

Como se pode observar na Figura 5, em E1 e E2 o consumo de memória foi aproximadamente 4 vezes superior ao demonstrado por E3 e E4. No término de E2, o valor do *buffer-cache* respondia por 66,5% da memória total utilizada, contra 35% alocados para os processos *httpd*.

Com base nestes experimentos, observa-se que os processos do Apache aparentaram boa estabilidade, no que tange ao consumo de memória, mesmo em condições de maior transferência de dados (E3 e E4) e freqüência elevada de operações de acesso em arquivos (E1 e E2). Este comportamento indica, para os 4 primeiros experimentos, a ausência de envelhecimento sobre o recurso memória (hipótese em questão).

O comportamento do Apache, relativo ao consumo de memória, nos quatro últimos experimentos foi sensivelmente diferente daquele apresentado nos anteriores. A Figura 6 apresenta os resultados obtidos com os experimentos E5 e E6. Nos primeiros 40 minutos, de ambos experimentos, os processos *httpd* cresceram em média o dobro do registrado no final dos experimentos anteriores. A partir de então, o crescimento continuou, de forma gradativa, em pequenos incrementos. Verificando o tamanho dos processos *httpd* antes e depois dos testes, o crescimento médio foi de aproximadamente

80 MBytes e 90 MBytes para os experimentos E5 e E6 respectivamente. Considerando o incremento que ocorreu na maior parte do experimento, ou seja, após os 40 minutos iniciais, o incremento foi em torno de 25 MBytes (E5) e 38 MBytes (E6). Distribuindo estes valores para os 200 processos, tem-se um valor médio de 125 KB (E5) e 190 KB (E6) por processo *httpd* durante aproximadamente 8 horas de teste. Se comparado ao comportamento estável demonstrado nos quatro primeiros experimentos, estes valores são significativos.

Pelo fato dos processos não terem sido substituídos durante a experimentação e dado que a carga de trabalho foi constante, tudo leva a crer que nestes dois ensaios os processos estão envelhecendo durante seu tempo de execução.

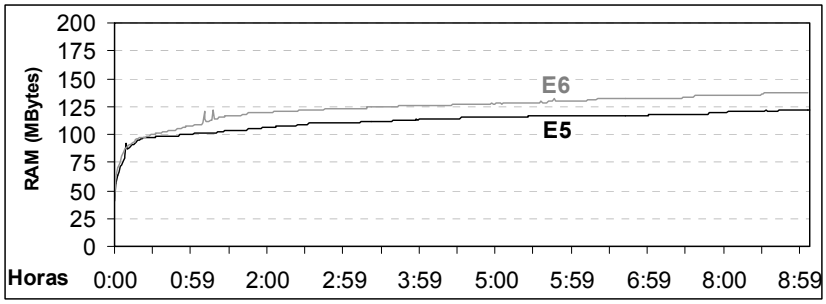

Figura 6. Consumo de Memória do Apache (E5,E6)

Nos experimentos E7 e E8, o comportamento do sistema foi similar aos dois anteriores (E5 e E6), diferenciando-se apenas na intensidade do consumo de memória. A Figura 7 apresenta estes resultados. Vale ressaltar que a escala do eixo Y foi modificada para acomodar os novos valores reportados. Como em E5 e E6, um período de crescimento acentuado ocorreu no início do teste (≅ 90 min.) e posteriormente o crescimento dos processos continua gradativamente por todo o período de execução.

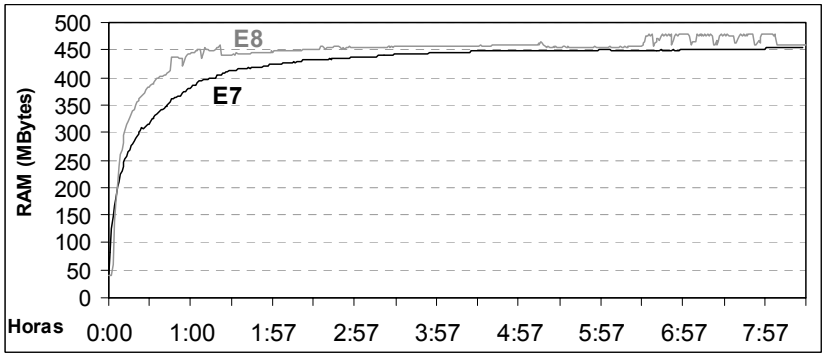

Figura 7. Consumo de Memória RAM (E7 e E8)

