Benchmark para análise comportamental do sistema de memória virtual do Linux

Rúbens Antônio Rodrigues[2], Lourenço Alves Pereira Júnior[1,2], Regina H. C. Santana[1], Marcos José Santana[1], Francisco José Monaco[1]

[1] Universidade de São Paulo — USP

{ljr,rcs,mjs,monaco}@icmc.usp.br

[2]Instituto Federal de São Paulo — IFSP

rubensan.rodrigues@gmail.com, ljr@ifsp.edu.br

Abstract. *Knowing the operating system performance is important to specify the operational environment of computing services. System's performance may vary when exposed to different workloads, in a controlled experiment it can be exposed to different levels of conditions. Specifically, in the case of Linux's virtual memory management, severe workloads can lead to performance degradation and thrashing. Although, it is possible to notice the occurrence of these phenomena, a tool allowing the conduction of controlled experiments intended to produce empirical data to fit mathematical models to represent the virtual memory system Linux is scarce. This paper presents (1) the proposal of a benchmark for this type of application and (2) results that demonstrate its operation.*

Resumo. *Conhecer o desempenho do sistema operacional é importante para o projeto e especificação do ambiente de implantação de serviços computacionais. O desempenho pode variar quando exposto a diferentes cargas de trabalho, em ambiente controlado pode-se excitar o sistema em estudo em diferentes níveis de exigência computacional. Especificamente, no caso do sistema de memória virtual do Linux, pode haver degradação do desempenho e até mesmo sua inutilização no caso de* thrashing. *Muito embora, seja possível perceber a ocorrência desses fenômenos, é muito escassa a existência de uma ferramenta que permita a condução de experimentos controlados a fim de produzir dados empíricos para estimação de modelos matemáticos que representam o sistema de memória virtual do Linux. Este trabalho apresenta (1) a proposta de um* benchmark *para este tipo de aplicação e (2) resultados que demonstram seu funcionamento.*

1. Introdução

O desempenho de um sistema operacional é sensível ao tipo de carga de trabalho na qual ele é exposto, pois cargas mais severas podem comprometer o desempenho de todas aplicações e serviços sob seu domínio. Nesse contexto, o sistema gerenciador de memória possui um papel importante, porque flutuações que extrapolem a quantidade de memória física podem levar o sistema a uma condição conhecida como *thrashing*, cujo resultado é a completa inutilização do sistema [Denning 1968]. Uma representação matemática do sistema que possibilite análise e previsão do comportamento de um sistema

é algo importante e pode servir como ferramenta para o planejamento de capacidade de sistemas computacionais. Dependendo do tipo do modelo, sua utilização pode trazer benefícios quando simulados diferentes tipos de carga de trabalho, impactando no projeto e implantação de sistemas mais confiáveis e resilientes [Hellerstein et al. 2004].

O Linux é um sistema operacional de código aberto amplamente utilizado. Possui um sistema de memória virtual baseado em páginas, em que é possível alocar uma quantidade de memória maior do que a quantidade fisicamente disponível. Assim como outros sistemas operacionais, o seu sistema de memória pode ficar comprometido quando exposto a cargas de trabalho mais severas.

O objetivo deste artigo é apresentar uma proposta de um *benchmark* cujas funcionalidades auxiliam na análise comportamental do sistema de memória do Linux de acordo uma carga sintética em um ambiente controlado, identificando as responsabilidades dos componentes que compõem o *benchmark* e apresentando resultados empíricos obtidos através da execução em ambiente controlado de um protótipo. Destaca-se como benefício a possibilidade de reprodução dos experimentos conduzidos com o *benchmark* proposto. Diferentes versões do *kernel* do Linux, podem apresentar comportamentos diferentes, o que justifica o uso deste *benchmark* para um estudo direcionado a versão desejada. Como a memória virtual utiliza uma área do disco, a análise do comportamento do sistemas configurados com *Redundant Array of Independent Disks* — RAID e *Solid-state Driver* — SSD pode ser conduzida. Outro uso para este *benchmark* diz respeito a avaliação de infraestruturas de implantação com presença de concorrência por recursos, por exemplo um ambiente virtualizado (várias máquinas virtuais hospedadas em uma máquina física).

