Investigação dos Efeitos do Envelhecimento de *Software* na Plataforma Docker

Felipe Oliveira[1], **Luan Lins**[1], **André Barreto**[1], **Jean Araújo**[1]

[1]Unidade Acadêmica de Garanhuns - UAG
Universidade Federal Rural de Pernambuco - UFRPE

{flpdias14, luancsl95, barretorodrigues16}@gmail.com

jean.teixeira@ufrpe.br

Abstract. *Docker is one of the platforms for the creation and management of containers with wide use in the market. A problem that affects availability is the phenomenon of software aging, an unavoidable process, where the application processes suffer degradation of performance throughout their use. This study aims to monitor and evaluate the performance of the Docker platform in the context of the cloud computing paradigm, including possible software aging effects. We conducted an experimental study with a workload to simulate the life cycle of containers, while the system monitoring was performed. The results show high resource consumption as RAM and CPU usage in the network utility of operating system (OS) , in addition to memory fragmentation an important subprocess of the platform.*

Resumo. *O Docker é uma das plataformas para criação e gerenciamento de contêineres com ampla utilização no mercado. Um problema que afeta a disponibilidade é o fenômeno do envelhecimento de software, um processo inevitável, onde os processos de aplicações sofrem degradação de desempenho ao longo de sua utilização. Este estudo objetiva monitorar e avaliar o desempenho da plataforma Docker no contexto do paradigma de computação em nuvem, incluindo possıveis efeitos de envelhecimento de software. Realizou-se um estudo experimental, com uma carga de trabalho simulando o ciclo de vida de contêineres, enquanto o monitoramento do sistema era realizado. Os resultados mostram alto consumo de recursos como memória RAM e uso de CPU em utilitário de rede do sistema operacional (SO), além de fragmentação de memória em subprocesso importante da plataforma.*

1. Introdução

Em diversos setores da sociedade os *softwares* tornam-se cada vez mais presentes e devido a crescente complexidade dos softwares, é cada vez mais difícil produzir softwares que não contenham *bugs*. Uma falha em um sistema que controla recursos importantes pode causar impactos considerados catastróficos, acarretando em perdas financeiras, destruição de reputações ou até mesmo perda de vidas [Grottke et al. 2008].

A implantação de arquiteturas baseadas em computação em nuvem (*cloud computing*) tem crescido nos últimos anos, já que constituem uma plataforma escalável, robusta

e de baixo custo. A tecnologia chave para a computação em nuvem é a virtualização. Essa técnica possibilita a criação de ambientes virtuais de hardware, simulando servidores físicos. Isso permite que os recursos do servidor sejam melhor alocados.

A dependabilidade é "a propriedade do sistema que impede que este falhe de uma forma inesperada ou catastrófica" [Avizienis et al. 2004]. A disponibilidade e a confiabilidade de serviços hospedados em uma nuvem podem ser afetadas por diferentes problemas. O fenômeno do envelhecimento de software é um processo inevitável, onde as aplicações sofrem degradação de desempenho ao longo de sua utilização, aumentando sua probabilidade de indisponibilidade.

Este estudo busca investigar os possíveis efeitos de envelhecimento de software em sistemas de gerenciamento de contêineres, como a plataforma Docker. Os contêineres são chamados de virtualizadores leves e são uma alternativa as máquinas virtuais convencionais, caracterizados pela baixa sobrecarga no sistema hospedeiro.

A organização deste artigo é descrita a seguir: a Seção 2 o método adotado é descrito; a Seção 3 descreve o estudo experimental realizado com a plataforma Docker e os resultados obtidos são discutidos; e por fim, na Seção 4 algumas conclusões e possíveis direcionamentos são realizados.

2. Metodologia

A Figura 1(a) mostra um diagrama da metodologia adotada. Ela consiste de 5 etapas, são elas: estudo do sistema, definição de métricas, planejamento da carga de trabalho, execução do experimento e análise de desempenho.