Observa-se que no instante 6:10hs, na linha representando E8, inicia-se uma oscilação que continua até o final do teste. Exatamente neste momento, verificou-se o início da utilização da área reservada para *swap*, o que provocou uma queda substancial de performance, incrementando consideravelmente (acima de 60 seg.) o tempo de resposta. Este aumento no tempo de resposta causou o fechamento de diversas conexões

por *timeout*[7]. Este período de instabilidade durou aproximadamente 30 minutos no decorrer da fase de oscilação.

Através dos valores obtidos, pode-se concluir que no conjunto de experimentos E5-E8 esteja ocorrendo o envelhecimento dos processos *httpd*. Além disso, conclui-se que o fator de maior efeito tem sido o *tipoPág*. A fim de verificar, analiticamente, a influência de cada fator neste conjunto de experimentos, resolveu-se a tabela de sinais, cujos resultados estão apresentados a seguir.

Tabela 3. Tabela de sinais resolvida

I	**A**	**B**	**C**	**AB**	**AC**	**BC**	**ABC**
1E+06	24352	657840	996704	-1584	12752	662416	-3728
175762	3044	82230	124588	-198	1594	82802	-466
SQT	**SQA**	**SQB**	**SQC**	**SQAB**	**SQAC**	**SQBC**	**SQABC**
2,3E+11	7,4E+07	5,4E+10	1,2E+11	313632	2E+07	5,5E+10	1737248
Variação de *VR* explicada pelos fatores e suas combinações							
	0,03%	**23,19%**	**53,25%**	**0,00%**	**0,01%**	**23,52%**	**0,00%**

A segunda linha da tabela contém os valores dos efeitos de cada fator sobre *VR*. Estes valores são calculados através do produto escalar do vetor de coeficientes (coluna de cada fator) pelo vetor de respostas (coluna da *VR*). Na terceira linha têm-se os valores dos efeitos divididos pelo número de experimentos. A equação 2 representa estes procedimentos no caso da a coluna A.

$$_{Efeito\ Fator}\, A = \frac{1}{8} \cdot \left(X_A\right)^t \cdot VR \qquad (2)$$

A contribuição de cada fator e combinações, sobre *VR*, é calculada utilizando-se as equações 3 e 4.

$$SQT = \sum_{i=1}^{2^3} \left(r_i - \bar{r}\right)^2 \qquad (3)$$

onde $\bar{r}$ compreende a média das respostas dos 8 experimentos (linha 3 coluna I).

$$SQA = {}_{EfeitoFator} A^2 \cdot 2^k \qquad (4)$$

A proporção apresentada na última linha da tabela é o resultado da divisão da variação explicada por cada fator ou combinações (ex. SQAB) pela variação total (SQT). A partir destes resultados, comprovou-se que o fator *tipoPág* é o que provoca maior influencia (53,2%), dentre todos analisados. O nível que causa este aumento gradativo no tamanho dos processos é quando se tem a criação dinâmica de páginas.

Em ambientes de produção, o incremento gradativo no tamanho dos processos deve ser controlado, pois ao se atingir valores próximos do limite físico da memória principal o sistema operacional iniciará procedimentos de *swapping*. Como conseqüência deste cenário, tem-se um impacto direto e negativo sobre o desempenho

[7] O valor de *timeout* adotado em todos os experimentos foi de 60 segundos.

de muitas aplicações. Esta mesma situação foi verificada no experimento E8, onde o gerador de tráfego reportou o fechamento de várias conexões no mesmo período em que se iniciou o processo de *swapping*.

Neste sentido, a próxima seção apresenta a implementação de um *ARS*, o qual foi usado para controlar o tamanho dos processos do Apache durante testes com o mesmo cenário do experimento E8.