O diferencial deste trabalho em relação a outros relacionados [Carneiro et al. 2009, Terraneo and Leva 2013, Santos and Maziero 2009, Maziero et al. 2013] é que este é direcionada a experimentação em ambiente controlado com reprodução de experimentos e foco na produção de dados para análise e modelagem matemática.

2. Projeto do *benchmark*

Os experimentos para a geração de um modelo capaz de representar o sistema de memória do Linux podem ser feitos através do método caixa preta. Para isso, são especificados dois conjuntos de variáveis um para entrada e outro para saída, alteram-se os valores das variáveis de entrada com base em faixas determinadas e observam-se os valores das variáveis de saída. A identificação das variáveis e das faixas nem sempre ocorre de forma clara, algumas variáveis podem ser especificadas de forma empírica.

Para excitar de forma controlada o sistema de memória, foi desenvolvido um modelo de *benchmark* tendo como base o algorítimo utilizado por [Santos and Maziero 2009]. Nesse modelo, são criados três programas: (1) o gerenciador responsável por receber parâmetros de execução controlada de experimentos, (2) o monitor responsável por gerar os traços de execução dos experimentos e (3) o gerador de carga de trabalho que aloca uma área de memória e efetua operações de leitura e escrita. O intuito do presente *benchmark* é impor uma carga de trabalho controlada ao sistema de memória do Linux, executando operações de leitura e escrita em endereços de memória pseudo-aleatórios e obter os estados das variáveis de saída. A execução dos experimentos exige replicação, e por isso, foi idealizado um modelo de implantação em um gerenciador

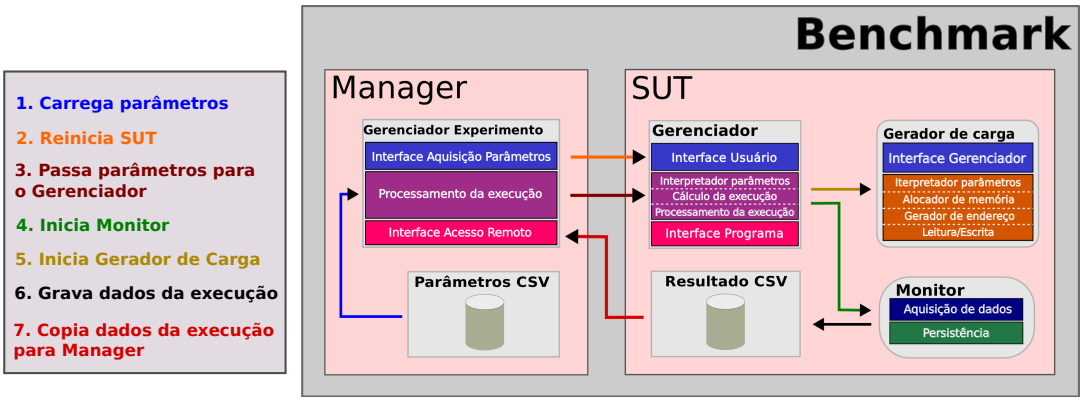

Figura 1. Distribuição dos programas do *benchmark*. Os dois blocos principais do *Benchmark* (`Manager` e `SUT`) representam duas máquinas, uma para gerenciar o experimento e outra com o sistema em teste. O `Manager` é constituído pelo Gerenciador de Experimentos e os Parâmetros em uma base de dados CSV. Já o `SUT` é composto pelo Gerenciador, Gerador de Carga, Monitor e Resultados (bases de dados CSV). As setas coloridas representam o fluxo da execução de experimentos, conforme legenda.

(*Manager*) controla externamente a execução do sistema em teste (*System Under Test* — SUT).