A primeira etapa consiste em entender o funcionamento do sistema a ser analisado. Na sequência, é realizada a definição da estratégia de monitoramento. Nessa etapa são definidas as métricas de interesse, como utilização de memória e CPU, e quais serão os processos monitorados. Os efeitos de envelhecimento podem demorar muito tempo para aparecer, caso ocorram. Para acelerar o processo, criou-se uma carga de trabalho para estressar o sistema.

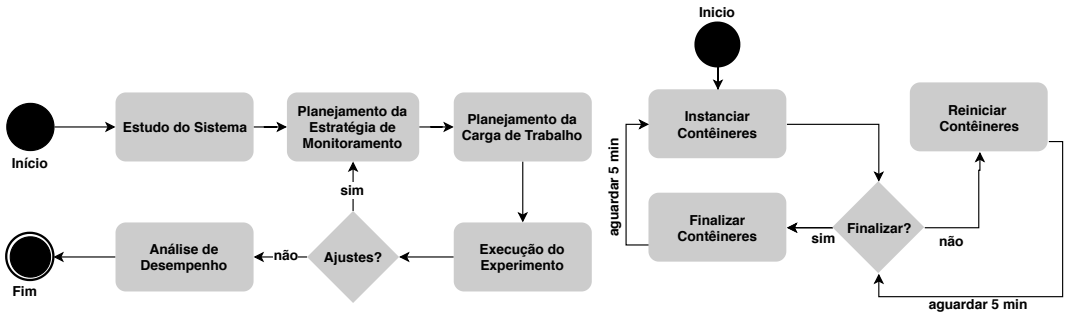
A etapa de execução do sistema consiste em monitorar o ambiente enquanto a carga de trabalho (ver Figura 1(b)) é aplicada. Em caso de necessidade, ajustes são realizados. Na etapa de análise de desempenho, os dados coletados são analisados e indícios de envelhecimento são investigados.

3. Estudo Experimental

3.1. Ambiente de Testes

Para este estudo o ambiente de teste foi composto por uma máquina HP Compaq 6005, com processador AMD Athlon II X2 220 de 2,8 GHz, 8 GB de memória principal e 8 GB adicionais de *swap*. O sistema operacional utilizado foi o Debian 9 *Stretch* (*kernel* 4.9.0-8-amd64), com o Docker versão 17.05.0-ce. Todos os contêineres são criados com uma imagem do servidor *web apache2*, com uma limitação de uso de memória de 256 MB.

A carga de trabalho utilizada foi pensada para simular as ações de uso do Docker. O diagrama mostrado na Figura 1(b) ilustra a carga de trabalho utilizada no experimento.



(a) Metodologia (b) Workload

Figura 1. Metodologia e Carga de Trabalho adotadas

A atividade de instanciar contêineres cria 20 contêineres através do comando *docker create* e os inicia com *docker start*. A cada 5 minutos é verificado se o tempo de execução dos contêineres é maior que 2 horas. Caso afirmativo, os contêineres são finalizados. Caso negativo, os contêineres são reiniciados (*docker restart*) e o contador T é incrementado. Na atividade Finalizar Contêineres realiza-se a parada (*docker stop*) e remoção (*docker rm*) de todos os contêineres. Após 5 minutos e um novo ciclo é iniciado.

Paralelamente a execução da carga de trabalho, o sistema era monitorado utilizando-se *scripts bash*. Esses *scripts* utilizam comandos utilitários do Linux, como *free, ps, df* e informações do arquivo */proc* [Blum 2008]. Foram monitorados recursos de interesse como CPU, uso de disco e utilização de memória de todo o sistema. Além disso, os consumos de processos específicos foram observados como Docker e NetworkManager[1] . Para minimizar interferências do monitoramento no sistema, os dados foram coletados a cada 60 segundos. Devido ao ciclo de vida efêmero dos contêineres, tempo de resposta dos servidores*web apache2* não foram coletados. Utilizando o *SystemTap*[2], foi verificada a ocorrência de fragmentação de memória.