4. Agente de Rejuvenescimento de Software (*ARS*)

A tecnologia de agentes de software tem sido amplamente utilizada em diversas áreas da ciência da computação, tendo grande aplicação no gerenciamento de performance de sistemas [Kraus 2000]. Um *ARS* tem propósito pró-ativo, pois seu objetivo não é remover a falta (*defeito*) causadora do envelhecimento, mas sim aliviar seus efeitos mantendo o sistema/processo em níveis "*saudáveis*" de operação. Para a implementação de um *ARS* é fundamental caracterizar o fenômeno do envelhecimento a ser tratado, para posteriormente se definir a melhor abordagem de aplicação da técnica de rejuvenescimento. Neste trabalho, a fase de caracterização foi realizada através dos experimentos descritos na seção 3. A partir daqueles resultados foi possível identificar as condições que mais favorecem o envelhecimento dos processos do Apache, permitindo agora definir a abordagem utilizada pelo *ARS*. A seguir serão apresentados os principais aspectos envolvidos no projeto do agente.

4.1. Arquitetura Adotada

Atualmente, existem duas principais abordagens na implementação de agentes de rejuvenescimento de software que, de acordo com a taxonomia de [Vaidyanathan et al. 2001b], são denominadas *closed-loop* e *open-loop*.

No enfoque *open-loop*, o *ARS* está programado para executar o rejuvenescimento independente do estado do sistema[8], normalmente executando-o em intervalos de tempo pré-determinados. No *closed-loop* o *ARS* monitora o sistema, continuamente, a fim de obter dados que servirão para inferir a melhor técnica e momento para se realizar o rejuvenescimento. Esta coleta e análise dos dados podem ser realizadas tanto *on-line* quanto *off-line*. No modo *on-line*, se tem o processo de monitoração e análise sendo realizados durante a execução do *ARS*, diferente do *off-line* onde estas atividades são realizadas previamente à sua execução. Neste último, normalmente são utilizados dados históricos do sistema (ex. arquivos de *log*) para fins de estimação dos parâmetros (ex. limiares de recursos) que serão usados pelo agente durante sua execução *a posteriori*. Detalhes adicionais sobre ambas abordagens podem ser encontrados em [Vaidyanathan et al. 2001b].

As duas abordagens tem sido implementadas [Vaidyanathan et al. 2001b] [Castelli et al. 2001], tendo cada uma sua aplicação de acordo com o problema de envelhecimento a ser tratado. Neste trabalho, não se realizou um estudo exaustivo a cerca da melhor alternativa, mas sim de uma implementação que permitisse validar o conceito de rejuvenescimento aplicado ao caso de envelhecimento identificado

[8] Neste contexto, o termo "sistema" se refere ao componente (recursos do sistema operacional, processos de software, dentre outros) sujeito aos efeitos do envelhecimento.

anteriormente (experimento E8). Portanto, definiu-se como enfoque adotado para esta implementação o *closed-loop* com a monitoração sendo realizada na forma *on-line*. Definiu-se como política de rejuvenescimento adotada foi a monitoração do crescimento dos processos *httpd*, tendo como limiar, um valor médio de 450 Mbytes, obtido a partir dos resultados do experimento E8. Quando os processos ultrapassam este limite, o mecanismo de rejuvenescimento é executado.

A decisão em se utilizar o tamanho dos processos *httpd* e não o valor da memória total disponível e/ou utilizada, como sugerido em [Li et al. 2002], tem como base evitar a execução do rejuvenescimento em situações consideradas "*alarmes falsos*". Baseando-se na memória total (utilizada ou disponível), bem como em outros recursos de uso compartilhado (ex. *swap* utilizado), cenários como aqueles verificados em E1 e E2 podem executar o rejuvenescimento desnecessariamente. Naqueles dois experimentos, verificou-se que o tamanho da memória decrementava gradativamente, porém não devido ao envelhecimento dos processos *httpd*, mas sim por um crescimento contínuo do *buffer-cache*. Naqueles dois casos, aplicar o rejuvenescimento nos processos servidores não resolveria a questão da redução da memória disponível. Uma solução seria a utilização de um *ARS* híbrido, rejuvenescendo não só os processos *httpd*, mas também componentes do sistema operacional. A seguir serão apresentados detalhes do mecanismo de rejuvenescimento utilizado.