Os componentes do *benchmark* estão ilustrados na Figura 1. O bloco principal denominado `Benchmark`, é composto por dois blocos que representam duas máquinas. Uma denominada `Manager` responsável por gerenciar a execução de experimentos e outra denominada `SUT` que contém o sistema a ser testado. Dentro do `Manager` está o `Gerenciado de Experimentos`, que é um programa responsável por carregar um conjunto de parâmetros (fatores) e conduzir o experimento de acordo com eles e uma base de dados (Parâmetros CSV) constituída com os possíveis valores das variáveis de entrada. O `SUT` contém o Gerenciador, que tem a função de: (1) receber e interpretar os parâmetros; (2) iniciar o `Monitor` para gravar em uma base de dados (Resultados CSV) os valores amostrados periodicamente das variáveis de saída; (3) e iniciar de forma parametrizada um conjunto de n geradores de carga que alocam uma regiões de memória que serão utilizadas para operações de leitura/escrita. As setas coloridas relacionam-se com a legenda e demonstram o fluxo de execução de experimentos. Seta azul: são carregados os parâmetros para execução do experimento; seta laranja: a máquina com o `SUT` é reiniciada; seta marrom: são passados os parâmetros para o Gerenciador no `SUT`. Ele processa os parâmetros e na seta verde inicia o Monitor e depois na dourada parametriza e inicia geradores de carga; seta preta: os dados coletados pelo monitor são gravados na base de dados (Resultados CSV). E ao fim do experimento, seta vermelha, os dados são copiados do `SUT` para `Manager`.

3. Experimentos e resultados

Os resultados da execução são apresentados nos gráficos da Figura 2 (os valores representam as médias dos valores obtidos das execuções das réplicas). Foram relacionadas a utilização da CPU e a alocação de memória com a movimentação de páginas entre a memória e o disco nas operações de *page-in* e *page-out*. Como pode ser observado nos gráficos, as taxas de entrada e de saída de páginas são assimétricas, a taxa de entrada é

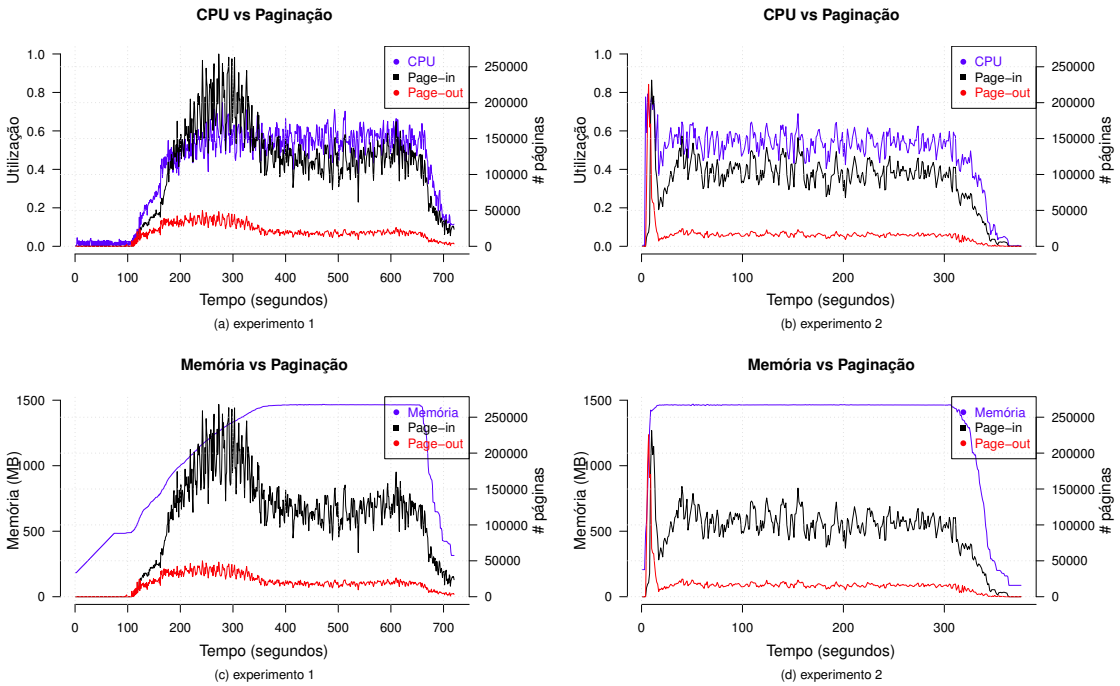

Figura 2. Resutados

maior que a de saída, esse comportamento pode ser produto da política de substituição de páginas do Linux onde são reescritas no disco apenas as páginas que foram modificadas.