Monitorou-se o *trace point mm_page_alloc_extfrag* que se refere a fragmentação de memória pelo *kernel* Linux. Uma alta ocorrência desse evento implicará que a memória está fragmentada e as alocações de alta prioridade poderão falhar no futuro[3] .

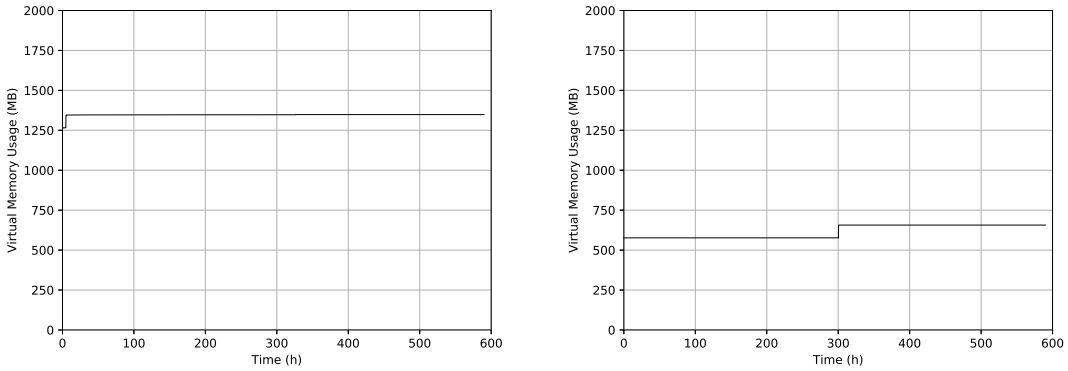
Ao fim de 25 dias de execução da carga de trabalho, o experimento foi encerrado. Os logs de dados foram coletados e deu-se início a etapa de análise de desempenho.

3.2. Análise dos Resultados

Constatou-se ao analisar os dados gerais do computador que há indícios de exaustão de recursos no experimento realizado.

Os processos relacionados ao Docker não apresentaram indícios de vazamento de memória. O consumo de recursos como CPU, uso de disco e número de *threads* não apresentaram variações significativas e, portanto, não serão exibidos. A utilização de memória (Figuras 2(a) e 2(b)) dois principais processos ficaram praticamente constantes durante toda a execução do experimento.

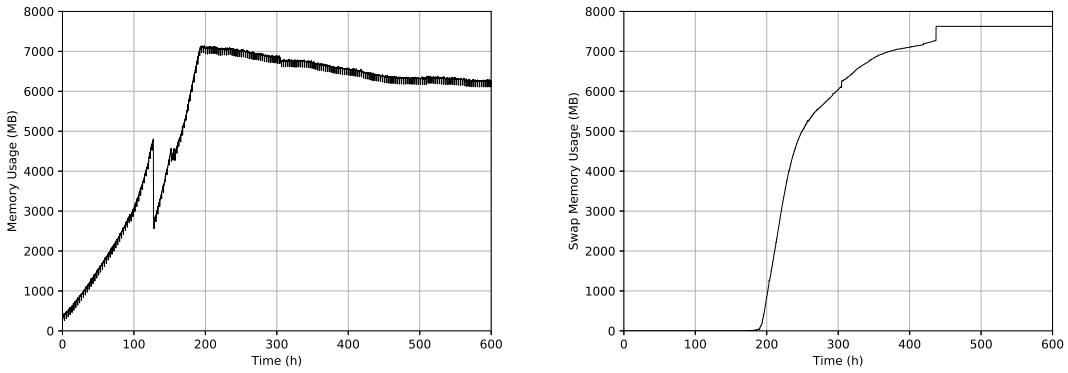
[1]https://developer.gnome.org/NetworkManager/stable/NetworkManager.html
[2]http://sourceware.org/systemtap
[3]https://www.kernel.org/doc/html/v4.18/trace/events-kmem.html



(a) Processo dockerd (b) Processo containerd

Figura 2. Utilização Memória Virtual Docker

As Figuras 3(a) e 3(b) mostram a utilização de memória RAM e *swap*, respectivamente. Verificou-se que houve esgotamento de memória ao longo da execução do experimento, com a utilização total de memória RAM e de memória *swap*. Vê-se na Figura 3(a) que ao fim de pouco mais de 200 horas de execução da carga de trabalho a memória utilizada no hospedeiro atinge seu ápice. Esse momento coincide com o início da utilização de memória *swap*, evidenciado na Figura 3(b).

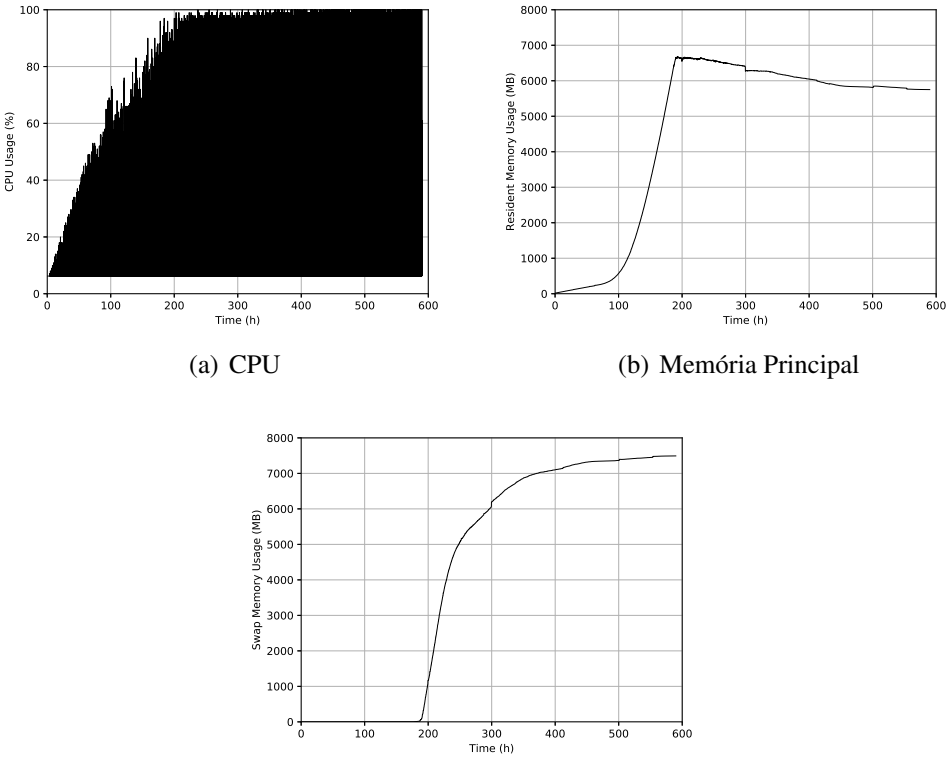


(a) Memória Principal (b) Memória Swap

Figura 3. Utilização Geral de Memória

Como a memória *swap* trata-se de uma área de disco rígido, é mais lenta que a memória RAM principal. Sendo o acesso a memória *swap* mais lento, o comportamento do sistema hospedeiro irá progressivamente tornar-se lento. O processo que provoca essa exaustão de recursos foi identificado e monitorado. O *NetworkManager* é um utilitário de gerenciamento automático de rede, realizando transições entre diferentes tipos de rede, buscando estabelecer a melhor conexão possível. A Figura 4(a) mostra a utilização de CPU pelo processo *NetworkManager*. Vê-se que a medida que o sistema se deteriora o consumo de CPU pelo processo aumenta. Por volta das 200 horas de experimento o processo atinge 100% de utilização. Esse momento ocorre no instante em que o SO começa a gerenciar memória *swap*.

As Figuras 4(b) e 4(b) mostram a utilização de memória residente e *swap* pelo processo *NetworkManager*. Constata-se que há vazamento de memória relacionada ao processo, já que o processo aloca memória de forma crescente. Esse consumo demasiado não está relacionado ao fenômeno da fragmentação já que como visto na Figura 5, o processo não apresenta mais que 11658 ocorrências de fragmentação ao longo do tempo observado.



(a) CPU

(b) Memória Principal

(c) Memória Swap

Figura 4. Utilização de Recursos pelo processo NetworkManager

As ocorrências de fragmentação de memória podem comprometer o bom funcionamento do sistema de duas formas [Macêdo et al. 2010]. Aumentando a quantidade de memória utilizada pelo processo, já que o SO pode atender requisições de alocação para novos blocos de memória mesmo havendo memória não contínua disponível. Essa memória não será liberada antes da finalização do processo. A outra, implica em complexidade adicional no gerenciamento de memória pelo SO. Na Figura 5 visualiza-se a acumulação de ocorrências de fragmentação de memória ao longo do tempo.

O processo *docker-runc* é responsável por fornecer um ambiente para execução de aplicativos em tempo de execução. Uma explicação para a não visualização dos efeitos dessa fragmentação é a natureza da carga de trabalho utilizada no estudo, já que o processo *docker-runc* está associado aos contêineres que são destruídos a cada 2 horas.

4. Considerações Finais

Neste trabalho foi realizada uma investigação dos efeitos do fenômeno envelhecimento de software na plataforma Docker. Verificou-se que há evidências de vazamento de memória

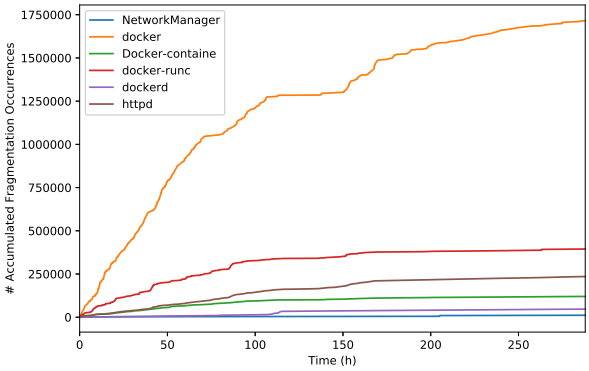


Figura 5. Fragmentação de Memória por processos

no processo *NetworkManager*. Este processo não possui relação direta com o funcionamento do Docker, mas o seu funcionamento junto a plataforma faz com que haja um aumento no seu consumo de memória. O vazamento de memória é um problema que não atinge somente o processo afetado, mas compromete a oferta de memória do sistema como um todo. Em algum momento, a oferta de memória livre se tornará insuficiente para que o Docker consiga instanciar novos contêineres, comprometendo a confiabilidade do sistema. Os resultados mostram grande ocorrência de eventos de fragmentação pelo processo *docker-runc*.Uma possível forma de verificar esses efeitos nesse processo é o planejamento de uma nova carga de trabalho, que utilize recursos que dependam desse processo e que não finalizem o contêiner periodicamente.

Agradecimentos

Gostaríamos de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq, e ao *UNAME Research Group* pelo apoio.

Referências

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33.

Blum, R. (2008). *Linux command line and shell scripting bible*, volume 481. John Wiley & Sons.

Grottke, M., Matias, R., and Trivedi, K. S. (2008). The fundamentals of software aging. In *2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp)*, pages 1–6. Ieee.

Macêdo, A., Ferreira, T. B., and Matias, R. (2010). The mechanics of memory-related software aging. In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation*, pages 1–5. Ieee.