4.1.1 Mecanismo de Rejuvenescimento

Tendo definida a estratégia de coleta e análise de dados, modo de operação e os limiares para o seu acionamento, faz-se necessário definir como será realizado o rejuvenescimento dos processos. Neste caso, entender o conceito de *granularidade* do rejuvenescimento se faz necessário, visto estar este intrinsecamente associado à ação que deve ser executada pelo ARS sobre o sistema envelhecido. Basicamente, tem-se dois níveis de atuação, que são o rejuvenescimento parcial e total. No rejuvenescimento total, o sistema será reinicializado por completo (ex. reinicialização do equipamento), o que sempre implica em um tempo de inatividade (*downtime*) do serviço prestado. No rejuvenescimento parcial, apenas parte do sistema deve ser reiniciada, sendo que o tempo de inatividade nestes casos é muito dependente da forma como os componentes reinicializados estão organizados. Como exemplo, apenas alguns processos, dentre um conjunto, poderiam ser reinicializados reduzindo assim a capacidade de atendimento do serviço prestado, porém evitando a inatividade total durante o rejuvenescimento.

Como se pode verificar, a definição da abrangência (*granularidade*) e de como o rejuvenescimento será realizado deve ser o resultado de um estudo minucioso da arquitetura do sistema ao qual o rejuvenescimento será aplicado. Assim sendo, estudou-se a organização dos componentes de software que fazem parte do Apache, a fim de se definir a melhor forma para aplicação do rejuvenescimento.

No caso do servidor Apache, sua arquitetura é baseada em um processo mestre, que controla diversos outros processos servidores. O mestre não atende requisições HTTP, sendo apenas responsável pela monitoração dos processos servidores, criando e removendo-os de acordo com configurações pré-definidas pelo usuário. A comunicação entre o mestre e os processos servidores ocorre através de uma área de memória compartilhada, a qual é usada para fins de monitoração dentre outros propósitos. Os

processos servidores por sua vez ficam dedicados à realização do processamento das requisições HTTP que chegam dos clientes. A Figura 8 ilustra esta arquitetura.

Vale ressaltar, que na versão 2.0 o Apache introduz novas formas de organizar seus processos servidores. Uma delas é a utilização de diversas *threads* por processos, onde cada *thread* fica responsável por atender requisições HTTP. Neste trabalho, considerou-se a configuração sem *threads* (módulo *prefork*), a fim de facilitar a monitoração dos processos de forma individual.

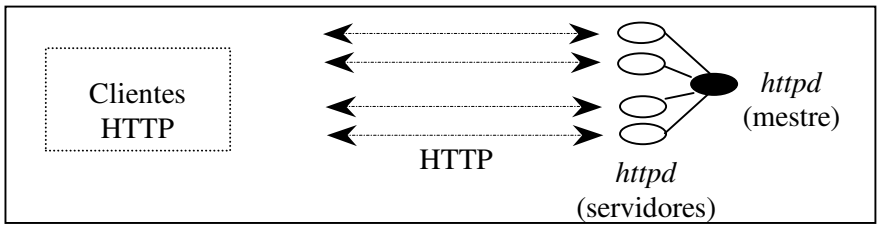

Figura 8. Visão Geral dos Processos no Apache

Na busca de implementar um mecanismo de rejuvenescimento aplicado ao Apache, foram estudadas as suas configurações e identificou-se algumas que poderiam ser usadas para este propósito. As principais foram *MaxRequestperChild e MaxKeepAliveRequests*. A diretiva *MaxRequestperChild* permite configurar um número máximo de requisições por processo servidor, a fim de substituí-lo quando o mesmo alcança este limite de requisições processadas. Outra diretiva com o mesmo propósito é a *MaxKeepAliveRequests*, contudo considera-se neste caso requisições realizadadas através de conexões persistentes, uma característica específica da versão 1.1 do protocolo HTTP.