A CPU é influenciada pela entrada de páginas (Figura 2a), uma vez que ocorre um aumento da utilização logo após o sistema começar a fazer paginação, tal comportamento também pode ser atribuído ao algoritmo de substituição de páginas, porém é necessário realizar um estudo direcionado a fim de evidenciar esse fato. No Figura 2b, ocorre um pico seguido de uma oscilação e, por volta dos 50 segundos, volta ao regime estacionário. Quando relacionadas as Figuras 2a e 2b, nota-se que a taxa de entrada de páginas apresenta uma diferença de aproximadamente $11,1\%$. Essa diferença pode ocorre pelos motivos: (1) menor grau de paralelismo, pois não haviam operações de leitura/escrita durante a criação dos processos; (2) pelo mecanismo de carregamento de programas do Linux que apenas mapeia as páginas de memória para o arquivo no disco deixando para fazer sua carga para a memória quando o processo demandar; (3) na Figura 2b, devido ao fato de todos processos do experimento começarem a execução quase simultaneamente, o algoritmo tende ter acesso a informação de quais páginas devem ser alocadas, fazendo com que a escolha seja mais certeira de que na Figura 2a, caso em que uma página que seria a melhor escolha no instante de tempo t pode acontecer no futuro $(t + \Delta t)$, portanto indisponível.

Analisando a Figura 2c é visível uma grande diferença entre as atividades de *page-in* e *page-out*, aproximadamente 90%, que pode ser atribuída novamente a política de substituição de páginas. Na atividade de alocação de memória, entre os instantes 75 e 105, ocorre uma parada que pode ter sido ocasionada pelo escalonador de entrada e saída que tem a finalidade de minimizar a movimentação das cabeças do disco, a atividade de paginação que se iniciou no instante 105 dá suporte a essa afirmação. Na Figura 2d entre

os instantes 2 e 15, ocorre um pico de *page-out* provocado pela alta demanda de páginas que foi gerada a partir instante de tempo 1 momento em que há uma rajada de processos e provoca muitas faltas de páginas, portanto a quantidade de páginas transferidas para o disco nesse momento é alta e aproximadamente no instante 35 o comportamento fica similar aos dos outros ensaios.

4. Conclusões

Foi proposto o protótipo de um *benchmark* com a finalidade de auxiliar a produção de dados para a análise comportamental do sistema de memória do Linux em um ambiente controlado. Cada componente tem suas responsabilidades bem definidas. Idealmente, eles devem ser distribuídos em máquinas distintas, um com o sistema em teste e outro para gerenciar a execução do experimento. A execução de experimentos é automatizada e definida por um conjunto de fatores que descrevem a execução dos experimentos. Ao término da execução dos experimentos, os dados estarão disponíveis. Trabalhos futuros contemplam a execução de mais experimentos de forma que seja possível criar um modelo que seja útil para implantação de sistemas de execução online, possibilitando a criação de políticas de admissão, bem como políticas para gerenciamento de memória em sistemas operacionais. O *benchmark* pode ser estendido para diferentes funções de excitação (PRBS, senóide etc.). Pode ser implementado para outros sistemas operacionais como FreeBSD, OpenBSD, Solaris, MacOS etc.

Agradecimentos

Os autores agradecem ao IFSP câmpus Araraquara, à CAPES, ao CNPq e ao NAPSoL-ICMC/USP pelo apoio durante a execução deste trabalho.

Referências

Carneiro, I. S. C., Barreto, L. P., and Sá, A. S. d. (2009). Aplicação e análise de teoria de controle realimentado no tratamento de faltas de páginas em sistemas de gerenciamento de memória. In *VI Workshop de Sistemas Operacionais*. Bento Gonçalves, RS.

Denning, P. J. (1968). Thrashing: Its causes and prevention. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 915–922, New York, NY, USA. ACM.

Hellerstein, J. L., Diao, Y., Parekh, S., and Tilbury, D. M. (2004). *Feedback Control of Computing Systems*. John Wiley & Sons.

Maziero, C., dos Santos, D., and Santin, A. (2013). Evaluation of desktop operating systems under thrashing conditions. *Journal of the Brazilian Computer Society*, 19(1):29–42.

Santos, D. and Maziero, C. (2009). Avaliação do comportamento de sistemas operacionais de mercado em situação de *thrashing*. In *VI Workshop de Sistemas Operacionais*. Bento Gonçalves, RS.

Terraneo, F. and Leva, A. (2013). Feedback-based memory management with active swap-in. In *Control Conference (ECC), 2013 European*, pages 620–625.