Uma limitação ao se utilizar às opções citadas anteriormente, se refere à estimativa destes valores. Como são baseadas no número de requisições atendidas, é uma tarefa complexa encontrar um valor adequado para o limite de requisições a ser processado, que indique situações de envelhecimento acima deste valor. Esta complexidade existe, pois que não se tem verificado, em nenhum dos trabalhos pesquisados, indícios de comportamento determinístico para o envelhecimento de software. Além disso, a escolha de valores reduzidos, como forma de prevenção, causará uma maior freqüência nas reinicializações dos processos servidores, introduzindo um custo extra de processamento. Ao contrário, selecionar valores elevados para se evitar este custo de processamento aumenta o risco dos processos envelhecerem antes de sua reciclagem, ocasionando problemas semelhantes aos verificados no experimento E8. Deste modo, descartou-se a possibilidade de usar estas diretivas de configuração, buscando novas alternativas para a implementação do *ARS*.

Buscando novas alternativas e considerando a arquitetura do Apache, identificou-se três possibilidades de implementação do rejuvenescimento. A primeira, que foi chamada *parcial seletiva*, monitora todos os processos e rejuvenesce apenas aqueles cujo tamanho (memória alocada) representam maior contribuição dentre os demais. Uma segunda proposta, denominada de *reinicialização completa*, realiza a parada do serviço como um todo, finalizando e reinicializando todos os processos *httpd de* uma só vez. Estudando suas implicações, ambas introduzem períodos de

inatividades, principalmente no caso da segunda proposta. Além disso, o primeiro caso, preferido pelo menor impacto em termos de inatividade do serviço, possui elevada complexidade em sua implementação. Isto, devido à necessidade de se conhecer o estado do processo, a respeito deste estar com não atendendo requisições naquele instante, a fim de se evitar o fechamento abrupto de uma sessão HTTP em curso. Uma das formas verificadas para realizar tal implementação seria modificar o código do *httpd* mestre, que monitora os demais servidores, obtendo assim esta informação de estado.

A fim de evitar modificações no código do servidor, buscou-se uma solução menos intrusiva e ao mesmo tempo que reduzisse, tanto quanto possível, o tempo de inatividade durante a realização do rejuvenescimento. Esta alternativa foi encontrada, demonstrando desempenho superior às três consideradas até o momento. De fato, esta solução utiliza uma característica do próprio Apache, implementada desde as primeiras versões do software. O Apache, nas versões para UNIX/Linux[9], implementa o tratamento de diversos sinais (*interrupções de software*), para propósitos gerais variados. No caso do Apache para Unix/Linux, utilizado neste trabalho, um sinal implementado é o *SIGUSR1*. Este, ao ser enviado para o processo *httpd* mestre, diz para ele finalizar cada um dos processos servidores, porém somente quando estes terminarem de atender a atual requisição em progresso. Caso existam processos em estado de espera, os mesmos são reinicializados imediatamente. Esta característica resolve o problema de complexidade, citado anteriormente, no que tocante a se determinar quando um processo selecionado não está mais atendendo requisições. Além disso, existe uma redução significativa no tempo de inatividade, pois não se tem a parada completa do serviço, como na proposta de *reinicialização completa*. Neste caso, o *httpd* mestre continua executando, o que garante o estabelecimento de novas conexões na porta 80, mesmo durante aquelas situações em que todos os processos estão sendo rejuvenescidos. Isto ocorre, pois a porta padrão do serviço continua aberta e, portanto, com uma fila de entrada associada em nível de *kernel* (*tcp backlog*).

O aproveitamento desta característica, presente no código do próprio Apache, ofereceu a melhor alternativa dentre as consideradas até o momento. Esta conclusão se deve não somente pelo benefício na redução do tempo de inatividade, mas também por oferecer uma forma padronizada de comunicação entre o *ARS* e o Apache, que pode ser usada em qualquer versão deste servidor. A seguir, serão apresentados os resultados desta implementação.

4.1.2 Avaliação do *ARS*

Foram utilizados dois cenários para validar a utilização do *ARS* implementado, integrado ao servidor web Apache. O primeiro, idêntico ao utilizado no experimento E8, visou comparar os resultados do mesmo experimento sem e com o rejuvenescimento. No segundo cenário, buscou-se criar uma condição de estresse para o ambiente operacional do servidor, com a duplicação do número de processos servidores, resultando em um total de 400 processos. Os resultados obtidos em ambos cenários estão ilustrados a seguir.

---

[9] Funcionalidades equivalentes existem em versões para outras plataformas.

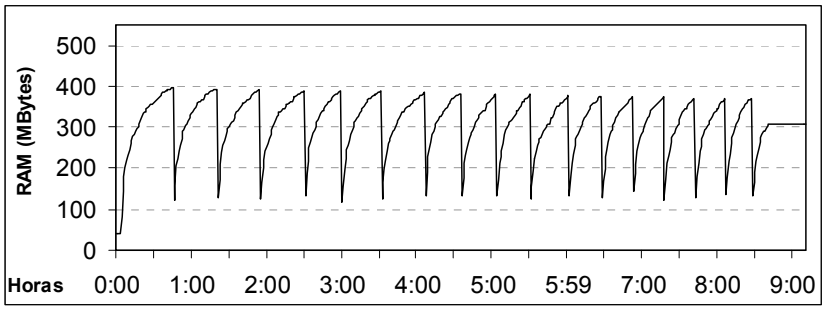

Figura 9. Visão Geral dos Processos no Apache

Na Figura 9, tem-se o mesmo cenário de carga de trabalho daquele utilizado no experimento E8. Pode-se observar que o *ARS* rejuvenesce os processos quando estes ultrapassam o limiar de 395 Mbytes, definido para o teste. A monitoração feita pelo *ARS* ocorre a cada 60 segundos, o que ocasiona pequenas variações no que se refere ao momento de execução do rejuvenescimento.Verificou-se que o desvio padrão em relação ao limniar é de ± 5,386 Kbytes, ou seja, com a periodicidade de 60 segundos tem-se uma boa precisão em termos de controle no avanço do envelhecimento. A necessidade de se aplicar o rejuvenescimento foi em média 30 minutos, o que se demonstra compatível com um cenário de carga de trabalho em torno de 90% da capacidade efetiva do servidor.

Outro aspecto importante, foi o reflexo do rejuvenescimento no tempo de resposta do servidor. No experimento E8, devido ao processo de *swapping*, várias conexões foram encerradas por *timeout*. Na realização deste experimento com o rejuvenescimento, o tempo de resposta do servidor se manteve constante, como pode ser verificado na Figuras 10.

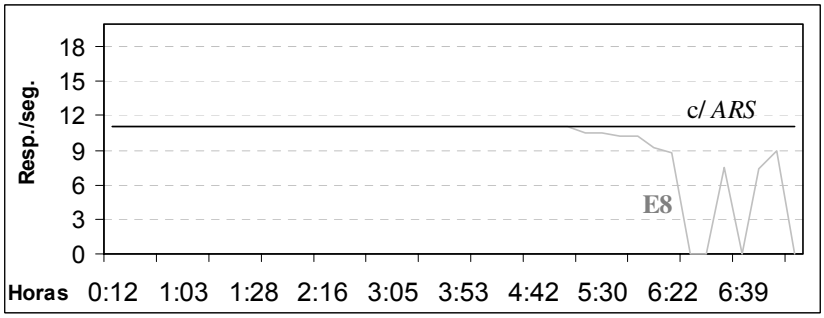

Figura 10. Taxa de Resposta c/ e sem *ARS*.

Um segundo cenário avaliado foi para uma situação de estresse. Foram utilizados 400 processos, com uma taxa de requisição próxima de 100% da máxima suportada pelo servidor. Além disso, utilizou-se a criação dinâmica de páginas com tamanho de 2 MBytes. As Figuras 11 e 12 ilustram este cenário sem e com o *ARS*.

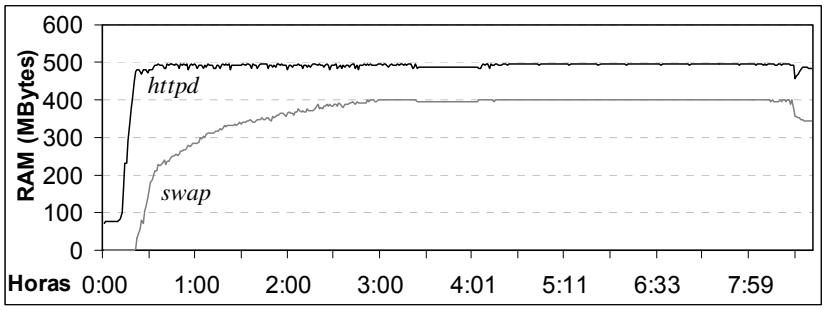

Figura 11. Condição de estresse sem o *ARS*.

Devido à condição de estresse, ocorreu uma aceleração no consumo de memória dos processos *httpd*, o que exigiu a utilizaçãodo, prematura (a partir dos 20 minutos iniciais), do espaço em disco reservado para *swap.* Na configuração utilizada, o espaço reservado para *swap* foi de 400 MBytes, o qual foi plenamente utilizado a partir das 3 primeiras horas de teste. Isto demonstra claramente que os processos *httpd*, mesmo atingindo o limite de 500 MBytes da memória principal, continuaram crescendo até a saturação do *swap*. Verificou-se nos registros do httperf, que a partir dos 25 minutos de teste diversas conexões foram encerradas por *timeout*.

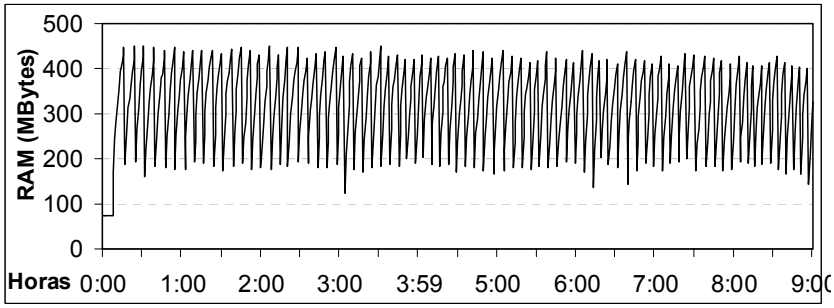

Figura 12. Condição de estresse sem o *ARS*.

Os resultados do teste em condições de estresse, porém utilizandoo-se o ARS, etão apresentados na Figura 12. Para este caso, considerou um limiar de 460 MBytes para o acionamento do mecanismo de rejuvenescimento. Com relação a taxa de resposta do servidor, utilizando o *ARS*, a mesma se manteve constante, com o mesmo comportamento demonstrado na Figura 10.

5. Conclusões

Este trabalho apresentou um estudo experimental sobre o envelhecimento de software em servidores web, bem como a implementação, avaliação e validação de um agente de rejuvenescimento implementado para servidores Apache.

Previamente à seleção do enfoque adotado para a construção do agente, foi realizado um projeto de experimentos, do tipo fatorial completo, a fim de caracterizar o comportamento do envelhecimento no ambiente utilizado neste estudo. A partir dos resultados obtidos neste projeto experimental comprovou-se a existência do envelhecimento como também reportado em [Li et al. 2002]. Complementarmente identificou-se, através do método da tabela de sinais, as condições (fatores, níveis e combinações) que mais contribuem para a ocorrência do envelhecimento no Apache.

De acordo com os resultados dos experimentos (E5-E8), verificou-se que, independente do tamanho da página, sempre ocorreu um envelhecimento precoce dos processos servidores. A intensidade deste envelhecimento, nestes casos, foi influenciada pelo tamanho da página. A partir destes resultados, um ARS foi implementado para o Apache, seguindo uma abordagem do tipo *closed-loop*. A principal dificuldade nesta fase foi definir um mecanismo de rejuvenescimento que causasse menor impacto na disponibilidade do serviço prestado. Este requisito foi alcançado, através de uma funcionalidade do próprio Apache (*SIGUSR1*), a qual permitiu o rejuvenescimento dos processos servidores sem desabilitar o recebimento de novas conexões durante a fase de rejuvenescimento.

A efetividade do ARS no controle do tamanho dos processos demonstrou ser satisfatória e promissora, tendo em vista que o seu principal propósito, evitar a exaustão do recurso memória, foi alcançado. Além disso, outro benefício direto do rejuvenescimento foi à manutenção da taxa de resposta do servidor. Verificou-se no experimento E8 que a mesma foi negativamente influenciada pelo envelhecimento dos processos servidores, e no cenário com a presença do ARS, esta se manteve constante durante todo o período de execução.

Trabalhos complementares ao estudo apresentado estão em andamento, no sentido de se propor metodologias, ferramentas e modelos de simulação, que facilitem o processo de identificação e caracterização do envelhecimento em aplicações susceptíveis a este fenômeno. Dentre tais propostas, estão em andamento pesquisas para a realização de ensaios acelerados de forma a se estudar os efeitos do envelhecimento de software em menor tempo do que é atualmente empregado. Além disso, trabalhos empregando técnicas de análise de risco e simulação computacional tem sido estudados a fim de avaliar suas aplicações no contexto dos agentes de rejuvenescimento de software.

6. Referências

Avritzer, A. e Weyuker, E. J. (1997) "A Monitoring Smoothly Degradating Systems for Increased Dependability", In: Empirical Software Engineering Journal, Vol. 2, Nº 1, p. 59-77.

Castelli, V., Harper, R. E., Heidelberger, P., Hunter, S. W., Trivedi, K. S. , Vaidyanathan , K. e Zeggert, W. P. (2001) "Proactive Management of Software Aging", IBM Journal of Research & Development, Vol. 45, No. 2, March 2001.

Garg, S., Moorsel, A., Vaidyanathan, K. e Trivedi, K. (1998a) "A Methodology for Detection and Estimation of Software Aging", In: Proceedings of the 9th International Symposium on Software Reliability Engineering ,p. 282-292, Paderborn, Alemanha.

Garg, S., Puliafito, A., Telek, M. e Trivedi, K. (1998b) "Analysis of Preventive Maintenance in Transactions Based Software Systems", In: IEEE Transactions on Computers 47(1), p. 96-107.

Gray, J. (1986) "Why Do Computers Stop and What Can Be Done About It?", In: Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems", p. 3-12, IEEE Computer Society Press, Los Angeles, California, USA.

Huang, Y., Kintala, C.,Kolettis e N., Fulton, N. D. (1995) “Software Rejuvenation: Analysis, Module and Applications”, In: Proceedings of the 25th Symposium on Fault Tolerant Computer Systems, p. 381-390, Pasadena, CA.

Jain, R. “The Art of Computer Systems Performance Analysis: *Techniques for Experimental Design, Measurement, Simulation, and Modeling*.”, John Wiley & Sons, 1991.

Kraus, I. F. (2000) “Agent Technology in Performance and Availability Management”, 26th International Computer Measurement Group Conference, p. 139-152, Orlando, FL, USA.

Li, L., Vaidyanathan, K. e Trivedi, K. S. (2002) “An Approach for Estimation of Software Aging in a Web Server”, International Symposium on Empirical Software Engineering, p. 91-100, Nara, Japan.

Mosberger, D. e Jin, T. (1998) “Httperf – A Tool for Measuring Web Server Performance”, In First Workshop on Internet Server Performance, Madison, WI.

Netcraft (2004) “Web Server Survey Archive”, http://news.netcraft.com/ archives/ web_ server_survey.html, Março.

Shereshevsky, M., Cukic, B., Crowel , J., Gandikota , V. e Liu, Y. (2003) "Software Aging and Multifractality of Memory Resources", The International Conference on Dependable Systems and Networks (DSN-2003),p. 721-730, San Francisco, CA,USA.

Trivedi, K. S., Vaidyanathan, K. e GosevaPopstojanova, K. (2000) “Modeling and Analysis of Software Aging and Rejuvenation”, In: Proceedings of the 33rd Annual Simulation Symposium, p. 270-279, Los Alamitos, CA, IEEE Computer Society Press.

Vaidyanathan, K. e Trivedi, K. S. (1998) “A Measurement-based Model for Estimation of Resource Exhaustion in Operational Software Systems”, In: Proceedings of the Tenth IEEE International Symposium on Software Reliability Engineering, p. 84–93,Boca Raton, FL,USA.

Vaidyanathan, K., Harper, E. S., Hunter , W. e Trivedi., K. S. (2001a) “Analysis and Implementation of Software Rejuvenation in Cluster Systems”, ACM SIGMETRICS 2001/Performance 2001, Cambridge, MA, USA.

Vaidyanathan, K. e Trivedi, K. S. (2001b) “Extended Classification of Software Faults Based on Aging”, *Fast Abstracts*, In: Proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong.