UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
UNIDADE ACADÊMICA DE GARANHUNS
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

FELIPE DIAS DE OLIVEIRA

Avaliação de Desempenho de Plataforma de Provisionamento de Contêineres de Virtualização para Computação em Nuvem

GARANHUNS - PE

2019

Felipe Dias de Oliveira

Avaliação de Desempenho de Plataforma de Provisionamento de Contêineres de Virtualização para Computação em Nuvem

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Unidade Acadêmica de Garanhuns da Universidade Federal Rural de Pernambuco.

Orientador:

Prof. Dr. Jean Carlos Teixeira de Araujo

GARANHUNS - PE

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

O48aa Oliveira, Felipe Dias de
Avaliação de Desempenho de Plataforma de Provisionamento de Contêineres de Virtualização para Computação em Nuvem / Felipe Dias de Oliveira. - 2019.
80 f. : il.

Orientador: Jean Carlos Teixeira de Ara Araújo.
Inclui referências e apêndice(s).

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Ciência da Computação, Garanhuns, 2019.

1. computação em nuvem. 2. avaliação de desempenho. 3. virtualização. 4. contêineres. I. Araújo, Jean Carlos Teixeira de Ara, orient. II. Título

CDD 004

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Unidade Acadêmica de Garanhuns da Universidade Federal Rural de Pernambuco, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. Jean Carlos Teixeira de Araujo - Orientador
Unidade Acadêmica de Garanhuns (UAG)
Universidade Federal Rural de Pernambuco (UFRPE)

Prof. Dr. Jamilson Ramalho Dantas - Examinador
Campus Salgueiro
Universidade Federal do Vale do São Francisco (UNIVASF)

Dr. João Ferreira da Silva Júnior - Examinador
Colégio Agrícola Dom Agostinho Ikas (CODAI)
Universidade Federal Rural de Pernambuco (UFRPE)

Aos meus pais Paulo e Maria.
Aos meus irmãos Lucas, Matheus, Poliana e Fabiana.
À minha amada Elaine.

Agradecimentos

Agradeço aos meus pais Paulo e Maria por terem me criado e educado da melhor maneira possível. Aos meus irmãos Lucas Matheus e Fabiana agradeço as palavras de incentivo e a consideração. Obrigado por todo o suporte e carinho. Agradeço à Poliana e Adeildo por terem me abrigado em sua casa boa parte desses 6 anos. Certamente se não fosse por vocês jamais teria condições de ter realizado esse curso. Obrigado pela paciência e compreensão ao longo desse processo.

À minha companheira e amada Elaine, sem seu apoio certamente não teria conseguido. Muito obrigado pelo seu carinho e incentivos ao longo desses anos e em especial nos últimos meses.

Agradeço aos amigos Juan Augusto, Diogo Espinhara, Felipe, Daniel Alves, Bruno Barros, Eduardo, Ramon Santos, Arnaldo, Wagner, Witássio, Diego Nascimento, Diego Freitas, Jorge, Ana Raquel, Paulo Matheus, Marrone Dantas, Vinícios, Jefferson, Israel, Glaydson, Luan Lins (pra cima deles Passarinho!), André Barreto, Raul e Cloves pelas incontáveis ajudas sempre de boa vontade, pelas inúmeras caronas e pelos momentos descontraídos. À todos os amigos citados e não citados meu muito obrigado.

Agradeço ao meu amigo David Beserra que em diversos momentos me incentivou a não desistir e seguir em frente. Obrigado pelas conversas, pelos cafés e pelas muitas caminhadas. Aprendi muito com você quando trabalhamos juntos (faz cluster mermao hehehe). Obrigado pela ajuda na revisão deste trabalho.

Agradeço à todos os professores que fazem parte do curso BCC, pelos ensinamentos de profissão e de vida. Sou grato à todos os técnicos que fazem a UAG. Obrigado a todos os terceirizados que fazem o apoio didático, segurança e limpeza dos prédios, pelo importante trabalho que desempenham, sem o qual não seria possível a realização das atividades de pesquisa e monitoria das quais participei ao longo desses anos. Meu agradecimento especial para Cris (Cristiane Galindo) pessoa única que tive a honra de conhecer, sempre disposta a ajudar com uma boa conversa e um bom café.

Agradeço ao Professor e meu amigo Jean Araújo, por todas as oportunidades e projetos que tivemos e trabalhamos ao longo desta jornada. Obrigado pela paciência, pelos incentivos a continuar e por ter acreditado que eu conseguiria mesmo quando nem eu acreditava. Obrigado pelo aprendizado, conselhos e ideias, assim como pelas conversas sem compromisso e cervejas. Agradeço ao grupo de pesquisa UNAME por todo o suporte.

Tenho o privilégio de não saber quase tudo.
E isso explica
o resto.
—MANUEL DE BARROS, (CADERNO DE APRENDIZ)

Resumo

Sistemas com arquiteturas baseadas em computação em nuvem (*cloud computing*) tem crescido nos últimos anos, principalmente porque constituem uma plataforma escalável, de serviço robusto e de baixo custo. A tecnologia chave para a computação em nuvem é a virtualização, que permitiu a criação de ambientes virtuais de hardware simulando servidores físicos. Os sistemas de virtualização, possibilitam que a infraestrutura dos servidores seja melhor aproveitada, reduzindo custos e facilitando a manutenção de servidores. Virtualizadores, como o Xen e o KVM, são utilizados na virtualização de recursos computacionais, mas também são conhecidos por penalizarem a infraestrutura de nuvem com *overheads*. Em contrapartida, os virtualizadores baseados em contêineres, como LXC e Docker, mostram-se promissores para serem utilizados como base para o provimento de serviços na nuvem, pois utilizam menos recursos para simular ambientes isolados. Diversos trabalhos avaliaram o impacto causado pela camada de virtualização em serviços hospedados dentro da infraestrutura virtualizada, sendo importante para que se conheça como diferentes aplicações sofrem interferência das técnicas de virtualização. Este trabalho propõe uma avaliação de desempenho de uma plataforma de virtualização baseada em contêineres, o Docker, para caracterizar a utilização de recursos em cenários de computação em nuvem. Diferentes cargas de trabalho foram aplicadas considerando diferentes modelos de serviços. Os resultados evidenciam a boa gestão de recursos do Docker em modo simples. Porém, há indícios de exaustão de recursos quando o modo *Swarm* está ativo.

Palavras-chave: computação em nuvem; avaliação de desempenho; virtualização; contêineres;

Abstract

Systems with cloud-based architectures has grown in recent years, mainly because they provide a scalable platform, robust service and low-cost. The key technology for cloud computing is virtualization, which enabled the creation of virtual hardware environments simulating physical servers. Virtualization systems enable server infrastructure to be better leveraged, reducing costs and making server maintenance easier. Virtualizers, such as Xen and KVM, are used in computing resource virtualization but are also known to penalize cloud infrastructure with overheads. In contrast, container-based virtualizers such as LXC and Docker are proving to be promising to be used as a basis for delivering cloud services because they use fewer resources to simulate isolated environments. Several studies have evaluated the impact of the virtualization layer on services hosted within the virtualized infrastructure and it is important to know how different applications are affected by virtualization techniques. This paper proposes a performance evaluation of a container-based virtualization platform, Docker, to characterize resource utilization in cloud computing scenarios. Different workloads were applied considering different service models. The results highlight Docker's good resource management in simple mode. However, there is evidence of resource exhaustion when the Swarm mode is active.

Keywords: cloud computing; performance evaluation; virtualization; containers;

Lista de Figuras

2.1 Modelos de serviço de computação em nuvem 20
2.2 Principais tipos de virtualização 23
2.3 Arquitetura geral do Docker *Engine* 25
4.1 Metodologia de execução de atividades 34
5.1 Carga de trabalho cenário #1 40
5.2 Arquitetura do *cluster httpd* criado com Docker no modo *Swarm* 41
5.3 Carga de trabalho cenário #2 42
5.4 Fluxo de execução do SystemTap 44
5.5 Passos de execução do SystemTap 45
6.1 Utilização de memória principal pelo hospedeiro no cenário #1A 47
6.2 Utilização de memória *Swap* pelo hospedeiro no cenário #1A 47
6.3 Utilização de CPU pelo hospedeiro no cenário #1A 48
6.4 Utilização de memória residente pelo processo *dockerd* no cenário #1A . . 48
6.5 Utilização de memória virtual pelo processo *dockerd* no cenário #1 49
6.6 Utilização de CPU e *threads* pelo processo *dockerd* no cenário #1A 49
6.7 Utilização de memória residente pelo processo *containerd* no cenário #1A . 50
6.8 Utilização de memória virtual pelo processo *containerd* no cenário #1A . . 50
6.9 Utilização CPU e *threads* pelo processo *containerd* no cenário #1A 51
6.10 Utilização de CPU pelo processo *NetworkManager* 51
6.11 Utilização de Memória pelo processo *NetworkManager* 52
6.12 Acumulativo de fragmentação de memória por processo no cenário #1A . . 53
6.13 Utilização de memória principal pelo hospedeiro no cenário #1B 54
6.14 Utilização geral de CPU no hospedeiro no no cenário #1B 55
6.15 Utilização recursos pelo processo *dockerd* no no cenário #1B 56
6.16 Utilização recursos pelo processo *containerd* no cenário #1B 57
6.17 Acumulativo de fragmentação de memória por processo no cenário #1B . . 57
6.18 Utilização de memória no hospedeiro *Master* 58
6.19 Utilização de memória no hospedeiro *Worker* 59
6.20 Utilização de CPU no hospedeiro *Master* 59
6.21 Utilização de CPU no hospedeiro Worker 60
6.22 Utilização de memória residente pelo processo *dockerd* no hospedeiro *Master* 61
6.23 Utilização de memória virtual pelo processo *dockerd* no hospedeiro *Master* 62
6.24 Utilização de CPU pelo processo *dockerd* no hospedeiro *Master* 62
6.25 Utilização de memória residente pelo processo *dockerd* no hospedeiro *Worker* 63
6.26 Utilização de CPU pelo processo *dockerd* no hospedeiro *Worker* 63

Lista de Tabelas

3.1 Comparação dos Trabalhos Relacionados 31
5.1 Tabela comandos linux utilizados 43
6.1 Total de Ocorrências de Fragmentação por Processo no cenário #1A 53
6.2 Total de Ocorrências de Fragmentação por Processo no cenário #1B 58
6.3 Estimativa de consumo de memória residente do processo *dockerd* no hospedeiro *Worker* 61

Lista de Acrônimos

API	*Application Programming Interface*
AWS	*Amazon Web Services*
CLI	*Cliente*
CPU	*Central Processing Unit*
DDR	*Double Data Rate*
EC2	*Elastic Compute Cloud*
GB	*Gigabyte*
HTTP	*Hypertext Transfer Protocol*
IaaS	*Infrastructure as a Service*
ID	Identificador
IoT	*Internet of Things*
IoTaaS	*IoT as a Service*
KVM	*Kernel-based Virtual Machine*
LXC	*Linux Containers*
LXD	*Linux Container Daemon*
MB	*Megabyte*
PaaS	*Platform as a Service - PaaS*
PC	*Personal Computer*
QoS	*Quality of Service*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
SaaS	*Software as a Service*
SLA	*Service Level Agreement*
SO	Sistema Operacional

TI Tecnologia da Informação

VM *Virtual Machine*
VMM *Virtual Machine Monitors*
VPS *Virtual Private Server*

Sumário

Lista de Figuras **vi**

Lista de Tabelas **vii**

Lista de Acrônimos **viii**

1 Introdução **15**
1.1 Objetivos . . . 17
1.1.1 Objetivo Geral . . . 17
1.1.2 Objetivos Específicos . . . 17
1.2 Estrutura do Documento . . . 17

2 Fundamentação Teórica **19**
2.1 Computação em Nuvem . . . 19
2.2 Virtualização . . . 22
2.3 Docker . . . 24
2.4 Avaliação de Desempenho de Sistemas . . . 25
2.5 Envelhecimento de *Software* . . . 26
2.6 Considerações Finais . . . 27

3 Trabalhos Relacionados **28**
3.1 Comparação dos Trabalhos . . . 30
3.2 Considerações Finais . . . 32

4 Metodologia **33**
4.1 Visão Geral . . . 33
4.2 Pré-Avaliação . . . 34
4.3 Avaliação . . . 36
4.4 Considerações Finais . . . 37

5 Planejamento dos Experimentos **39**
5.1 Ambiente de Testes . . . 39
5.1.1 Cenário #1: Utilização de Contêineres para Provimento de IaaS . . . 39
5.1.2 Cenário #2: Utilização de Contêineres para Provimento de IaaS e PaaS . . . 41
5.2 Recursos Monitorados e Ferramentas de Monitoramento Utilizadas . . . 43
5.2.1 Scripts de Monitoramento de Recursos . . . 43
5.2.2 SystemTap . . . 43
5.3 Considerações Finais . . . 45

6 Estudos Experimentais **46**
6.1 Análise dos Resultados 46
6.1.1 Resultados Cenário #1A 46
6.1.2 Resultados Cenário #1B - Sem NetworkManager 53
6.1.3 Resultados Cenário #2 58
6.2 Considerações Finais 64

7 Conclusões **65**
7.1 Contribuições 66
7.2 Limitações 66
7.3 Trabalhos Futuros 67

Referências **72**

Apêndice **73**

A Cargas de Trabalho **74**
A.1 Cargas de Trabalho Cenário #1 74
A.2 Cargas de Trabalho Cenário #2 76

B *Scripts* de Monitoramento **77**
B.1 Monitor de CPU 77
B.2 Monitor de Memória 77
B.3 Monitor de Disco 78
B.4 Monitor de Processo 79
B.5 Monitor de Processos Zumbis 79

Capítulo 1

Introdução

Milhares de pessoas em todo o mundo fazem o uso de algum recurso relacionado à computação em nuvem. O acesso à planilhas *online*, os filmes e séries que assistem por *streams* de vídeo: tudo isso está utilizando recursos computacionais em algum lugar chamado nuvem. A computação em nuvem, que até poucos anos atrás era vista com desconfiança mesmo por pessoas da área de Tecnologia da Informação (TI), é hoje uma das tecnologias mais demandadas do mercado. Com as redes móveis de alta velocidade e com a propensão de conectar tudo à Internet, estima-se que essa demanda deva crescer ainda mais nos próximos anos [10]. Esse fluxo crescente de dados que trafega na internet tem como origem ou destino serviços que estão funcionando em uma infraestrutura de diversos centros de dados distribuída globalmente.

Para suprir essa demanda crescente por poder computacional, os provedores de computação em nuvem tem que optimizar a utilização de recursos de seus servidores. Uma das principais tecnologias que serve de base para a computação em nuvem é a virtualização. Com ela é possível a criação de ambientes virtuais simulando servidores físicos. Os sistemas de virtualização, também chamados de virtualizadores ou *hypervisors*, possibilitaram que a infraestrutura dos servidores fosse melhor aproveitada, reduzindo custos e facilitando a consolidação de serviços.

A escolha da tecnologia de virtualização pode influenciar diretamente no desempenho dos serviços hospedados na infraestrutura virtualizada [32]. Problemas relacionados com algumas das principais plataformas de virtualização foram relatadas nos últimos anos [30, 23]. Tais problemas afetam os serviços hospedados nessas infraestruturas, prejudicando a confiabilidade e a disponibilidade das aplicações. Geralmente provedores de serviço formalizam acordos com seus clientes assegurando as condições do serviço que será prestado, os chamados acordos de nível de serviço (*Service Level Agreement*, SLA). O não cumprimento desses acordos acarreta em multas e compromete a reputação dos prestadores de serviços, já que não há reconhecimento da qualidade dos produtos e serviço que fornece. Com isso, minimizar desperdícios e utilizar tecnologias de virtualização

que façam boa gestão de recursos são cruciais para que provedores possam se manter competitivos.

O uso de aplicações baseadas em virtualizadores leves, também chamada de baseada em contêineres, tem crescido nos últimos anos. Isso se deve em parte à facilidades de migração, já que é possível migrar aplicações de ambientes de desenvolvimento para ambientes de testes e para produção, utilizando-se as mesmas configurações de ambiente. Por outro lado, há o interesse em alternativas aos virtualizadores tradicionais. Essa tecnologia é chamada de virtualização leve, pois utiliza menos recursos que os *hypervisors* tradicionalmente utilizados na nuvem. Um dos principais nomes do mercado é o Docker [13], uma plataforma de provimento de contêineres leve e amigável. Um dos principais pontos destacados quando se compara *hypervisors* e contêineres é o tempo de inicialização das instâncias virtuais [52]. Diferentes autores tem verificado questões de desempenho em contêineres, comparando com soluções de virtualização já consolidadas [44] e realizando análises mais específicas, como verificações de sobrecargas de aplicações que fazem uso intensivo de disco [48, 51] e aplicações de computação de alto desempenho [3, 5].

Contudo, em cenários de computação em nuvem, a sobrecarga causada pela camada de virtualização é apenas um dos fatores a analisar antes da adoção de uma tecnologia de virtualização. Diversas pesquisas relacionadas à sistemas de computação em nuvem baseados em contêineres vêm sendo desenvolvidas, tanto na área de modelagem [38] quanto para realizar simulações levando em consideração SLA e eficiência energética [37]. Outros autores [41] propõem um modelo para avaliação de desempenho de rede considerando adições de contêineres em uma nuvem computacional. Dada a relevância do tema, se faz necessário avaliar o consumo de recursos de uma tecnologia de virtualização baseada em contêineres sob influência de cargas de trabalho similares às encontradas em ambientes de computação em nuvem.

Virtualizadores, como o Xen e o KVM, são utilizados na virtualização de recursos computacionais e também são conhecidos por penalizarem a infraestrutura de nuvem com *overheads*[1] [49, 4]. Em contrapartida, os virtualizadores baseados em contêineres, como *Linux Containers* (LXC) e Docker, apresentam as principais características funcionais de outros tipos de virtualizadores. Essas características consistem na criação de máquinas virtuais (VM), gerenciamento de discos virtuais, de interfaces de redes, além do gerenciamento da infraestrutura virtual, como inicialização, parada e remoção. Além disso, são mais eficazes na gestão de recursos [50], possuindo desempenho similar a sistemas nativos [32, 45, 40, 5] e com a vantagem de tempo de inicialização reduzido [52, 1]. Porém, contêineres ainda não provém o isolamento necessário para algumas aplicações, como aquelas que fazem uso intensivo de disco [48, 3].

Diversos autores têm mensurado a sobrecarga de contêineres, comparando-os com

[1]Sobrecarga de processamento, armazenamento ou uso de rede, diminuindo o desempenho

outros sistemas virtualizados e sistemas não virtualizados (nativos). Esses trabalhos geralmente avaliam o impacto causado pela camada de virtualização em serviços hospedados dentro da infraestrutura virtualizada. Isso é importante para saber qual tipo de virtualização é mais indicado para cada tipo de aplicação e para que o provedor da infraestrutura consiga garantir a qualidade do serviço (*Quality of Service*, QoS). Porém, esse tipo de estudo não responde qual é a influência das cargas de trabalho intrínsecas ao ambiente de computação em nuvem, como o escalonamento de VMs. Por isso, nesse estudo busca-se responder a seguinte questão: como se caracteriza a utilização dos recursos de sistemas de virtualização baseados em contêineres sob cargas de trabalho típicas de ambientes de computação em nuvem? Para responder essa questão, um estudo experimental foi realizado utilizando-se o Docker, uma plataforma de virtualização baseada em contêineres.

1.1 Objetivos

Nesta seção são apresentados os objetivos gerais e específicos que aspiram ser alcançados ao fim desse trabalho.

1.1.1 Objetivo Geral

O objetivo geral desse trabalho é identificar qual é a influência das cargas de trabalho intrínsecas ao ambiente de computação em nuvem (como o escalonamento de VMs) no desempenho desses ambientes (em termos de utilização de recursos e indicadores de envelhecimento de *software*)

1.1.2 Objetivos Específicos

Têm-se como objetivos específicos:

- Caracterizar a utilização dos recursos de servidores hospedeiros pelo Docker durante a execução de diferentes serviços típicos de computação em nuvem;
- Caracterizar o desempenho da plataforma Docker na execução de diferentes serviços e modelos de serviço típicos de ambientes de nuvem;
- Investigar possíveis efeitos de envelhecimento de *software* no Docker durante a execução dos cenários avaliados;

1.2 Estrutura do Documento

A organização deste trabalho segue da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica dos conceitos utilizados e que serão úteis para a compreensão do

trabalho; o Capítulo 3 apresenta alguns dos principais trabalhos no tema; o Capítulo 4 descreve a metodologia adotada para o desenvolvimento das atividades; o Capítulo 5 descreve os cenários avaliados e o ambiente de testes; o Capítulo 6 Apresenta os resultados obtidos nesse trabalho. Por fim, o Capítulo 7 mostra as conclusões decorrentes da análise desses resultados.

Capítulo 2

Fundamentação Teórica

Este capítulo tem por objetivo apresentar conceitos fundamentais para o entendimento deste trabalho. Primeiramente são apresentados os conceitos fundamentais de computação em nuvem, seguidos por uma explanação sobre virtualização. Por fim, alguns conceitos sobre análise de desempenho são apresentados.

2.1 Computação em Nuvem

A Computação em nuvem é um paradigma computacional que possibilita que recursos computacionais configuráveis possam ser contratados e utilizados através da rede. Recursos como servidores, capacidade de processamento, espaço de armazenamento, aplicações e serviços, podem ser acessados de modo conveniente e sob demanda [31, 7]. Nos modelos de computação em nuvem ocorre o deslocamento de recursos computacionais para a Internet, como uma forma de reduzir os custos associados com o gerenciamento de recursos de *hardware* e *software* [47]. Os recursos computacionais de uma nuvem podem ser rapidamente obtidos e liberados, demandando um esforço gerencial mínimo ou pouca interação com o provedor de serviços.

Computação em nuvem utiliza modelos de negócios que são orientados a serviços [51]. Provedores de computação em nuvem podem ofertar serviços em diferentes níveis de abstração que geralmente são agrupados em três categorias [31]: Infraestrutura como serviço (*Infrastructure as a Service* - IaaS), Plataforma como serviço (*Platform as a Service - PaaS* - PaaS) e *Software* como serviço (*Software as a Service* - SaaS). Essas categorias podem ser visualizadas como camadas, onde a camada inferior serve a camada superior. A Figura 2.1 ilustra essa relação, onde a base está mais próxima dos proprietários e gerentes da infraestrutura e as camadas superiores aos usuários finais.

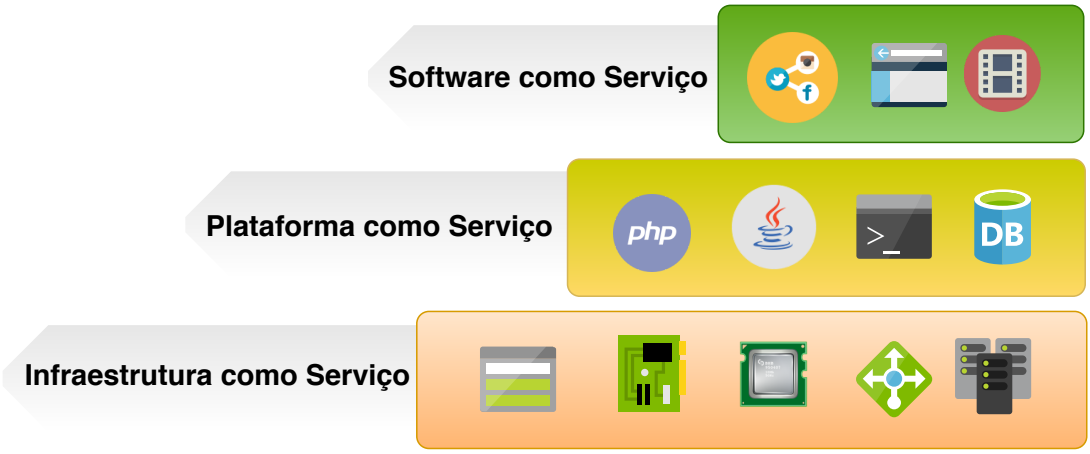


Figura 2.1: Modelos de serviço de computação em nuvem, adaptado de [25]

Infraestrutura como serviço (IaaS) refere-se à oferta de recursos de infraestrutura sob demanda para acesso e controle do cliente. O cliente pode armazenar dados diversos, bem como realizar a instalação e a execução de *softwares* arbitrários, que podem ser sistemas operacionais ou aplicações de acordo com suas necessidades. O cliente desse serviço não gerencia ou controla a infraestrutura onde a nuvem está, mas tem o controle sobre os sistemas operacionais, aplicativos implantados e armazenamento. Possivelmente, terá também o controle de recursos de rede (por exemplo, *firewalls*), ainda que de forma limitada [20]. Os provedores de IaaS utilizam virtualização de recursos para prover a infraestrutura, atribuindo e reatribuindo recursos de forma dinâmica de acordo com as demandas. Por isso, a IaaS tende a utilizar os recursos de forma mais eficiente, além de outros benefícios oriundos dessa tecnologia, tais como isolamento da carga de trabalho e confiabilidade [7].

Plataforma como serviço (PaaS) é um modelo onde os provedores concedem ambientes virtuais, incluindo sistemas, linguagens de programação e bibliotecas, propícios para o desenvolvimento e implantação de aplicações pelos clientes (desenvolvedores). Estes não gerenciam recursos subjacentes aos sistemas que utilizam, como sistemas operacionais, servidores, configurações de rede e armazenamento [31, 20];

Software como serviço (SaaS) é o modelo onde são fornecidos sistemas e aplicações, em ambientes virtuais que estão em execução na infraestrutura da nuvem. Desta maneira, usuários podem ter acesso às aplicações por meio de interfaces amigáveis destinadas à utilização de usuários finais [51]. Os recursos que as aplicações fornecidas necessitam estão na nuvem e são de responsabilidade do prestador do serviço. O usuário não tem acesso às configurações de infraestrutura ou da plataforma onde a aplicação que utiliza está sendo executada [20].

Dentre as características principais dos serviços ofertados como computação em nuvem [31], pode-se citar:

- Acesso sob demanda: o atendimento das demandas por recursos computacionais,

tanto o acréscimo ou redução de recursos, é orientado pelo cliente, conforme suas necessidades e sem a necessidade de interação humana com o provedor do serviço;

- *Pay-per-use*: permite que consumidores possam calcular os pagamentos aos fornecedores de acordo com a utilização dos serviços;
- Conectividade: característica que requer que o acesso aos servidores seja realizado em alta velocidade e que permite o tráfego de informações em grande volume. Permite que os recursos estejam disponíveis e sejam acessados por diferentes tipos de dispositivos, como por exemplo *notebooks*, PCs, *tablets* e celulares;
- Compartilhamento: tem-se a possibilidade de obtenção de ganhos de escala nas receitas dos serviços de computação em nuvem pelo compartilhamento do excesso de capacidade de infraestrutura de TI entre grupos de clientes;
- Abstração: há certa transparência quanto à localização dos recursos. Isto quer dizer que o cliente geralmente não sabe ou não tem controle sobre onde os recursos estão localizados de forma precisa. Pode-se ter especificações com níveis de abstração, onde o cliente pode selecionar a região onde os recursos estarão alocados;
- Comprometimento: cada provedor na computação em nuvem negocia com seus consumidores contratos que incluem cláusulas específicas para garantias de nível de serviço (SLA) [47] e condições de atendimento de demandas que podem ser desfavoráveis aos clientes. Para evitar a interrupção do serviço, o provedor deve adotar políticas para garantir alta disponibilidade e tolerância à falhas [7];

Os recursos computacionais da nuvem aparentam ser infinitos e estão sempre disponíveis para serem utilizados quando surgem picos de demanda [7, 47]. Essa elasticidade dos recursos ocorre de forma automática, sem que haja planejamento prévio dos usuários para a realização da ação.

A capacidade de pagar pelo o que for utilizado (*pay-per-use*), permite o uso de recursos computacionais sejam utilizados por curtos períodos de tempo [47]. Neste tipo de modalidade de cobrança, um utilizador da nuvem pode contratar processadores por hora e armazenamento por dia, como exemplos. Isso permite que a contratação aconteça somente em momentos em que o recurso é necessário, ocorrendo a liberação assim que não for mais exigido.

Para a realização da implantação de aplicações no ambiente de nuvem, alguns fatores devem ser observados. Pode-se pensar em migrar para a nuvem com objetivos de redução de custos operacionais com servidores; em contrapartida, pode-se ter a necessidade de características como confiabilidade e segurança [51]. Existem diferentes modelos de implantação de computação em nuvem, que podem ser divididos em quatro tipos: **nuvem privada, comunitária, pública e híbrida** [31].

No modelo de implantação de nuvem privada, a infraestrutura é mantida para uma única organização. Os recursos físicos que dão suporte a nuvem (por exemplo, servidores, *switches*, armazenamento, etc.) podem ser mantidos pela própria organização, por terceiros ou por uma combinação de ambos [31].

O modelo de implantação de nuvem comunitária é utilizado por um conjunto de organizações específicas que possuem interesses em comum (políticas de segurança, por exemplo) . Os recursos da nuvem podem ser mantidos por uma ou mais organizações que compõem a comunidade, ou por terceiros [31].

A nuvem pública é um modelo onde a infraestrutura é ofertada para o público geral por um provedor de nuvem [51]. Pode ser gerenciada por organizações governamentais, comerciais, acadêmicas ou por combinações destas [31].

Já no modelo de implantação de nuvem híbrida, a infraestrutura é uma composição entre duas ou mais infraestruturas de nuvem distintas (privada, pública ou comunitária). Essa composição faz com que essas infraestruturas diferentes sejam conectadas por meio de uma tecnologia padronizada ou proprietária, que realiza a interface e permite a atuação conjunta das infraestruturas [31].

Os serviços de computação em nuvem geralmente são hospedados em grandes centros de dados, com infraestrutura composta por milhares de computadores. Esses centros são criados para atender a muitos usuários e oferecer diversos serviços distintos. Para isso, a virtualização é uma tecnologia chave para superar a maioria dos problemas operacionais de construção e manutenção da infraestrutura [7].

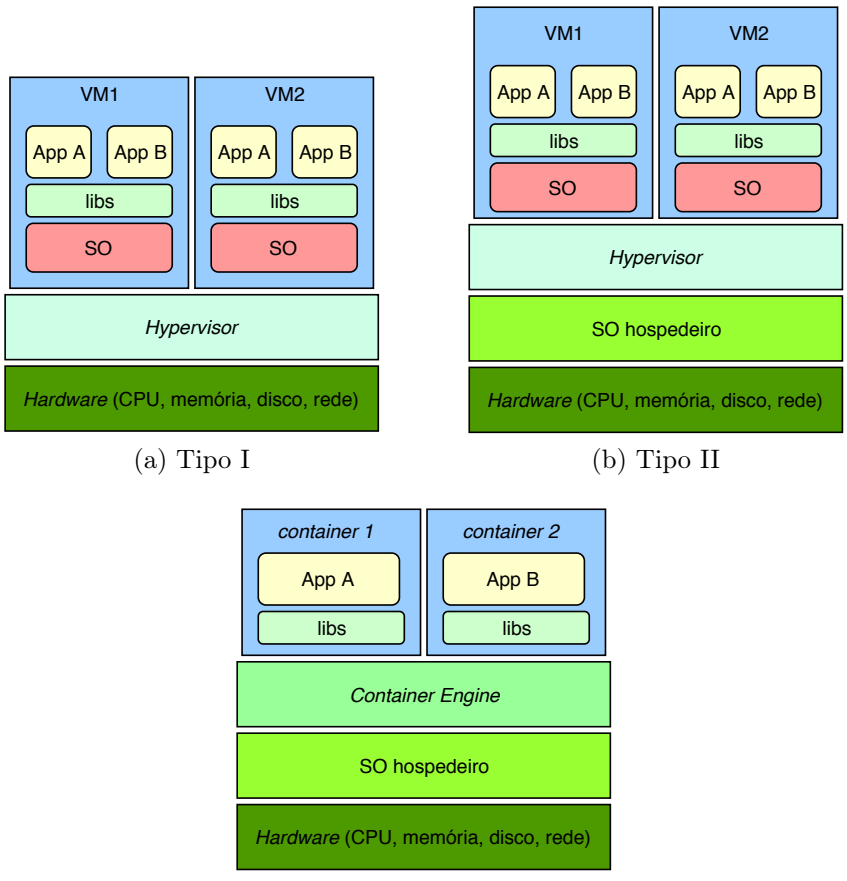
2.2 Virtualização

A virtualização pode ser entendida como um meio de abstrair recursos computacionais (CPU, memória RAM, disco, etc.) criando uma máquina virtual (VM), que é um ambiente isolado do Sistema Operacional (SO) hospedeiro. Isso permite a execução de múltiplos sistemas operacionais em uma mesma plataforma de *hardware* [7], compartilhando os recursos entre diferentes VMs. Existem duas técnicas de virtualização principais: a virtualização de *hardware* e a virtualização em nível de sistema operacional.

A virtualização de *hardware* utiliza um *software* especial, conhecido como monitor de máquinas virtuais (*Virtual Machine Monitors* - VMM) ou *hypervisor*, que é a camada que faz a intermediação entre o sistema operacional convidado e os recursos de *hardware* que são apresentados como VMs. Cada máquina virtual é um ambiente computacional independente, possuindo todos os recursos de uma máquina física, onde um SO e aplicações podem ser instalados. Isso permite que um usuário utilize diferentes ambientes em um único computador.

As formas de execução dos *hypervisors* podem ser de dois tipos: Tipo I ou *bare-metal*

e Tipo II [15]. A Figura 2.2 ilustra a pilha de componentes que geralmente é encontrada em sistemas de cada tipo. Na virtualização Tipo I, o *hypervisor* pode ser executado diretamente sobre o *hardware* ou há necessidade de modificações no SO do convidado para acesso a funções específicas do *hardware* (**paravirtualização**). Isto permite que o sistema convidado tenha acesso direto à recursos, o que melhora o desempenho quando comparado com virtualizadores de tipo I. Já no Tipo II ou *virtualização total*, há o SO do servidor hospedeiro entre o *hypervisor* e o *hardware*. Nesse tipo de sistema todas as instruções do SO convidado passam por verificações do *hypervisor*, implicando perdas de desempenho.



(a) Tipo I (b) Tipo II

(c) Contêiner

Figura 2.2: Principais tipos de virtualização, adaptado de [32, 4]

Já os **virtualizadores leves ou contêineres** são conhecidos por ter menor utilização de recursos e menos sobrecarga no sistema hospedeiro. Como não há emulação de *hardware*, não há necessidade de instalação de um SO convidado adicional. Nesse tipo de técnica, o *kernel* do SO hospedeiro é compartilhado com cada contêiner em execução. Isso faz com que os contêineres tenham seu tempo de inicialização inferior ao de uma VM convencional [44], além de desempenho otimizado próximo ao desempenho do sistema hospedeiro [4] .

Cada tipo de virtualização tenta resolver problemas diferentes. Virtualizadores tipos

I e II atendem os requesitos necessários para provimento de IaaS, como bom isolamento de recursos. *Contêineres* são ferramentas para entrega e desenvolvimento de *software*, com foco em PaaS [36]. Mas as possibilidades de utilização desse tipo de tecnologia não excluem aplicações em IaaS [43]. Existem diversas aplicações que fornecem virtualização baseada em contêineres como o LXC[1], OpenVZ[2], Rocket[3] e Docker. Tendo em vista que o Docker se tornou o padrão *de facto* em termos de solução de virtualização baseada em contêineres [46], e para delimitar o escopo desse trabalho, então o Docker foi escolhido como plataforma para realização dos testes.

2.3 Docker

O Docker [13] é uma das plataformas para provisionamento e gerenciamento de contêineres com ampla utilização no mercado [8]. É um projeto *opensource*, escrito em GO[4] pela empresa Docker, Inc[5]. Um contêiner Docker fornece uma maneira genérica de isolar um processo do resto do sistema. Isso se aplica a todos os processos filhos do processo inicialmente isolado. As tecnologias bases que permitem esse isolamento são ***Cgroups*** e ***Namespaces***. *Cgroups* é parte do *kernel* do SO hospedeiro que permite o compartilhamento e a limitação de recursos (como CPU, memória, etc.). *Namespaces* é responsável por criar barreiras que garantam que cada contêiner tenha seus IDs de processo, sua tabela de roteamento, etc., atuando em diferentes níveis de abstração. Possui uma arquitetura do tipo cliente-servidor, cujos os principais componentes é mostrada na Figura 2.3.

O *Docker Engine* é formado por três componentes principais:

- ***Docker CLI***: é um cliente de linha de comando onde usuários podem fazer requisições para o *dockerd*, como criar, iniciar e remover contêineres, através de uma API REST;
- ***Dockerd***: é o processo *daemon* do Docker, um programa de longa execução responsável por gerenciar os objetos Docker (contêineres, redes, imagens, etc.) e que pode se comunicar com outros *daemons* para gerenciar serviços. Fornece uma API REST para que outros programas possam se comunicar e receber solicitações;
- ***Containerd***: responsável por realizar a execução e supervisão dos contêineres, o gerenciamento das interfaces de rede, além de armazenamento e conexões de rede.

O Docker possui um recurso para gerenciamento de *clusters* de serviços chamado *Docker Swarm*. Esse modo quando ativado faz o gerenciamento de diferentes *Docker*

[1]Disponível em: https://www.linuxcontainers.org. Acesso em 01/07/2019.
[2]Disponível em: https://www.openvz.org. Acesso em 01/07/2019.
[3]Disponível em: https://github.com/rkt/rkt. Acesso em 01/07/2019.
[4]Disponível em: http://www.golang.org. Acesso em 01/07/2019.
[5]Disponível em: http://www.docker.com/company. Acesso em 01/07/2019.

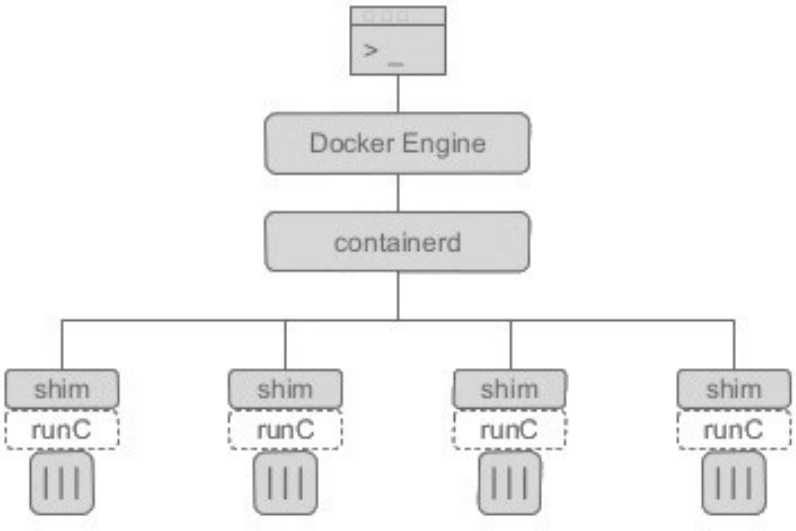


Figura 2.3: Arquitetura geral do Docker *Engine* [2][12]

Engines de forma nativa, permitindo que usuários através do Docker CLI controlem o comportamento do *cluster* e criem serviços de aplicações.

2.4 Avaliação de Desempenho de Sistemas

Segundo Lilja [27], a análise de desempenho aplicada à ciência e engenharia da computação experimental deve ser pensada como uma combinação de técnicas de medição, interpretação e comunicação da "velocidade" ou "tamanho" de um sistema de computador (às vezes chamada de "capacidade"). O autor destaca ainda que essas definições (capacidade, tamanho e velocidade) dependem do contexto de cada situação avaliada. As técnicas de avaliação de desempenho podem ser classificadas como **modelagem analítica**, **simulação** e **medição** [18, 27].

Sabe-se que a ação de monitorar um sistema computacional pode interferir no comportamento do sistema que está sendo medido. Ao considerar uma estratégia de medição, deve-se levar em conta a quantidade de pertubações que estão sendo inseridas no sistema objeto de observação [27]. Idealmente deve-se utilizar técnicas e ferramentas de medição que minimizem a inserção de perturbações no sistema. Quanto mais a estratégia de monitoramento interfere no sistema observado, menos confiáveis são os dados por ela coletados. Das estratégias apresentadas por Lilja [27] e por Jain [18], pode-se destacar:

- **Baseada em Eventos:** ocasiona uma perturbação no sistema apenas quando os eventos ocorrem. Condiciona a pertubação do sistema à frequência com a qual os eventos ocorrem. Se os eventos ocorrem com muita frequência, haverá muita pertubação sendo inserida; caso a frequência seja baixa, pouca pertubação será

inserida;

- **Baseada em Amostras:** amostras são obtidas a cada intervalo de tempo definido. Nessa estratégia, a pertubação do sistema não está condicionada à frequência com a qual os eventos ocorrem, mas sim ao tempo entre as medições.

As medições de desempenho compreendem a realização do monitoramento do sistema enquanto uma carga de trabalho específica é aplicada. Para que as medições sejam significativas, a carga de trabalho de testes deve ser selecionada criteriosamente. Cargas de trabalho podem ser reais ou sintéticas. Uma carga de trabalho real pode ser observada em um sistema sendo utilizado, realizando-se operações normais de utilização. Apesar de representar características do sistema de forma fidedigna, geralmente esse tipo de carga não pode ser reproduzida e, portanto, não é viável para uso como carga de trabalho em um cenário de teste. Já uma carga de trabalho sintética é desenvolvida e usada para estudos e representa um modelo da carga de trabalho real e pode ser aplicada repetidamente de maneira controlada. As vantagens de se utilizar uma carga de trabalho sintética estão na facilidade de modificação da carga sem afetar a operação [18].

Segundo Nelson [33], **testes de vida acelerado** são testes em que se aplica alto *stress* ao produto com o objetivo de acelerar a sua degradação. O objetivo desse tipo de teste é coletar dados sobre como o produto se comporta em situações extremas, para que estimativas de tempo de vida possam ser feitas. Isso é importante para medir a confiabilidade ou a degradação do produto no seu futuro. Caso fossem aplicadas níveis de *stress* normais, o produto poderia degradar de forma lenta ou nunca degradar. O teste acelerado permite que essas estimativas sejam obtidas em um tempo muito mais curto. Em sistemas computacionais, a degradação progressiva do desempenho está relacionada ao conceito de envelhecimento de *software*, um problema que afeta aplicações que executam de forma ininterrupta através de extensos intervalos de tempo.

2.5 Envelhecimento de *Software*

O termo envelhecimento do *software* não diz respeito à idade cronológica de um *software* ou à sua obsolescência. O envelhecimento do *software* é um fenômeno que pode ser observado como a diminuição progressiva do desempenho e/ou aumento da taxa de falhas de um *software* em seu tempo de execução [16].

O envelhecimento do *software* é um problema que afeta os sistemas de *software* que estão em execução por um longo período de tempo. Os efeitos do envelhecimento estão relacionados a *bugs* na etapa de desenvolvimento e ao acúmulo de erros, que se propagam ao longo do tempo induzindo falhas relacionadas ao envelhecimento [11, 16]. Esses erros podem interferir no funcionamento adequado do sistema, levando gradualmente a um estado com maiores tendências de falha.

Alguns problemas podem ocorrer em decorrência do envelhecimento, como exaustão de recursos do sistema operacional, diminuição do tempo de resposta de aplicações e inconsistência de dados. Os efeitos do envelhecimento podem ser observados monitorando-se os chamados indicadores de envelhecimento [16]. Exemplos de indicadores de envelhecimento podem ser vistos no ambiente computacional como aumento do consumo de memória, utilização excessiva de CPU e tamanho da tabela de disco.

Os indicadores também podem estar relacionados à uma aplicação, como sua utilização de recursos ou seu tempo de resposta. Esses indicadores de envelhecimento apontam o estado do sistema, permitindo que ações proativas possam ser tomadas para mitigar seus efeitos e evitando que o sistema interrompa seu funcionamento devido a um estado de falha.

Os principais sintomas de envelhecimento relacionados à memória são **vazamento e fragmentação de memória** [29]. O vazamento de memória ocorre quando espaço é alocado e não é liberado quando não é mais utilizado. Já a fragmentação de memória é resultado da dinâmica de alocação/desalocação de memória pelo SO, resultando em lacunas inutilizadas. Em ambos os problemas há possibilidade esgotamento da memória principal disponível.

2.6 Considerações Finais

Neste capítulo, foram apresentados uma introdução sobre os principais conceitos de computação em nuvem. Levou-se em consideração os tipos de serviço oferecidos e a classificação quanto ao modelo de implantação. Posteriormente, conceitos relativos à virtualização foram abordados. Foram feitas considerações sobre os tipos de virtualização e sobre as ferramentas de virtualização utilizadas neste trabalho. Por fim, foram apresentados conceitos sobre avaliação de desempenho de sistemas computacionais. Os conceitos de computação em nuvem e virtualização são importantes para a compreensão dos estudos de casos realizados, através de características como escalonamento de recursos. As decisões sobre características avaliadas no estudo experimental são baseadas considerando-se os conceitos de avaliação de desempenho aqui exibidos.

Capítulo 3

Trabalhos Relacionados

Este capítulo traz uma visão geral dos trabalhos que foram utilizados neste estudo e que representam uma amostra do estado da arte no tema. Alguns dos trabalhos avaliados fazem comparativos entre diferentes tipos de virtualização, evidenciando que a sobrecarga adicionada por diferentes técnicas de virtualização impacta diretamente na oferta dos recursos.

Alguns estudos vêm sendo realizados para avaliar a utilização do Docker em cenários de computação em nuvem. O trabalho de Silva [45] faz uma avaliação de desempenho de contêineres Docker para o provimento de aplicações em uma instituição pública. O autor compara a utilização de recursos de um serviço hospedado em contêineres Docker, VMs *Hyper-V* e sistema nativo. Considera dois cenários: verifica a utilização de recursos por testes de carga no serviço hospedado na infraestrutura, considerando situações de uso normal e de pico; verifica a sobrecarga causada pela adição de instâncias virtuais no hospedeiro, enquanto há utilização do sistema hospedado nas instâncias virtuais em situações de carga normal. Conclui que o uso de CPU e memória do Docker é $3,5$ vezes menor em comparação à VM, sendo próximo ao utilizado pelo sistema nativo, tanto em situações de utilização normal quanto em períodos e de alto uso. Pondera ainda que a sobrecarga nos recursos (CPU, memória, etc.) do sistema hospedeiro causada pelos contêineres é inferior ao causado por VMs do *hypervisor* Hiper-V.

Salah et al. [42] realizou uma avaliação de desempenho para comparar serviços baseados em computação em nuvem inseridos em contêineres Docker e em VMs, ambos hospedados no serviço EC2 da *Amazon Web Services* (AWS). Para isso, avaliou um serviço Apache HTTP em três cenários: no primeiro, considera um serviço *web* hospedado em um contêiner e em uma VM; no segundo cenário, avalia dois serviços *web* hospedados individualmente em dois contêineres; no terceiro cenário, avaliou mais de dois serviços hospedados na mesma instância de contêiner. O estudo obteve métricas de taxa de resposta por segundo e tempo de resposta utilizando a ferramenta de testes JMeter[1], uma ferra-

[1]Disponível em : https://www.jmeter.apache.org . Acesso em 01/07/2019.

menta de carga e de comportamento em serviços *web*, simulando requisições de usuários. A utilização de CPU também foi observada. Conclui que em todos os cenários avaliados, as aplicações baseadas em VMs tiveram melhor desempenho quando comparadas com os mesmos serviços hospedados em contêineres em condições equivalentes. Argumenta que o resultado desfavorável das aplicações conteinerizadas se deva ao fato que os contêineres da AWS estão executando sobre em VMs e não diretamente sobre os recursos do hospedeiro físico.

Já Morabito et al. [32] quantifica o nível de sobrecarga causado por diversas soluções de virtualização (KVM, LXC, Docker e OSv), utilizando como parâmetro de avaliação um sistema não virtualizado (nativo). Para isso, utiliza *benchmarks* que geram cargas de trabalho genéricas para conseguir métricas de aproveitamento de recursos, como poder de processamento, performance de rede E/S de disco e acesso à memória. Destaca o mal desempenho do virtualizador KVM quando comparado com as demais tecnologias analisadas, principalmente em operações de escrita cujo desempenho foi 50.26% inferior ao obtido no sistema não virtualizado. Frisa ainda, assim como nos trabalhos anteriores, que o desempenho obtido pelo LXC e Docker mostram que a sobrecarga gerada pela camada de isolamento são desconsideráveis.

Li et al. [26] compara o impacto de instâncias virtuais do virtualizador VMWare com contêineres Docker e o sistema nativo, considerando também como esse impacto varia no tempo. Suas considerações são similares às dos demais trabalhos já relatados, que contêineres produzem menos *overhead* que VMs tradicionais, mas que podem apresentar gargalos em transações de armazenamento.

Zhang et al. [52] avalia o uso de contêineres para aplicações de *big data*. Para isso faz uma comparação com KVM e Docker, utilizando a aplicação Spark como plataforma *big data*. Considera fatores quantitativos e qualitativos: a eficiência no uso dos recursos, como CPU e memória, e quesitos como o quão eficaz a configuração do ambiente de execução e questões de usabilidade. Apesar de afirmar que contêineres são mais eficazes na utilização de recursos, ressalva que podem apresentar gargalos em operações que fazem intensivas transações de armazenamento. Essa conclusão é reforçada em [19].

Jawarneh et al. [19] compara diferentes tecnologias de virtualização sob a ótica da qualidade de Serviço (QoS), criando *clusters* de VMs com a plataforma de computação em nuvem *OpenStack*. Avaliam o KVM, representando *hypervisors*, além de LXD e Docker, representando os virtualizadores baseados em contêineres. Consideram métricas de CPU, rede e disco, e mostram que ainda existem problemas a serem levados em consideração antes de uma migração completa para uma solução de computação em nuvem baseada em contêineres. Os resultados do LXD para escrita e leitura de disco apresentam resultados insatisfatórios. Isso deve-se à falhas de isolamento de recursos do LXC sob cargas de trabalho de uso intensivo de disco, como apontado em [48].

Zeng et al. [50] faz um levantamento dos principais modelos de rede para contêineres e

realizam experimentos para averiguar o desempenho de três soluções de rede para contêineres (*Flannel, Swarm Overlay, Calico*). Seus resutados mostram que o *Calico* apresenta melhor desempenho, aproximando-se do sistema nativo. Destaca o desempenho do *Swarm Overlay*, por estar próximo ao desempenho nativo, ser de fácil configuração e ser nativo da plataforma Docker, sendo uma solução relevante para diversas aplicações.

Celesti et al. [9] avalia o desempenho de contêineres Docker em dispositivos para internet das coisas (*Internet of Things*, IoT), na chamada nuvem IoT. Esse é um tipo de sistema distribuído que conecta dispositivos IoT com a nuvem para provimento de IoT como serviço (IoTaaS). Utiliza dois *Raspberry Pi* para a realização dos experimentos, onde realiza requisições entre eles utilizando um protocolo específico para comunicação assíncrona e mensurou os tempos de resposta considerando diferentes cenários. Considera que a sobrecarga ocasionada pela camada de virtualização é, medida em tempo de resposta, como 280 ms a mais que quando considerado a aplicação no sistema nativo.

Piraghaj et al. [38] apresenta uma arquitetura para simulação de cenários de computação em nuvem baseados em contêineres, o *ContainerCloudSim*. Como o nome indica, é uma extensão do simulador de ambientes de computação em nuvem, o *CloudSim*[2]. Mostram como a ferramenta pode ser utilizada para simular e comparar situações de provisionamento de recursos com base em eficiência de energia e políticas de SLA. Mostram três casos de uso e validam a escalabilidade do simulador utilizando medições de um ambiente real. Em [37], foram realizadas simulações para mensurar o desempenho do consumo energético de modelos de virtualização baseados em contêiner, considerando taxas de migração dos contêineres, violações de SLA e a média de VMs criadas. As simulações dos modelos foram executadas utilizando-se o *ContainerCloudSim* e os resultados apontam que o algoritmo apresentado pelos autores foi mais eficiente que os demais.

Rista et al. [41] propõe um modelo baseado em redes de Petri estocásticas para representar infraestruturas de rede de nuvens baseadas em contêineres. Realizaram experimentos onde comprovaram que o modelo pode prever a sobrecarga de novos contêineres em ambientes de nuvem.

3.1 Comparação dos Trabalhos

Percebe-se que há um consenso entre os trabalhos analisados de que contêineres adicionam pouca sobrecarga no sistema hospedeiro, obtendo resultados próximos ao do sistema nativo [40, 45, 50, 19, 26, 32]. Boa parte dos trabalhos fazem análises comparativas entre diferentes tipos de virtualização ou mesmo entre diferentes tecnologias de provimento de contêineres. Esses trabalhos levam em conta a sobrecarga gerada pela camada de virtualização. O fato de o contêiner avaliado em [42] estar sobre uma camada de virtualização

[2]Disponível em: https://cloudsimplus.org. Acesso em 01/07/2019.

limita a comparação feita pelos autores, pois os ambientes comparados não estão em condições equipotentes.

Tabela 3.1: Comparação dos Trabalhos Relacionados

Artigos	**Tipo de Avaliação**			**Contexto**
	Analítica	Medição	Simulação	
[32]		X		Benchmarking
[40]		X		Utilização de Recursos
[1]		X		
[48]		X		IaaS, Data Centers
[37]			X	Consumo energético
[22]	X			Micro Serviços
[9]		X		IoT
[26]		X		
[42]		X		WebServices, PaaS
[50]		X		Data Centers
[38]			X	Data Centers
[45]		X		PaaS
[28]	X		X	Computação em Névoa
[52]		X		IaaS, Big Data
[41]			X	Computação em nuvem, IaaS
[19]		X		IaaS, QoS
Este Trabalho		X		IaaS, PaaS

Considerar o impacto causado por diferenças inerentes às tecnologias que proveem a camada de virtualização é apenas um dos aspectos que devem ser observados na tomada de decisão sobre qual opção de virtualização adotar para prover os serviços na nuvem. Levar em consideração como os recursos são utilizados no tempo é importante para definir potenciais gargalos que impeçam a oferta do serviço com a qualidade acordada. Este trabalho difere dos demais pelo fato de verificar como os recursos utilizados pelos principais processos do Docker variam ao longo de um determinado período de tempo, sendo possível avaliar como estão sendo impactados pelo ato de escalonar as aplicações hospedadas na infraestrutura virtualizada. Optou-se pela realização de um estudo experimental para que se possa obter métricas que venham auxiliar a validar e/ou refinar modelos.

3.2 Considerações Finais

Neste capítulo, foi efetuado um levantamento de alguns dos principais trabalhos publicados relacionados a avaliação de desempenho de contêineres. Diversos trabalhos fazem avaliações comparativas com outras soluções de virtualização, mensurando métricas de desempenho e mostrando diferenças de sobrecargas e variações de desempenho. Outros valem-se de simulações para avaliar configurações de rede e eficiência de consumo de energia. Os contextos de aplicações vão deste aplicações de *big data* ao que é chamado de nuvem IoT.

Capítulo 4

Metodologia

Neste capítulo será apresentada a metodologia adotada durante a execução do trabalho. A organização do capítulo é descrita a seguir: Visão geral, Pré-Avaliação, Avaliação e Considerações finais.

4.1 Visão Geral

A metodologia adotada neste trabalho foi divida em duas macro-atividades: **Pré-avaliação** e **Avaliação**. Dentro de cada macro-atividade há outras subdivisões visando melhor estruturar as atividades. A pré-avaliação é subdivida em quatro atividades : estudo do sistema, planejamento da estratégia de monitoramento, planejamento da carga de trabalho e execução dos experimentos. Na macro-atividade de avaliação, as atividades são divididas em: realização de ajustes e análise de desempenho.

Um diagrama de fluxo que representa graficamente a metodologia adotada é mostrado na Figura 4.1. Neste diagrama, as ações são realizadas seguindo a ordem das setas que ligam cada caixa, partindo do ponto inicial até o estado final. Cada caixa indica uma etapa a ser realizada na execução das atividades. As etapas são realizadas de forma sequencial e somente quando todas as tarefas referentes a etapa atual são realizadas é que segue-se para a próxima etapa da sequência. O losango indica uma etapa de verificação que pode resultar em caminhos diferentes, dado o resultado da verificação. As setas horizontais indicam possíveis ações que podem ser adotadas em cada etapa. Cada elipse representa possíveis ações, estratégias, etc. para cada atividade da metodologia.

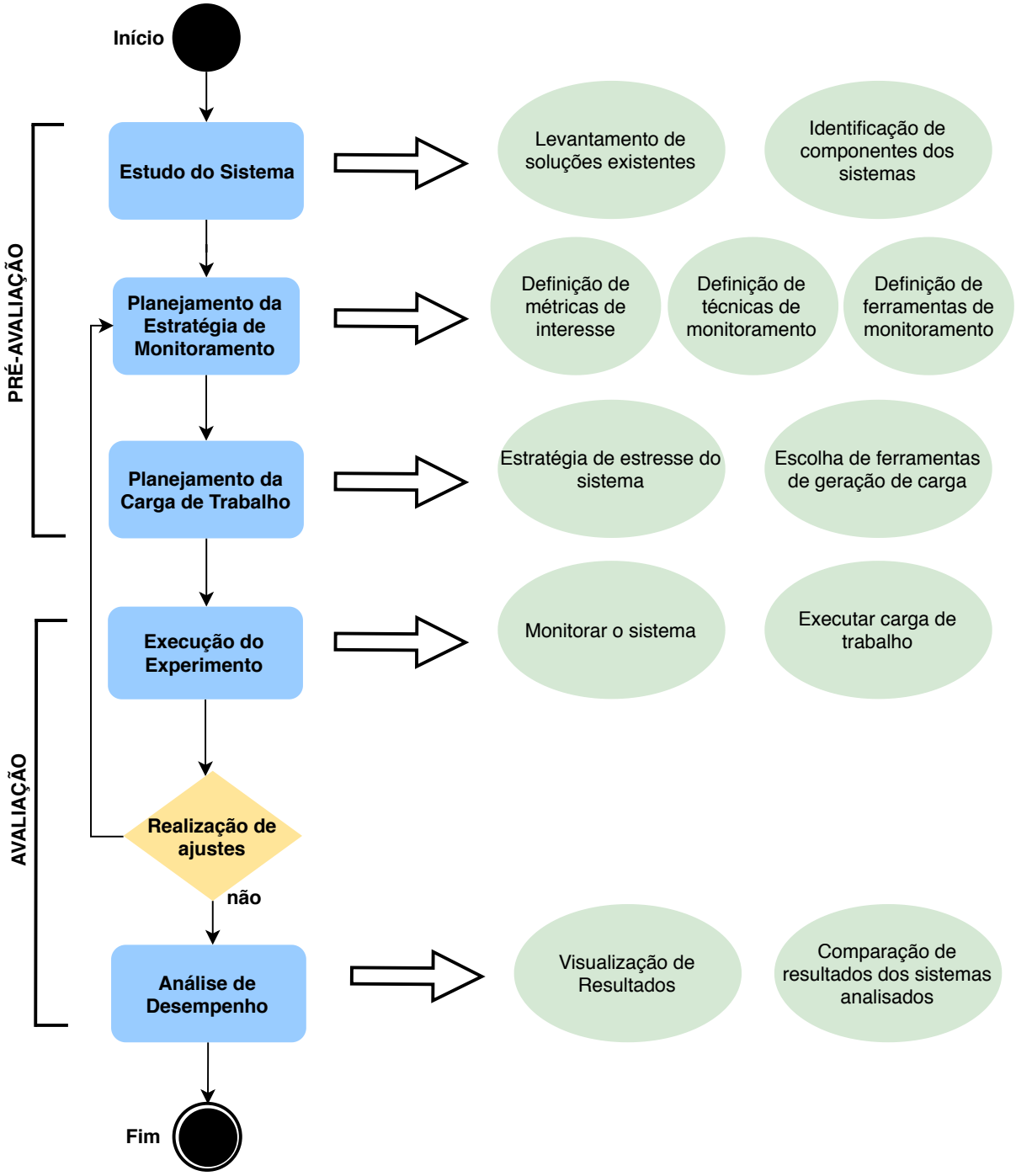


Figura 4.1: Metodologia de execução de atividades

4.2 Pré-Avaliação

O início do fluxo de atividades começa com a pré-avaliação. Esta etapa é constituída pelas atividades de estudo do sistema, planejamento da estratégia de monitoramento e planejamento da carga de trabalho. Cada atividade tem por finalidade fornecer os requisitos necessários para a execução das etapas posteriores. Abaixo segue uma descrição de cada uma das atividades contidas na pré-avaliação.

Estudo do sistema: nesta atividade deve-se compreender o funcionamento dos sistemas analisados, realizando a identificação dos componentes principais de cada sistema, assim como fazendo levantamentos das principais soluções existentes. Outro objetivo desta etapa é a aquisição de conhecimentos e técnicas que poderão ser utilizadas e adaptadas, bem como a identificação de quais soluções deverão ser criadas.

- Pré-condições: conhecimentos básicos em computação;
- Entradas: leitura de referências (artigos científicos, livros, sites, etc.) sobre computação em nuvem, servidores *web* e avaliação de desempenho;
- Ações: levantamento de soluções e identificação dos componentes dos sistemas;
- Produto: referencial teórico e trabalhos relacionados;
- Pós-condições: entendimento do funcionamento de infraestruturas de computação em nuvem, conhecimentos sobre funcionamento de servidores *web* e técnicas de avaliações de desempenho.

Planejamento da estratégia de monitoramento: agora que obteve-se os conhecimentos necessários sobre a infraestrutura de computação em nuvem e técnicas de avaliação de desempenho, pode-se planejar estratégias de monitoramento do ambiente. Este planejamento tem por objetivo obter as informações necessárias para a análise levando-se em consideração minimizar possíveis interferências no funcionamento dos sistemas observados [18, 27]. Nesta etapa serão definidas questões como ferramentas que serão utilizadas para obtenção das métricas, métricas de interesse (como tempo de resposta, utilização de CPU, consumo de memória RAM, uso de disco, tráfego de rede, etc.) e quais processos serão alvos de monitoramento.

- Pré-condições: conhecimentos sobre o funcionamento de virtualizadores, servidores *web* e análise de desempenho;
- Entradas: lista dos componentes dos sistemas analisados, ferramentas de monitoramento existentes ou requisitos para o desenvolvimento de soluções de monitoramento;
- Ações: escolha de métricas a serem observadas, assim como a maneira como serão mensuradas;
- Produto: métricas a serem avaliadas e relação de ferramentas a serem utilizadas para a tarefa;
- Pós-condições: métricas de interesse e ferramentas para obtê-las.

Planejamento da Carga de Trabalho: uma vez que foram definidas as métricas de interesse, pode-se avançar para a etapa de planejar uma carga de trabalho para estressar o sistema. Nesta atividade será definida qual o tipo de carga de trabalho adotada, como será gerada e quais parâmetros serão ajustados para que ela se aproxime de situações reais no cenário analisado.

- Pré-condições: conhecimentos sobre o funcionamento de virtualizadores, servidores *web* e análise de desempenho;
- Entradas: lista dos componentes dos sistemas analisados, ferramentas de geração de carga de trabalho existentes ou requisitos para o desenvolvimento de soluções de geração de carga,
- Ações: escolha da carga de trabalho;
- Produto: carga de trabalho;
- Pós-condições: carga de trabalho pronta para ser aplicada na próxima etapa.

4.3 Avaliação

A etapa de avaliação é composta por três atividades: execução do experimento, realização de ajustes e análise de desempenho. Neste ponto, o objetivo é valer-se do conhecimento adquirido nas etapas anteriores para executar o que foi planejado, tendo-se como produto final a análise do cenário avaliado.

Execução do Experimento: com o conhecimento obtido e o devidos planejamentos necessários realizados, pode-se por em prática e executar as ações de estresse e monitoramento dos sistemas. O objetivo desta etapa é configurar o ambiente de testes com todos o ferramental necessário para a realização do experimento e obtenção das métricas, além da execução do estudo experimental.

- Pré-condições: métricas de interesse; carga de trabalho;
- Entradas: métricas de interesse; lista de ferramentas necessárias para o monitoramento; carga de trabalho; lista de ferramentas para a execução da carga de trabalho;
- Ações: realizar a configuração do ambiente com as ferramentas necessárias para a correta execução das ações planejadas e a realização da execução do experimento;
- Produto: arquivos de registro com os dados coletados durante o experimento;
- Pós-condições: dados das métricas de interesse.

Realização de ajustes: com os dados obtidos ao fim da execução de cada experimento é possível realizar uma análise preliminar e averiguar a necessidade de ajustes nas etapas de planejamento do experimento. Em caso de necessidade, ajustes são realizados para adequar as métricas e técnicas de monitoramento refazendo-se as etapas anteriores. Caso não haja a necessidade de alterações nas etapas anteriores, segue-se para a próxima atividade do fluxo.

- Pré-condições: dados a serem utilizados como entrada obtidos;
- Entradas: arquivos com os dados das métricas de interesse;
- Ações: avaliar necessidade de replanejamento das etapas anteriores;
- Produto: decisão sobre a realização da próxima etapa de análise ou realização de alterações nas etapas de planejamento;
- Pós-condições: decisão de seguir para a etapa de análise com os dados coletados no cenário projetado ou voltar a etapas anteriores.

Análise de Desempenho: ao fim da etapa anterior, tem-se os dados das métricas de interesse obtidos enquanto a carga de trabalho é aplicada no sistema observado. Na etapa de análise de desempenho, os dados coletados são tratados, organizados para uma melhor compreensão das informações, analisados e comparados entre os diferentes sistemas analisados. Nessa etapa buscam-se também por indícios de envelhecimento de *software*.

- Pré-condições: dados obtidos de acordo com o cenário planejado;
- Entradas: arquivos com os dados das métricas de interesse;
- Ações: analisar os dados de forma objetiva, gerando gráficos e tabelas para auxiliar a visualização dos resultados;
- Produto: gráficos e tabelas com informações sobre o sistema monitorado;
- Pós-condições: considerações sobre as informações obtidas.

4.4 Considerações Finais

Este capítulo descreveu a metodologia adotada para a realização deste trabalho. Foram descritas todas as etapas necessárias para a reprodução do trabalho. Destaca-se que o procedimento metodológico adotado assume a forma de uma cadeia onde todo o conhecimento adquirido em uma etapa serve como base para a atividade seguinte. Inicialmente realiza-se um levantamento sobre o conhecimento básico para que se passe a etapa posterior, a etapa de planejamento. Nela sabendo-se como é o funcionamento do sistema

analisado pode-se planejar estratégias de monitoramento e quais técnicas serão utilizadas. Esse conhecimento obtido na etapa inicial também é de valia para a etapa de geração de cargas de trabalho, onde são definidas as formas que o sistema será estimulado. Na após a etapa de execução do experimento são verificadas as necessidades de ajustes e, se necessário, etapas anteriores são repetidas até que se possa avançar para avaliar os dados obtidos.

Capítulo 5

Planejamento dos Experimentos

Esse capítulo tem o objetivo de formalizar as decisões de projeto que foram tomadas ao longo da execução do estudo. Cada um dos cenários avaliados é descrito neste capítulo. Além disso, serão apresentadas quais foram as métricas coletadas, bem como o ferramental utilizado para a geração de cargas de trabalho, realização do monitoramento e tratamento das métricas.

Este capítulo está organizado na seguinte ordem: a Seção 5.1 descreve as configurações de *hardware* e *softwares* utilizados como base para os experimentos e os cenários que foram avaliados; a Seção 5.2 exibe os recursos que foram monitorados, assim como os intervalos de coleta das métricas e ferramentas utilizadas. Por fim tem-se as considerações finais do capítulo.

5.1 Ambiente de Testes

Para este estudo, o ambiente de teste foi composto por um computador HP Compaq 6005, equipado com processador AMD Athlon II X2 220 de 2,8 GHz, 8 GB DDR III de memória principal e 8 GB adicionais de *swap*. O sistema operacional utilizado foi o Debian 9 *Stretch* (*kernel* 4.9.0-8-amd64). O Docker foi instalado na sua versão 17.05.0-ce. Essa foi a configuração padrão em todos os cenários avaliados. Em situações em que mais de um computador foi utilizado, deve-se considerar que os computadores possuem configuração idêntica.

5.1.1 Cenário #1: Utilização de Contêineres para Provimento de IaaS

O objetivo deste cenário é simular ações de um usuário que utiliza serviços de infraestrutura como um serviço, a exemplo de um serviço de VPS[1]. Neste tipo de serviço, ações

[1] *Virtual Private Server*

como instanciação, reinicialização e remoção de contêineres são corriqueiras. Essas ações podem ser feitas de forma automatizada ou por ações indicadas pelo usuário. A carga de trabalho utilizada foi pensada para simular situações de uso do Docker nesse contexto, aparentando ações de usuários convencionais. O diagrama mostrado na Figura 5.1 ilustra a carga de trabalho utilizada no experimento.

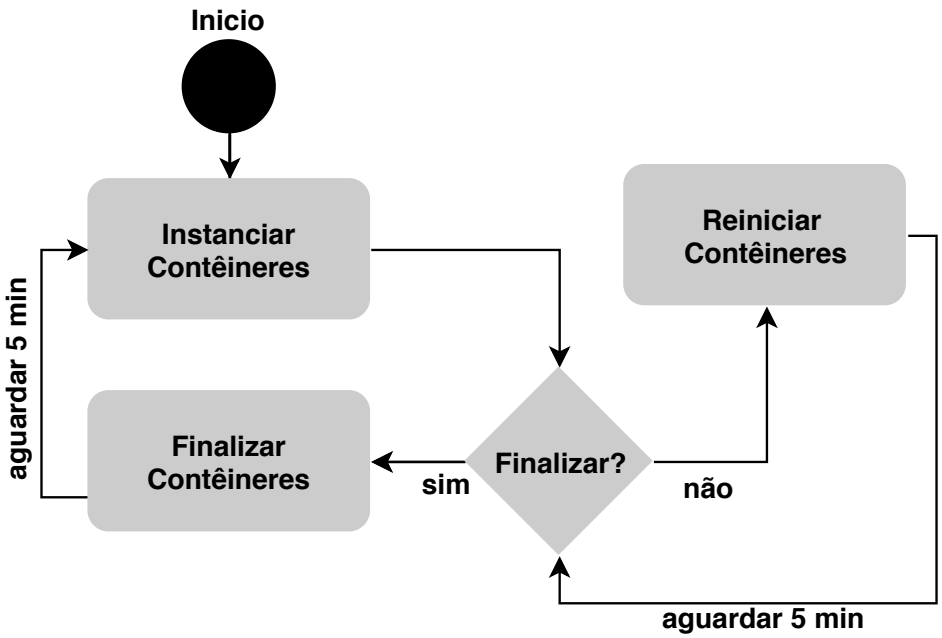


Figura 5.1: Carga de trabalho cenário #1

A atividade de instanciar contêineres consiste em criar todos os contêineres através do comando *docker create* e iniciá-los através do comando *docker start*. Nesta etapa, um contador de tempo é inicializado ($T = 0$) e 20 contêineres são criados com uma imagem de um serviço *web* Apache *httpd* e iniciados na sequência. Essa quantidade de contêineres foi escolhida com base na memória ofertada pelo servidor hospedeiro. Cada contêiner possui uma limitação no consumo de memória de 256 MB. Vale destacar que essa memória não está reservada, mas é um limite de utilização imposto ao processo que executa dentro do contêiner. No caso de todos os contêineres cheguem ao limite de utilização dessa memória (o que acredita-se que não ocorre, já que nenhuma requisição é feita no serviço que o contêiner hospeda), seriam utilizados 5120 MB, restando 3072 MB de memória disponível para o SO e o processo *daemon* do Docker. A cada 5 minutos é realizada uma verificação para checar se o tempo de execução dos contêineres é maior que 2 horas. Caso a condição seja verdadeira, os contêineres são finalizados. Caso a condição seja falsa, os contêineres são reiniciados (*docker restart*) e o contador T é incrementado. A atividade Finalizar Contêineres é formada pelas tarefas de parar (*docker stop*) e remover (*docker rm*) todos os contêineres. Após essa atividade, aguarda-se 5 minutos e um novo ciclo é iniciado. Empiricamente verificou-se, em algumas situações na etapa de execução do experimento, que os contêineres levaram de 45 segundos à 2 minutos para serem reiniciados. Com essa

constatação, definiu-se o tempo de 5 minutos entre as reinicializações/remoções. Isso serve para garantir que todos os contêineres sejam reiniciados ou removidos antes do próximo ciclo.

A automação da carga acima foi realizada utilizando um *script Bash*, que encontra-se disponível no Apêndice A.1. Para a realização desse Cenário #1 computador com as configurações descritas na Seção 5.1.

5.1.2 Cenário #2: Utilização de Contêineres para Provimento de IaaS e PaaS

Neste cenário, o objetivo é avaliar o desempenho da plataforma Docker com o modo *Swarm* ativo. Com esse recurso é possível criar *clusters* de computadores hospedeiros para serviços hospedados em contêineres Docker. Dessa forma é possível escalonar o serviço sempre que necessário e definir políticas que realizem essa ação de forma automática. Em um contexto de computação em nuvem, a criação de *clusters* de serviços garantem alta disponibilidade do serviço. Esse recurso é útil para criar *clusters* de banco de dados ou serviços que utilizem o protocolo *http*, ideais para oferta de plataforma como serviço, por exemplo.

Para a condução de um experimento utilizando o recurso, um *cluster* de dois computadores com as configurações descritas na Seção 5.1 foi criado utilizando-se o modo Swarm do Docker. A Figura 5.2 ilustra a arquitetura utilizada.

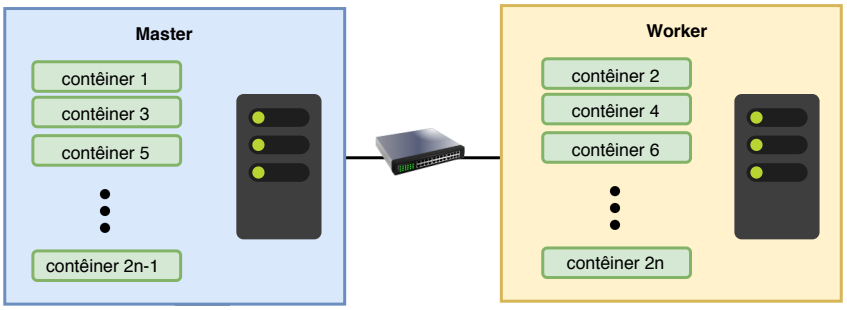


Figura 5.2: Arquitetura do *cluster httpd* criado com Docker no modo *Swarm*

O *cluster* é formado por dois servidores com as configurações descritas na seção 5.1. Quando o modo *Swarm* é habilitado, um serviço de descoberta de rede é ativado para auxiliar na descoberta dos nós. Após a configuração do *cluster*, foi realizada a criação de um serviço com uma imagem *apache httpd*. O trabalho consiste em agregar novos contêineres ao serviço e desagrega-los sempre que o limite definido seja atingido. A Figura 5.3 ilustra os passos da carga de trabalho utilizada. O *script Bash* que automatiza a carga de trabalho está disponível para consulta no Apêndice A.2.

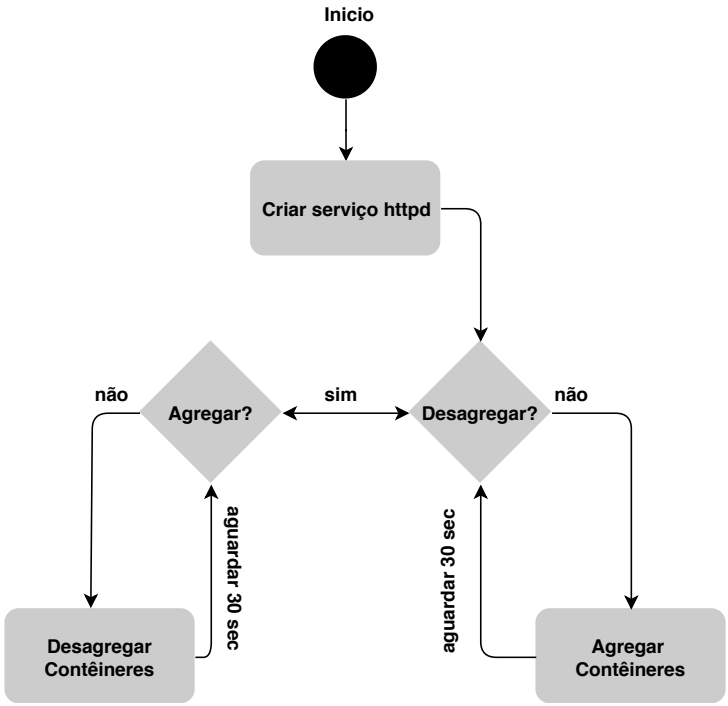


Figura 5.3: Carga de trabalho cenário #2

O processo de carga de trabalho é iniciado com a criação do serviço *httpd* com o comando *docker service*. Ao fim da criação do serviço, um contêiner terá sido hospedado na infraestrutura. O passo posterior é a verificação se há a necessidade de desagregar contêineres do serviço. Essa verificação é feita através de uma variável que controla a quantidade de contêineres do serviço. Sempre que existirem 40 contêineres hospedados na infraestrutura, a etapa de desagregação do contêineres é iniciada. Enquanto o número de contêineres é inferior a 40, a etapa agregar contêineres é realizada. Nesta etapa, o serviço é escalonado para $n = n + 5$, sendo n o número de contêineres da rodada anterior. Após essa etapa, aguarda-se 30 segundos e uma nova verificação de desagregação é feita.

Quando a quantidade de contêineres chega a 40, a etapa de desagregação é ativada e passa-se para a etapa de verificação se há necessidade de agregação. Essa etapa só será afirmativa caso a quantidade de contêineres seja $n = 1$. Enquanto a condição para novas agregações não é satisfeita, passa-se para a etapa de desagregar contêineres. A etapa consiste em remover contêineres do serviço para que a quantidade n agora seja $n = n - 5$. Após uma desagregação, aguarda-se 30 segundos e uma nova verificação de necessidade de desagregação é feita.

O escalonamento (agregação e desagregação de contêneres) é feito através do comando *docker service scale*, junto com o número de contêineres que devem estar ativos na rodada. Os experimentos foram encerrados ao fim de 25 dias de execução da carga de trabalho. Os *logs*[2] de dados foram coletados e realizou-se a etapa de análise de desempenho.

[2]Arquivos de texto onde os dados são armazenados

5.2 Recursos Monitorados e Ferramentas de Monitoramento Utilizadas

Esta seção apresenta os principais recursos monitorados e as ferramentas utilizadas nesse processo.

5.2.1 Scripts de Monitoramento de Recursos

Paralelamente à execução da carga de trabalho, o sistema é monitorado utilizando *scripts Bash*. Esses *scripts* utilizam comandos utilitários do Linux, como *free, ps, df* e do sistema de arquivos */proc* [6]. A Tabela 5.1 exibe informações sobre os principais comandos e utilitários usados no processo de monitoramento. Foram monitorados recursos de interesse gerais como CPU, uso de disco e utilização de memória de todo o sistema. Além disso, os consumos de processos específicos foram observados, como os do Docker e NetworkManager [34]. Os scripts coletam dados a cada 60 segundos. Esse tempo foi definido com o objetivo de minimizar as possíveis interferências do monitoramento no funcionamento do sistema, como é recomendado em [27].

Tabela 5.1: Tabela comandos linux utilizados

Comando/Utilitário	**Descrição**
free	Exibe a quantidade de memória livre e utilizada no sistema
ps	Mostra o status dos atuais processos no sistema
df	Relata o espaço de disco usado pelo sistema de arquivo
ifconfig	Mostra o estado das interfaces correntemente definidas
awk	Linguagem para processamento de texto
date	Exibe informações de hora e data
pidstat	É usado para monitorar tarefas individuais
mpstat	Relatar estatísticas relacionadas as CPUs
/proc	Contém informações sobre a execução do sistema Linux
sed	Editor para filtrar e transformar texto

5.2.2 SystemTap

O SystemTap é uma ferramenta de código aberto desenvolvida para monitorar as atividades do *kernel* de sistemas baseados em Linux, oferecendo informações detalhadas para análises de desempenho e detecção de erros [14, 35]. Foi criada em uma parceria que incluiu empresas como IBM, Intel, Hitachi, Oracle e Red Hat [17, 39]. O SystemTap possui uma linguagem própria, similar à linguagem C. O usuário pode criar *scripts* que verifiquem eventos específicos do sistema, com segurança e possibilidade de reutilização

dos testes em outras situações [39]. Na maioria das distribuições Linux, o SystemTap pode ser instalado através de repositórios.

O funcionamento do SystemTap consiste em rastrear os eventos denominados *probe points*, que podem ser a chamada ou o retorno de uma função, recebimento de pacotes de rede, ou mesmo contadores. Pode-se fazer *probe points* em qualquer função do kernel que não seja *inline* [24]. Quando o evento é disparado, chama-se o *handler* (manipulador) que coleta e mostra os dados desejados. Quando o manipulador termina sua execução, o *kernel* volta a sua rotina normal [39, 17]. O fluxo de execução do SystemTap pode ser visto na Figura 5.4.

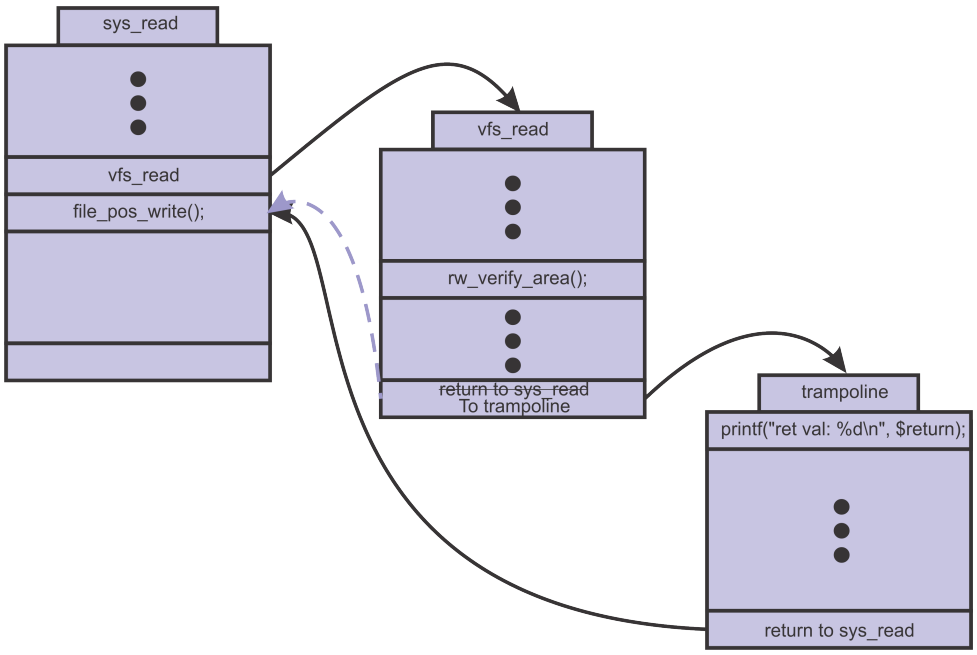


Figura 5.4: Fluxo de execução do SystemTap, adaptado de [17]

A Figura 5.5 exemplifica a lógica de execução do SystemTap. O código passa por um processo de elaboração, onde são verificadas as referências a outras bibliotecas; depois é traduzido para a linguagem C e logo em seguida é compilado como um módulo do *kernel*. Posteriormente, este módulo é carregado no *kernel*, dando início à coleta dos dados. Após o período de coleta, o módulo é descarregado e os dados são enviados para a saída padrão [39]. Abaixo um exemplo de script do SystemTap:

```
# stap -ve 'probe begin { log("hello world") exit () }'
```

Durante o experimento do cenário #1, monitorou-se o *trace point mm_page_alloc_extfrag* que se refere à 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 [21]. O *script* utilizado monitora quando este evento ocorre e incrementa uma variável sempre que a quantidade de memória liberada for menor que a

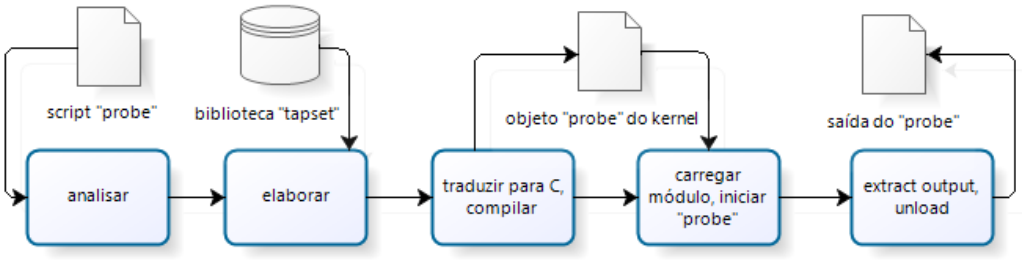


Figura 5.5: Passos de execução do SystemTap [35]

que está alocada. Decidiu-se que a cada 2 minutos a lista com os processos fragmentados e suas fragmentações no período são armazenados em arquivo. O mesmo ocorre quando a lista atinge seu limite de registros, limitação da estrutura de dados da linguagem da ferramenta.

5.3 Considerações Finais

Este capítulo descreveu a metodologia adotada para a realização deste trabalho. Foram descritas todas as etapas necessárias para a reprodução do trabalho, iniciando-se em atividades de estudo dos funcionamentos dos sistemas analisados, planejamentos de estratégias de monitoramento e geração de cargas de trabalho até a análise e comparação dos sistemas observados.

Capítulo 6

Estudos Experimentais

O atual capítulo descreve os estudos experimentais que foram realizados neste trabalho. Para atingir o objetivo de avaliar o desempenho da plataforma Docker no contexto de computação em nuvem, alguns cenários foram pensados para simular diferentes níveis de serviço. Cada uma das especificações desses cenários e métricas coletadas foram mostradas no capítulo anterior. O capítulo está organizado na Seção 6.1, que exibe os principais resultados dos experimentos realizados. Por fim, tem-se as considerações finais do capítulo.

6.1 Análise dos Resultados

Esta seção apresenta os resultados obtidos ao fim da execução de experimentos de cada cenário observado.

6.1.1 Resultados Cenário #1A

A partir das métricas coletadas, foram obtidos resultados significativos para os objetivos da pesquisa. Constatou-se ao analisar os dados gerais do computador que há indícios de exaustão de recursos no cenário analisado. Nesta etapa, como descrito na Seção 5.1, foi executada uma carga de trabalho simulando ações comuns ao modelo de serviço IaaS. Isto é, a carga de trabalho simulou a criação, inicialização, reinicialização e remoção de VMs (contêineres) em um hospedeiro físico.

Recursos Gerais

As Figuras 6.1 e 6.2 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 *swap*. Vê-se na Figura 6.1

que a memória utilizada no hospedeiro atinge seu ápice ao fim de pouco mais de 200 horas de execução da carga de trabalho. Esse momento coincide com o início da utilização

de memória *swap*, evidenciado na Figura 6.2.

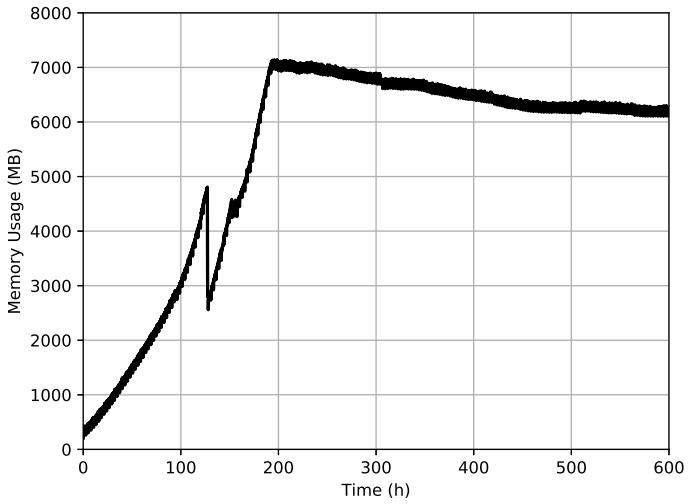


Figura 6.1: Utilização de memória principal pelo hospedeiro no cenário #1A

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.

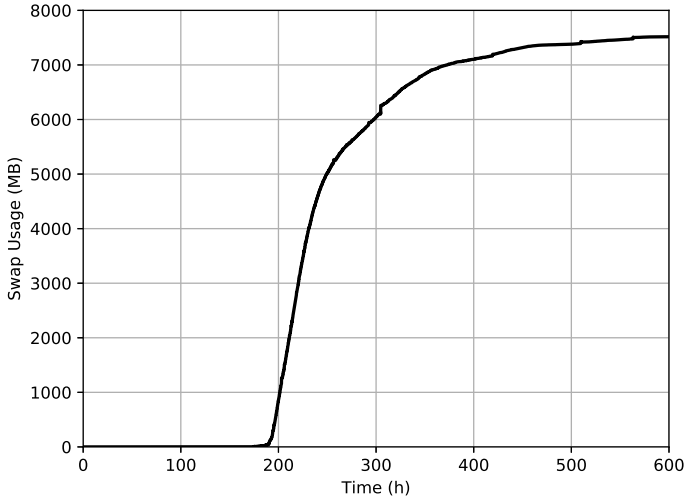


Figura 6.2: Utilização de memória *Swap* pelo hospedeiro no cenário #1A

A Figura 6.3 exibe o grau de utilização de CPU em função do tempo. Observa-se no gráfico que a utilização de CPU no computador que está hospedando a plataforma não ultrapassou 4% durante toda a execução do experimento. Porém, observa-se no gráfico uma tendência de crescimento na utilização do recurso, o que indica que a eficiência

do sistema para realizar uma mesma carga de trabalho está diminuindo, fato esse não verificado nos trabalhos relacionados consultados.

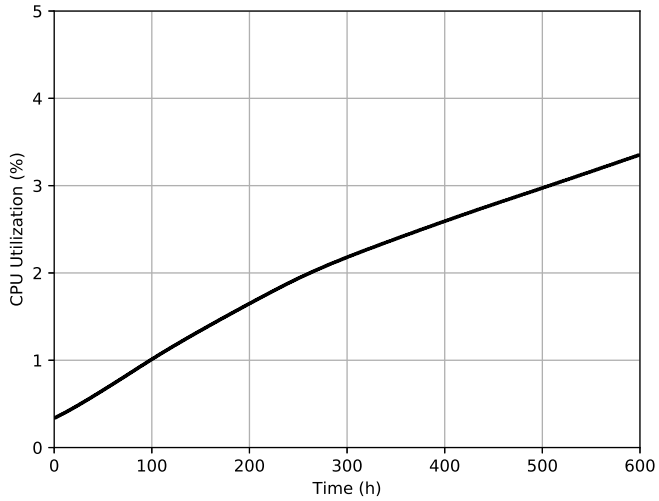


Figura 6.3: Utilização de CPU pelo hospedeiro no cenário #1A

Recursos Específicos

Os principais processos relacionados ao Docker não apresentaram indícios de degradação de desempenho. A Figura 6.4 mostra a utilização de memória residente pelo processo *dockerd* ao longo do experimento.

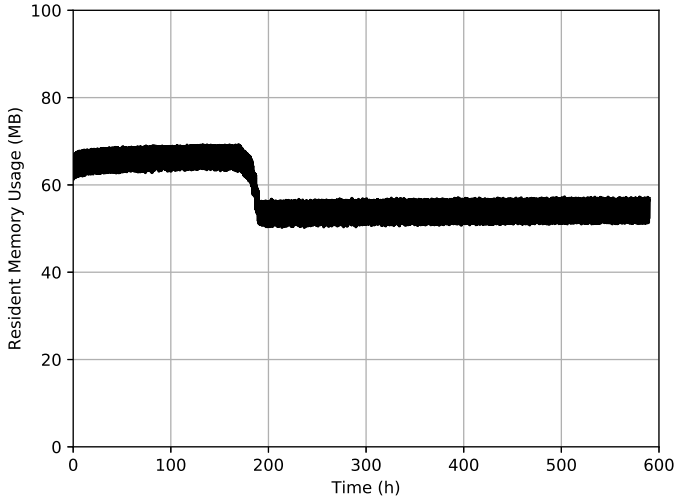


Figura 6.4: Utilização de memória residente pelo processo *dockerd* no cenário #1A

O uso de memória residente varia dentro de uma faixa que vai de pouco mais de 60

MB até 70 MB, havendo um decaimento para valores abaixo de 60 MB no momento em que a memória *swap* começa a ser utilizada. Já a utilização de memória virtual, vista na Figura 6.5, mostra uma utilização constante durante todo o período monitorado, sendo pouco inferior a 1400 MB. O consumo de recursos como CPU e número de *threads* são apresentados nas Figuras 6.6a e 6.6b. A utilização de CPU pelo processo permaneceu inferior a $12,5\%$, sem picos de de utilização que indiquem uma tendência de mudança no seu atual comportamento. Os recursos observados para o processo não apresentaram variações significativas que impactassem o desempenho.

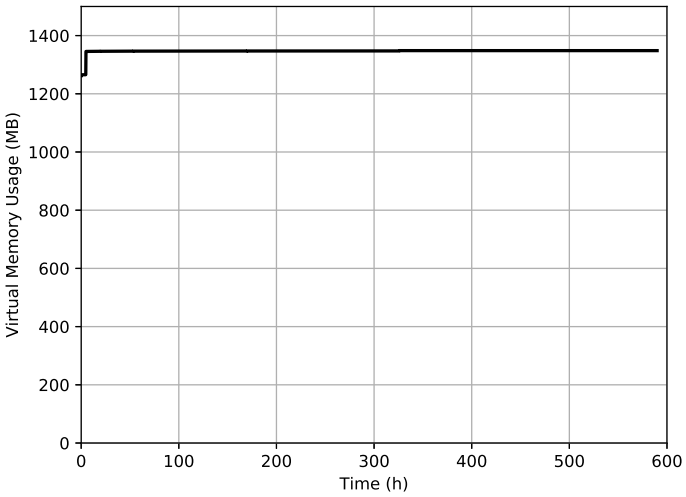
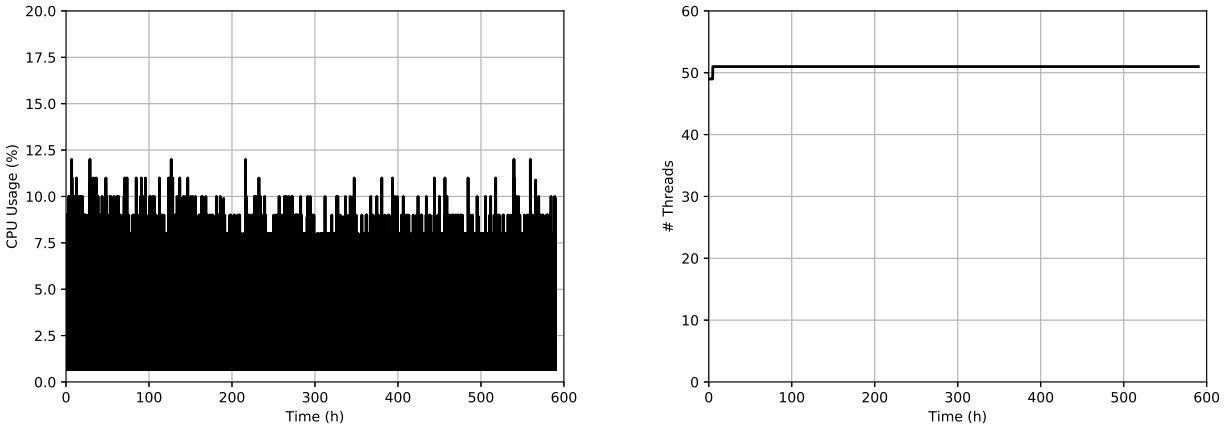


Figura 6.5: Utilização de memória virtual pelo processo *dockerd* no cenário #1



(a) *CPU* (b) *Threads*

Figura 6.6: Utilização de CPU e *threads* pelo processo *dockerd* no cenário #1A

Comportamento similar pode ser visto quando observado o processo *containerd*. A

Figura 6.7 mostra a utilização de memória residente pelo processo. Ao longo do experimento, o uso desse recurso permaneceu abaixo de 20 MB. A utilização de memória virtual, mostrado na Figura 6.8, exibe uma utilização próxima de 600 MB durante as primeiras 300 horas, aumentando para para 650 MB após esse período e assim permanecendo até o encerramento do experimento. O comportamento de uso de CPU e número de *threads* utilizadas são exibidos nas Figuras 6.9a e 6.9b.

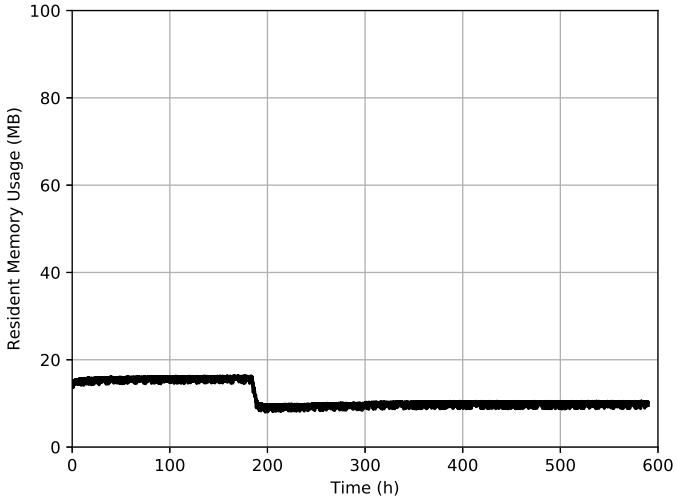


Figura 6.7: Utilização de memória residente pelo processo *containerd* no cenário #1A

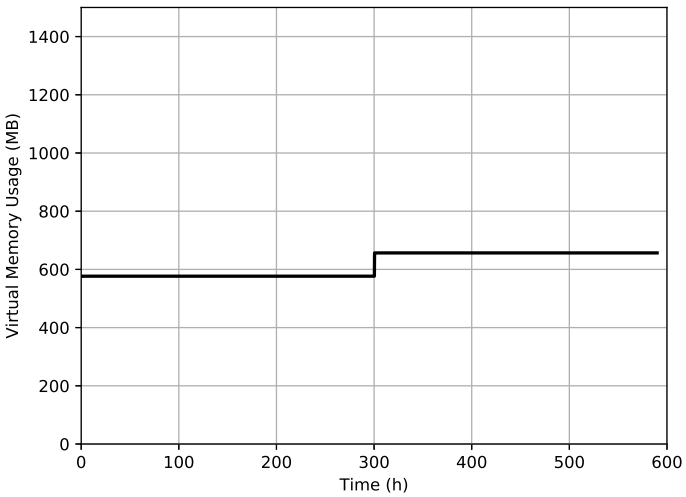
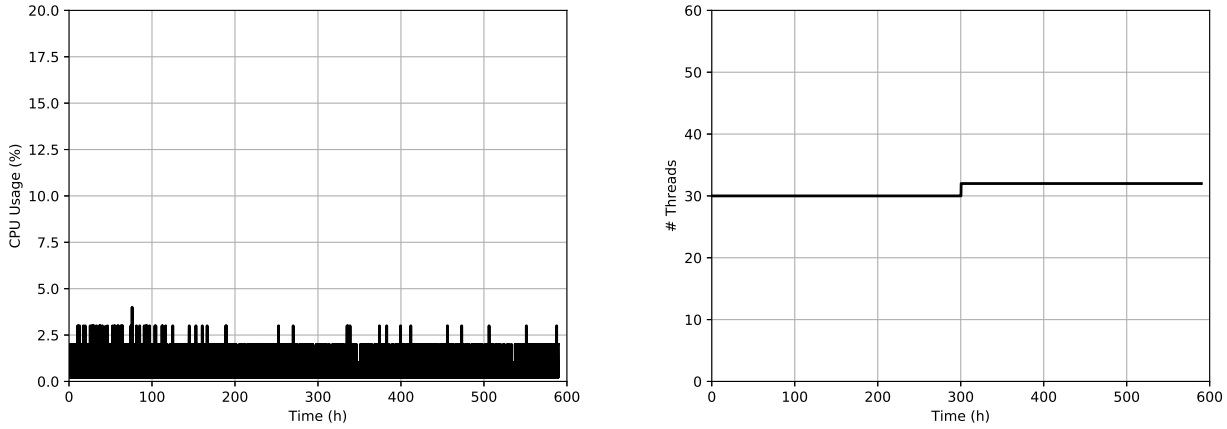


Figura 6.8: Utilização de memória virtual pelo processo *containerd* no cenário #1A

Pode-se notar que a contribuição do Docker não é significativa para o comportamento do geral observado nos recursos gerais do computador que hospeda a plataforma. O



(a) *CPU* (b) *Threads*

Figura 6.9: Utilização CPU e *threads* pelo processo *containerd* no cenário #1A

processo que provoca a 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 [34].

A Figura 6.10 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 alcança a marca de 100% de utilização. Esse momento ocorre no instante em que o SO começa a gerenciar memória *swap*.

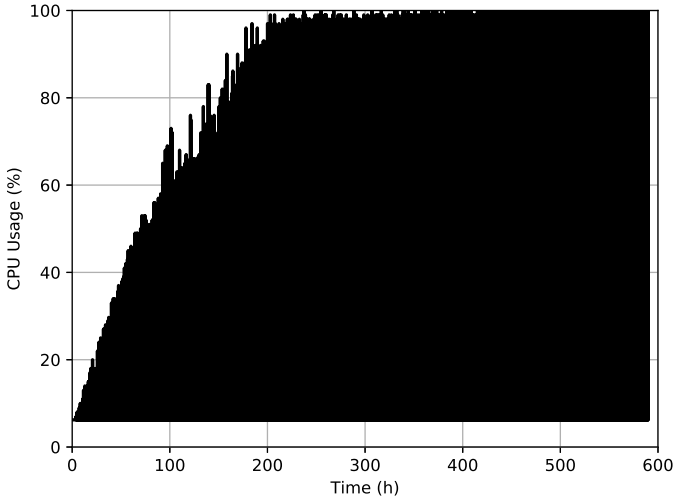
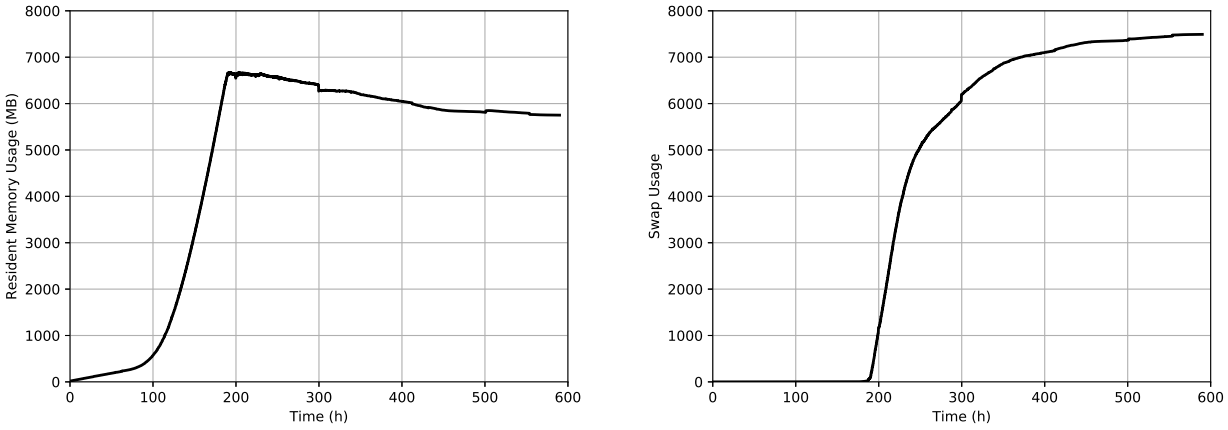


Figura 6.10: Utilização de CPU pelo processo *NetworkManager*

As Figuras 6.11a e 6.11b mostram a utilização de memória residente e virtual pelo processo *NetworkManager*. Nota-se o aumento contínuo de memória, utilizando-se aproximadamente 14000 MB (RAM + *swap*). Constata-se que há indícios de vazamento de

memória relacionada ao processo, problema relacionado a *bugs* em situações onde realiza-se alocação de espaços de memória não ocorrendo a liberação quando o espaço não é mais necessário, já que o processo aloca memória de forma crescente. Esse consumo demasiado não está relacionado ao fenômeno da fragmentação de memória, já que, como visto na Tabela 6.1 e na Figura 6.12, o processo não apresenta mais que $11,658$ ocorrências de fragmentação ao longo do tempo observado.



(a) Residente (b) Virtual

Figura 6.11: Utilização de Memória pelo processo *NetworkManager*

As ocorrências de fragmentação de memória podem comprometer o bom funcionamento do sistema de duas maneiras [29]. A primeira consiste no aumentando da quantidade de memória utilizada pelo processo, já que o SO pode atender requisições de alocação para novos blocos de memória, mesmo não havendo memória contínua disponível. Essa memória não será liberada antes da finalização do processo. A segunda, implica em complexidade adicional no gerenciamento de memória pelo SO. Na Figura 6.12 visualiza-se a acumulação de ocorrências de fragmentação de memória ao longo do tempo.

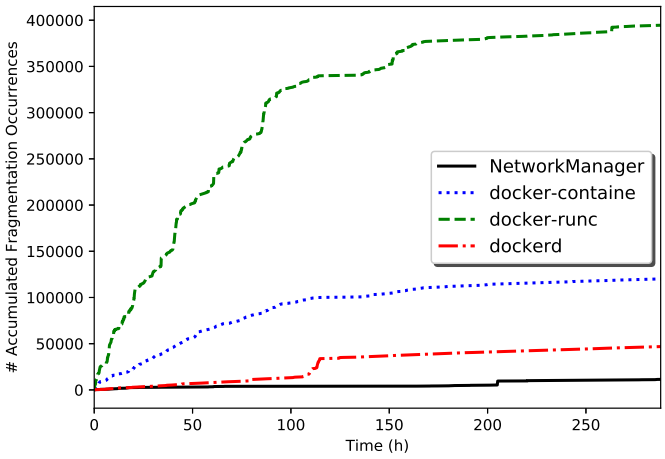


Figura 6.12: Acumulativo de fragmentação de memória por processo no cenário #1A

Já na Tabela 6.1, são mostrados os totais obtidos por processo ao final de 12 dias de monitoramento. O monitoramento de fragmentações foi encerrado ao fim desse período devido ao tamanho dos arquivos de *logs*. Observa-se que o processo com maior número de ocorrências é o *docker-runc*, seguido por *docker-containerd*, *dockerd* e *NetworkManager*.

Tabela 6.1: Total de Ocorrências de Fragmentação por Processo no cenário #1A

Processo	**Fragmentação Total**
docker-runc	395,086
Docker-containe	120,609
dockerd	47,393
NetworkManager	11,658

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 tem um ciclo de vida de 2 horas, sendo os recursos liberados para o início de uma nova rodada.

6.1.2 Resultados Cenário #1B - Sem NetworkManager

O objetivo para este estudo foi verificar o comportamento de utilização de recursos pelo Docker em um ambiente onde o *NetworkManager* não esteja presente. Como visto no cenário #1A, ao aplicar a carga de trabalho ao Docker, o utilitário *NetworkManager* apresentou problemas no seu gerenciamento de memória. A mesma carga de trabalho foi mantida e os mesmos recursos foram monitorados.

Recursos Gerais

A utilização de memória RAM do servidor hospedeiro ao longo do tempo de execução deste cenário é mostrada na Figura 6.13. Vemos que a quantidade de memória em uso cresce linearmente até atingir 4420 MB as 190 horas de experimento, quando decai para 1400 MB. Esse padrão de consumo continua com uma tendência de diminuição do limite superior até o momento em que se estabiliza por volta das 500 horas de experimentação. O padrão serrilhado que visualiza-se no gráfico mostrado na Figura 6.13 não pode ser explicado pelo consumo de recursos dos processos monitorados.

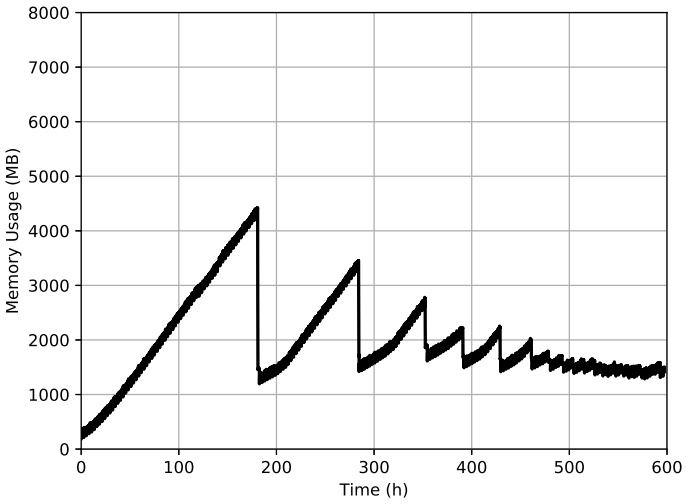


Figura 6.13: Utilização de memória principal pelo hospedeiro no cenário #1B

O consumo de CPU durante todo o experimento permaneceu pouco acima de 1% de utilização, não havendo indicação de degradação de desempenho.

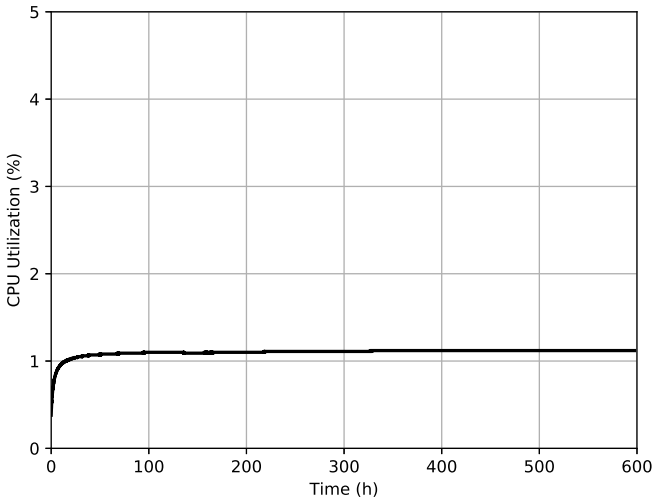
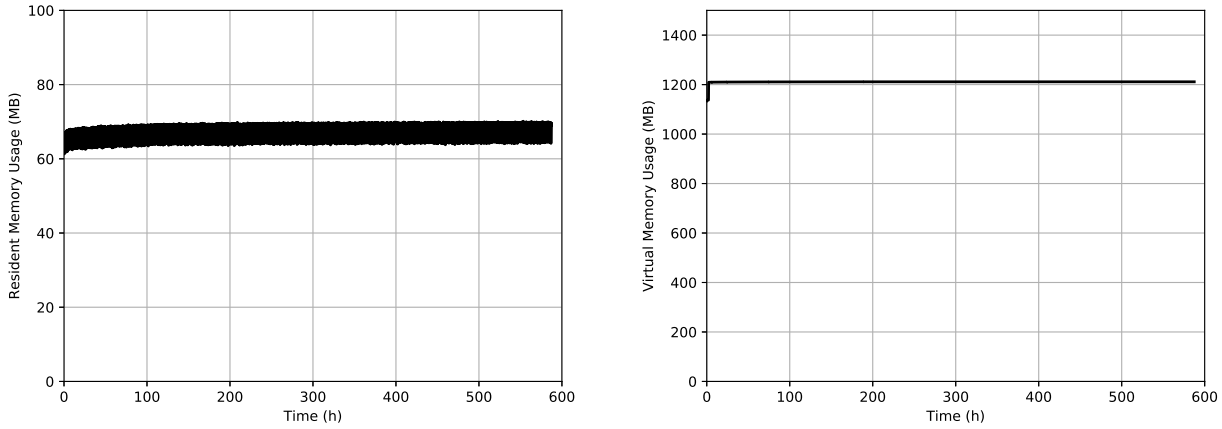


Figura 6.14: Utilização geral de CPU no hospedeiro no no cenário #1B

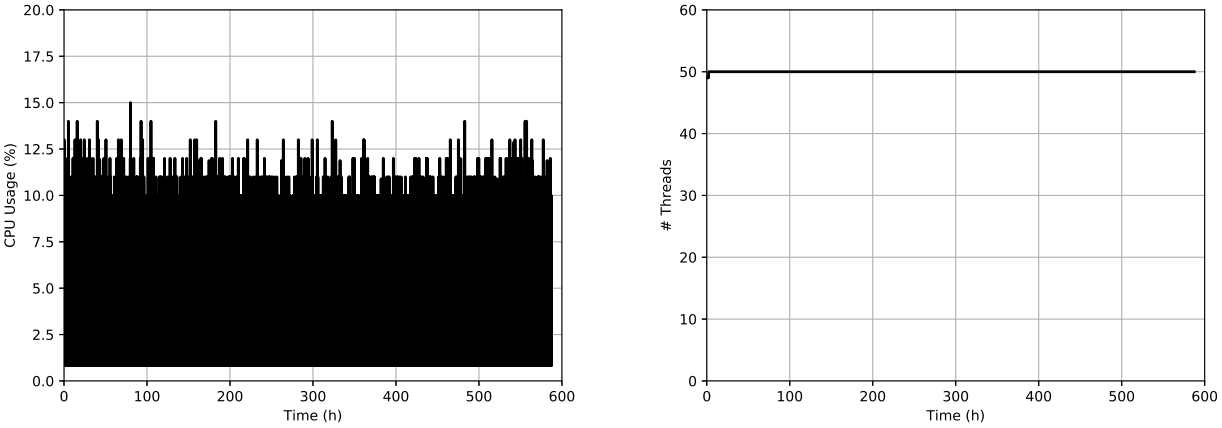
Recursos Específicos

O consumo de memória residente pelo processo *dockerd* é mostrado na Figura 6.15a. Nele vemos que a utilização de memória pode ser considerada constante, variando dentro de uma faixa pequena de valores, que vai de 65 MB até 70 MB. A memória virtual, mostrada na Figura 6.15b permaneceu constante durante todo o período observado. A utilização de CPU mostrada na Figura 6.15c não ultrapassou a marca de 15% de utilização e não demostrou tendências de crescimento.



(a) Memória Residente

(b) *Memória Virtual*



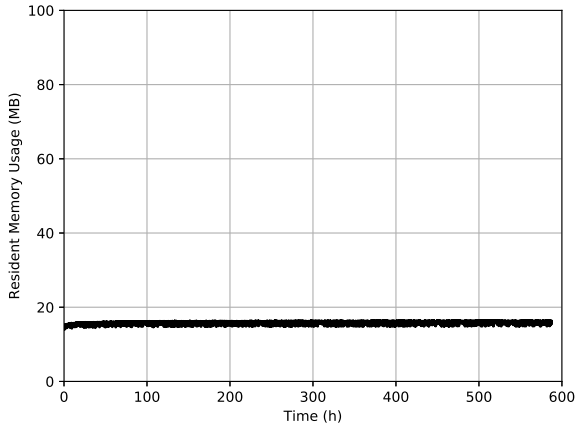
(c) *CPU*

(d) *Threads*

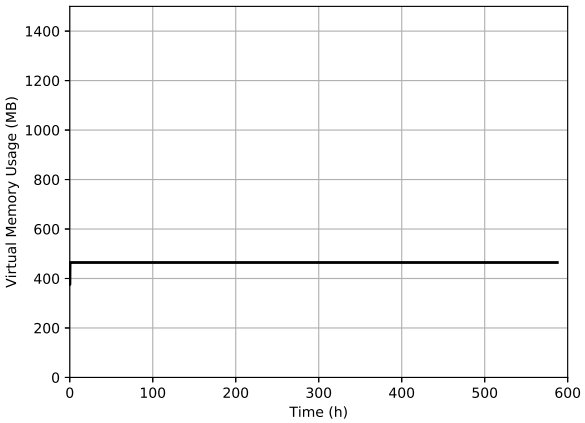
Figura 6.15: Utilização recursos pelo processo *dockerd* no no cenário #1B

O processo *containerd* apresentou comportamento similar ao *dockerd*. Os resultados obtidos para esse processo podem ser vistos na Figura 6.16. Nenhum dos recursos observados apresentou indícios de vazamentos de recursos ou tendências de sobrecarga no sistema.

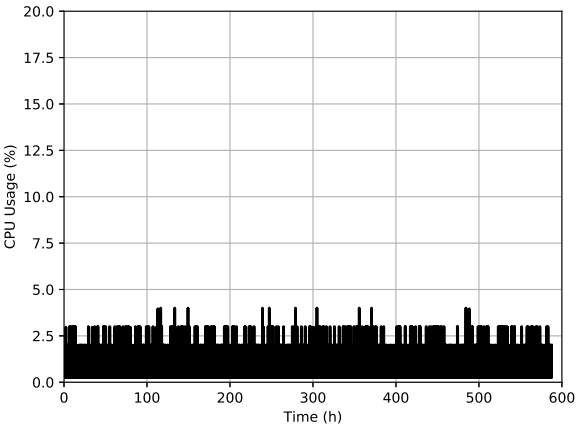
Neste cenário, assim como no anterior, processos relacionados ao Docker sofreram fragmentação. Os resultados mostrados na Figura 6.17 consideraram os primeiros 12 dias de experimentação. Nele vemos que o processo que mais fragmentou foi o *dockerd*, seguido pelo *docker-runc* e *docker-containerd*.



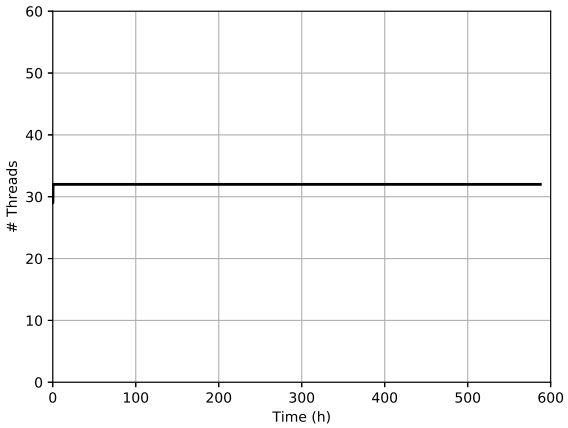
(a) Memória Residente



(b) *Memória Virtual*



(c) *CPU*



(d) *Threads*

Figura 6.16: Utilização recursos pelo processo *containerd* no cenário #1B

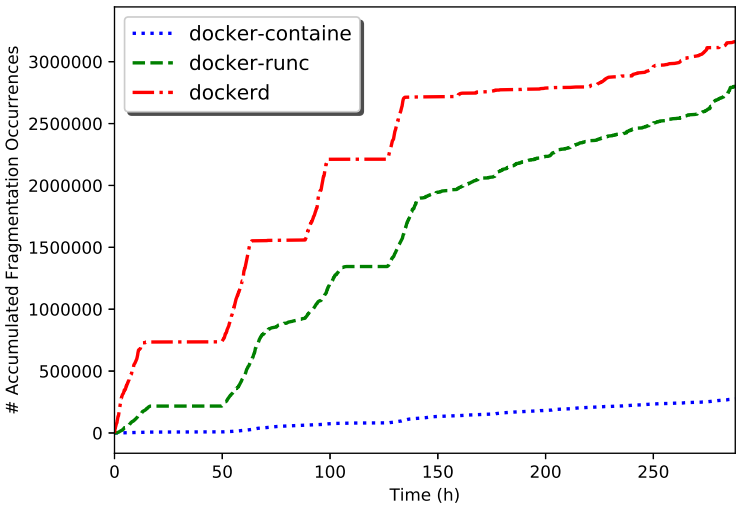


Figura 6.17: Acumulativo de fragmentação de memória por processo no cenário #1B

O processo que apresentou maior quantidade de fragmentações nesse cenário foi o *dockerd* totalizando um acúmulo de 3278616 fragmentações, como pode ser visto na Tabela 6.2 .

Tabela 6.2: Total de Ocorrências de Fragmentação por Processo no cenário #1B

Processo	**Fragmentação Total**
docker-runc	2,841,002
Docker-containe	283,835
dockerd	3,278,616

6.1.3 Resultados Cenário #2

O objetivo deste cenário foi verificar o impacto da utilização do Docker com modo *swarm* ativo.

Recursos Gerais

A Figura 6.18 mostra o consumo de memória pelo *host Worker*. Vemos que ocorreu um padrão similar ao que foi mostrado na Figura 6.13, no estudo do cenário #2. Houve um crescimento linear na utilização de memória durante as primeiras 90 horas, quando o atingiu-se o ápice em 3900 MB. Neste momento ouve um decréscimo no consumo ficando pouco maior que 1500 MB, onde um novo ciclo de crescimento com o mesmo padrão de queda se inicia. Por volta de 320 horas de execução, há certa estabilização do uso de memória em torno de 2000 MB. O mesmo ocorre com a memória do hospedeiro *Worker*,

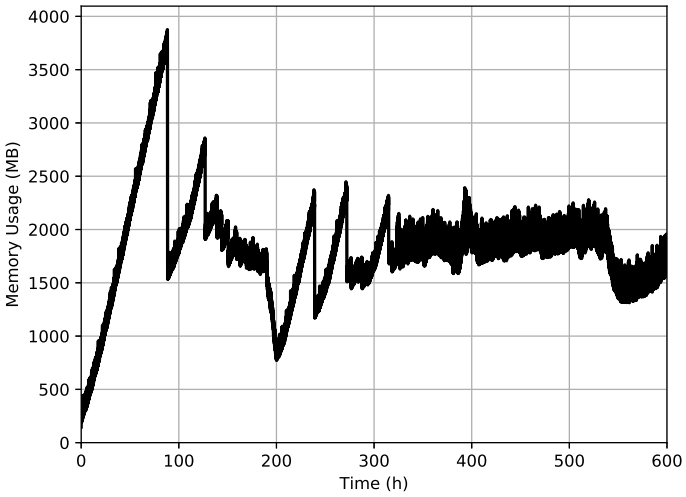


Figura 6.18: Utilização de memória no hospedeiro *Master*

como mostrado na Figura 6.19.

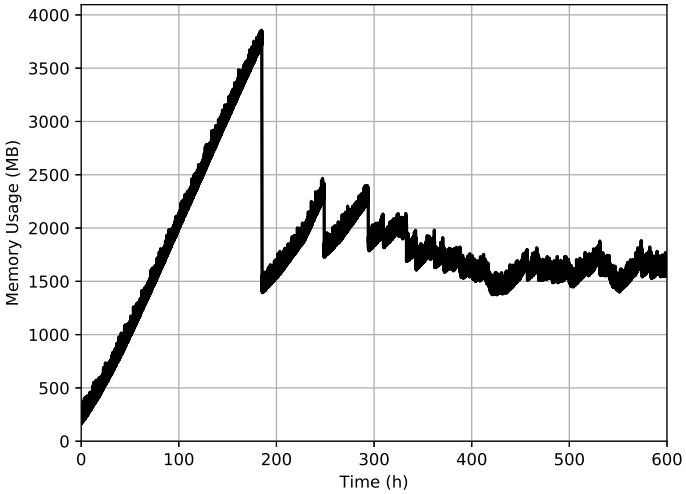


Figura 6.19: Utilização de memória no hospedeiro *Worker*

O uso de processamento permanece abaixo de 4% ao longo de todo o tempo observado. Porém, como pode ser visto na Figura 6.20, desde as horas iniciais há tendência de crescimento linear em sua utilização. Essa tendência acentua-se aproximadamente as 320 horas, onde pode-se ver que a taxa de crescimento na utilização do recurso aumenta.

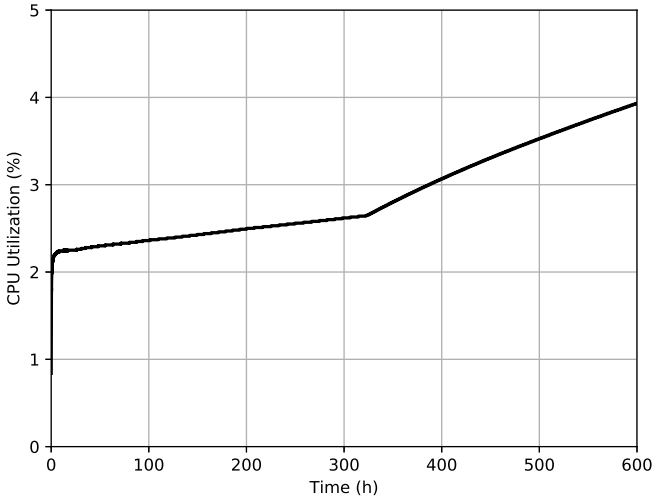


Figura 6.20: Utilização de CPU no hospedeiro *Master*

No hospedeiro *Worker* a CPU ficou abaixo de 2% de utilização, também apresentando indícios de crescimento. O gráfico desse recurso pode ser visto na Figura 6.21.

O aumento progressivo de utilização de CPU é um indicador de que o sistema está em processo de degradação do desempenho. Mesmo que a utilização total de CPU tenha

permanecido abaixo de 4% em ambos os hospedeiros, em algum momento futuro sua utilização alcançará 100%. A capacidade de processamento, por maior que ela seja, é um recurso limitado.

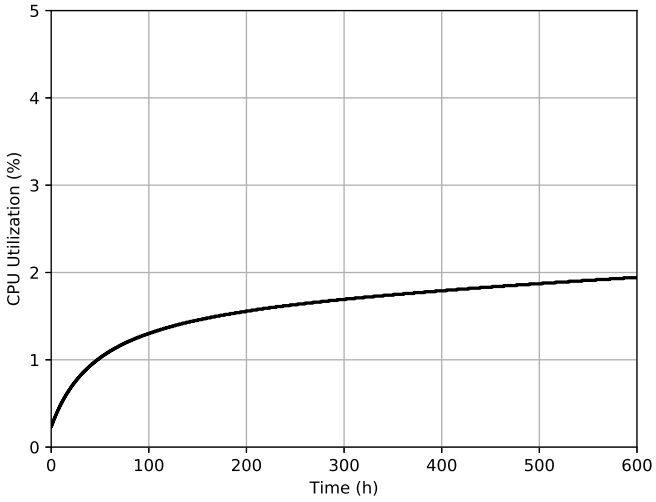


Figura 6.21: Utilização de CPU no hospedeiro Worker

Recursos Específicos

O processo *dockerd* apresentou um consumo crescente de memória residente, com algumas variações sazonais. Como pode ser visto na Figura 6.22, a utilização do recurso seguiu um padrão linear de crescimento aumentando sua utilização em mais de 300% no período de observação.

A Tabela 6.3 mostra as estimativas de consumo de memória residente pelo processo dockerd para os próximos meses. Os dados foram obtida por meio de uma regressão linear $frmd(t) = 24,43512 \cdot t + 56232,6646$ que descreve o comportamento dos dados em função do tempo t. A escolha do método de regressão para descrever o consumo se deu por questões de simplicidade, mas outras variáveis podem ser utilizadas para melhora-lo. De acordo com a Tabela 6.3, ao fim de 3 meses o uso de memória residente pelo processo *dockerd* seria de aproximadamente 3190 MB.

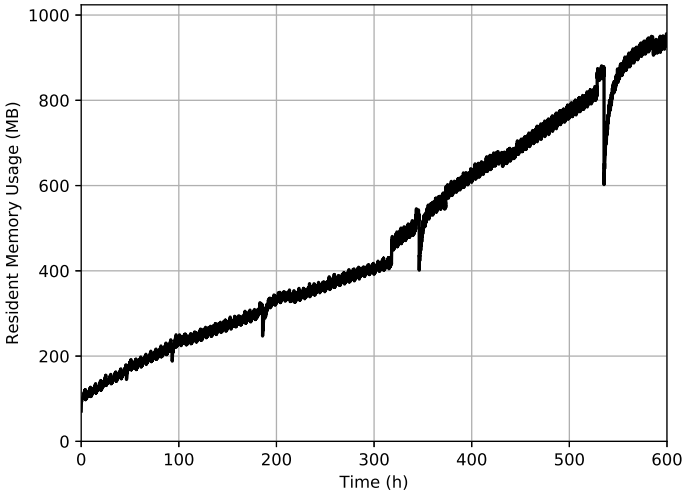


Figura 6.22: Utilização de memória residente pelo processo *dockerd* no hospedeiro *Master*

Tabela 6.3: Estimativa de consumo de memória residente do processo *dockerd* no hospedeiro *Worker*

Mês	**Estimativa (MB)**
1	1100, 0888
2	2145.2629
3	3190.4369
4	4235.6110
5	5280.7851
6	6325.9592
7	7371.1333
8	8416.3073

A memória virtual do processo *dockerd*, exibida pela Figura 6.23, também apresentou tendência de crescimento, dobrando sua utilização ao longo da experimentação.

A utilização de CPU pelo processo *dockerd* segue o mesmo padrão de crescimento, em sintoma de degradação de recursos. A Figura 6.24 exibe o gráfico do uso de CPU pelo processo. Vê-se que a utilização durante considerável tempo permanece abaixo de 20%, mas com picos crescentes que indicam uma evolução no uso.

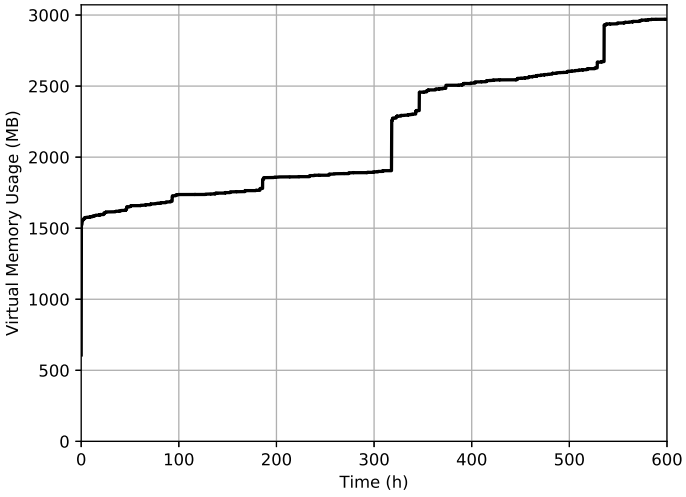


Figura 6.23: Utilização de memória virtual pelo processo *dockerd* no hospedeiro *Master*

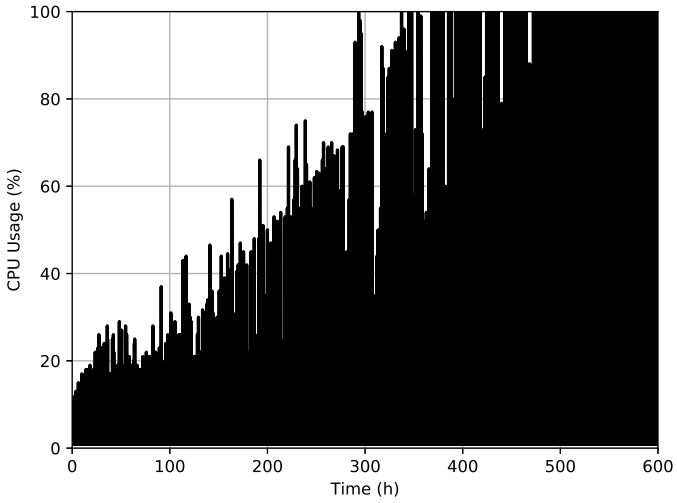


Figura 6.24: Utilização de CPU pelo processo *dockerd* no hospedeiro *Master*

O uso de CPU fica em média abaixo de 20%, mas é possível notar picos de utilização indicando uma propensão ao aumento dessa utilização no futuro. O comportamento observado é análogo ao que foi visto no hospedeiro *Master*.

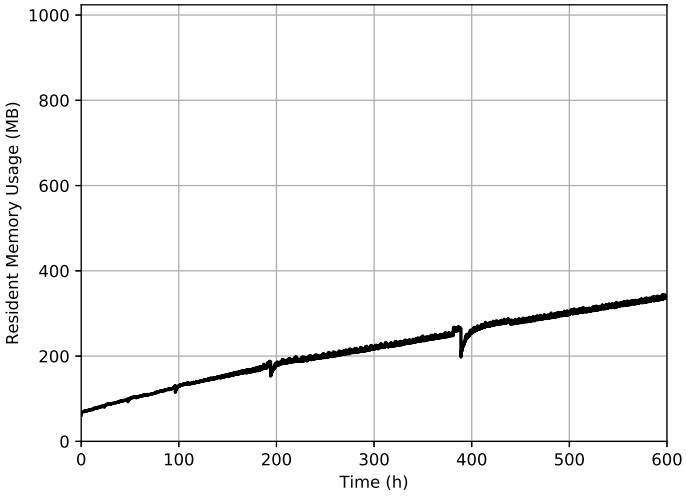


Figura 6.25: Utilização de memória residente pelo processo *dockerd* no hospedeiro *Worker*

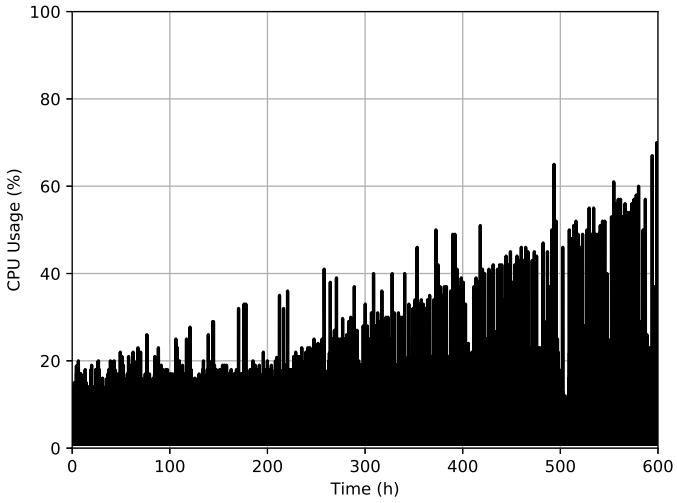


Figura 6.26: Utilização de CPU pelo processo *dockerd* no hospedeiro *Worker*

Nota-se que os recursos utilizados pelo processo *dockerd* no hospedeiro *Worker* foram consideravelmente inferiores aos que foram utilizados pelo mesmo processo no hospedeiro *Master*. Uma *possível* explicação para esse fato pode ser a tarefa adicional de gerenciamento do *cluster swarm* que é delegada ao hospedeiro *Master*.

Esgotamento de recursos estão relacionados na literatura como uma das causas de degradações de desempenho de sistemas. Esse comportamento está relacionado com o que na literatura é chamado de envelhecimento de *software* [16], fenômeno que aumenta as probabilidades de falha do sistema diminuindo suas métricas de dependabilidade [11].

6.2 Considerações Finais

Neste capítulo foram mostrados os principais resultados obtidos nessa pesquisa. Os resultados dos estudos experimentais realizados foram demonstrados para cada um dos cenários avaliados. Foram realizados 3 experimentos em 2 cenários avaliados: no primeiro simulou-se a utilização da plataforma para um ambiente de infraestrutura como um serviço, onde máquinas virtuais são criadas, reinicializadas e finalizadas ciclicamente; no segundo avaliou-se a plataforma no seu modo *cluster*, simulando um ambiente onde os contêineres que hospedam o serviço são escalonados de forma horizontal.

Os resultados mostram que no primeiro cenário avaliado, o ambiente sofre degradação no desempenho quando o utilitário NetworkManager está presente, impactando as operações da plataforma. Esse não é um aplicativo relacionado ao Docker e, portanto, uma nova rodada de destes foi realizada com o recurso desabilitado. Não foram observados alterações significativas que indicassem que o sistema está deteriorando sua performance.

Já no segundo cenário, tanto no hospedeiro que controla o cluster quanto no nó que fornece o espaço para que os contêineres sejam hospedados (*Master* e *Worker*) houveram indícios de que os recursos estão sendo esgotados pelo processo *dockerd*.

Capítulo 7

Conclusões

Neste trabalho foram realizadas experimentações para caracterizar o consumo de recursos pela plataforma Docker, um sistema de virtualização leve para provisionamento de contêineres. Foram avaliados cenários de uso intensivo da infraestrutura virtualizada, utilizando cargas de trabalho sintéticas. Esses cenários foram pensados para representar situações semelhantes ao que se teria em ambientes de computação em nuvem. Realizou-se dois casos de uso: primeiro, realizou-se 2 experimentos utilizando-se uma máquina e realizando-se ações de criação, reinicialização e finalização de contêineres; utilizando-se a funcionalidade nativa do Docker para criação de *clusters* de serviços.

Os resultados obtidos no primeiro cenário de testes com o utilitário NetworkManager, pode afetar o bom funcionamento do sistema e esgotar os recursos da máquina quando presente no mesmo ambiente que o Docker. Isso aumenta a possibilidade de falha da plataforma Docker mesmo não havendo relação entre os processo pois, como os recursos de memória estão sendo vazados, em um certo ponto a quantidade de memória livre disponível será insuficiente para o Docker realizar novas instanciações de contêineres. Observou-se que o tempo necessário para que toda a memória disponível fosse esgotada foi de aproximadamente 25 dias para a carga de trabalho utilizada.

Em um novo experimento, com o NetworkManager desabilitado, viu-se que o Docker apresenta boa gestão de recursos quando não está em modo *cluster*, sendo uma plataforma estável. A utilização de recursos pelos principais processos do Docker permaneceram variando dentro de faixas bem definidas, sem indicações de tendências de crescimento.

Foi observado a utilização de recursos pelo Docker com seu modo Swarm ativo. O Swarm é um recurso nativo do Docker para criação de *clusters* de serviços de alta disponibilidade. Nesse cenário verificou-se que o processo *dockerd* apresentou crescimento no consumo de memória em ambos os computadores (*Master* e *Worker*), com tendência linear. Uma previsão foi realizada baseando-se em regressão linear para prever o consumo de memória futuro, mostrando que em três meses a memória utilizada pelo processo *daemon* do Docker deve alcançar valores acima de 3000 MB. Isso representa um consumo de

mais de ⅓ da memória principal no ambiente utilizado como hospedeiro *Master*. Pode-se considerar essse consumo preocupante, já que trata-se de um único processo. O uso de CPU também apresentou propensão de crescimento, com picos de utilização que chegam a 100% de utilização no *Master* e 70% no *Worker*.

7.1 Contribuições

Esse trabalho possui as seguintes contribuições para o estado da arte:

- Caracteriza o consumo de recursos de servidores hospedeiros, evidenciando esgotamento de memória do hospedeiro quando a carga de trabalho é executada com a utilização do NetworkManager, mas que esse comportamento não é percebido quando o utilitário está desativado. Mostrou-se que há um aumento não explicado no consumo de memória nas horas iniciais do experimento, mas que há estabilização em horas posteriores. Verificou-se que a utilização de CPU tem uma tendência de crescimento no segundo cenário avaliado. Isso corresponde a conclusão do primeiro objetivo específico proposto;

- Mostra que o Docker apresenta pouco *overhead* também no seu gerenciamento de contêineres quando não está em modo *Swarm*, correspondendo ao objetivo de caracterizar o consumo de recursos da plataforma;

- Caracteriza o consumo de recursos com o modo Swarm ativo, apresentando indícios de degradação de desempenho;

7.2 Limitações

Este trabalho não verificou outras métricas, tais como tempo de resposta, relacionadas aos serviços hospedados nos contêineres. Essas poderiam ser medidas importantes para verificar se há interferências que impactem a qualidade do serviço, ocasionadas pela natureza da carga de trabalho adotada. Nesse trabalho apenas uma tecnologia de virtualização baseada em contêineres foi avaliada, impossibilitando a comparação com outros sistemas em situações equivalentes de *stress*.

Outro ponto que limitante é que não se monitorou os consumos de recursos pelos contêineres em si, já que esses são destruídos periodicamente e uma vez que o foco do trabalho era saber como o Docker lida com intensivas requisições de escalonamento.

7.3 Trabalhos Futuros

Como trabalhos futuros planeja-se a realização de novos experimentos monitorando-se os recursos utilizados pelos contêineres e monitorando-se outras métricas de rede (como vazão e tempo de resposta) relacionadas aos serviços hospedados na infraestrutura virtualizada. Considerar o desempenho de outras soluções existentes também é importante para fins de comparação com o estudo realizado.

Bibliografia

[1] Amaral, M. ; Polo, J. ; Carrera, D. ; Mohomed, I. ; Unuvar, M. ; Steinder, M. Performance evaluation of microservices architectures using containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, (2015), pp. 27–34.

[2] Amri, A. E. *Docker, Containerd and Standalone Runtimes: Here's What You Should Know*. `https://link.medium.com/npAbiNY1XX`, 2017. – Acessado em : 26/05/2019.

[3] Beserra, D. ; Moreno, E. D. ; Endo, P. T. ; Barreto, J. Performance evaluation of a lightweight virtualization solution for HPC I/O scenarios. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, (2016), pp. 004681–004686.

[4] Beserra, D. ; Moreno, E. D. ; Endo, P. T. ; Barreto, J. ; Sadok, D. ; Fernandes, S. Performance analysis of LXC for HPC environments. In *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, (2015), pp. 358–363.

[5] Beserra, D. ; Pinheiro, M. K. ; Souveyet, C. ; Steffenel, L. A. ; Moreno, E. D. Performance evaluation of os-level virtualization solutions for hpc purposes on soc-based systems. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, (2017), pp. 363–370.

[6] Blum, R. *Linux command line and shell scripting bible*. 481. John Wiley & Sons, 2008.

[7] Buyya, R. ; Broberg, J. ; Goscinski, A. M. *Cloud computing: Principles and paradigms*. 87. John Wiley & Sons, 2010.

[8] Carter, E. *2018 Docker Usage Report*. `https://sysdig.com/blog/2018-docker-usage-report/`, 2018. – Acessado em : 25/03/2019.

[9] Celesti, A. ; Mulfari, D. ; Fazio, M. ; Villari, M. ; Puliafito, A. Exploring container virtualization in IoT clouds. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, (2016), pp. 1–6.

[10] Cisco *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022 White Paper.* https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html#_Toc953333, 2019. – Acessado em: 26/05/2019.

[11] Cotroneo, D. ; Natella, R. ; Pietrantuono, R. ; Russo, S. A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 10, 1 (2014), 8.

[12] Docker *The Industry-Leading Container Runtime.* https://www.docker.com/products/container-runtime, 2019. – Acessado em : 26/05/2019.

[13] Docker *What is a Container?* https://www.docker.com/resources/what-container, 2019. – Acessado em : 26/03/2019.

[14] Eigler, F. C. Problem Solving With Systemtap. *Proceedings of the Linux Symposium* 1 (2006), 261–268.

[15] Fayyad-Kazan, H. ; Perneel, L. ; Timmerman, M. Full and para-virtualization with Xen: a performance comparison. *Journal of Emerging Trends in Computing and Information Sciences* 4, 9 (2013), 719–727.

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

[17] Jacob, B. ; Larson, P. ; Leitão, B. H. ; Silva, S. A. M. M. *Systemtap: Instrumenting the Linux kernel for analyzing performance and functional problems.* IBM Redbook, 2009. ISSN 1071–9458.

[18] Jain, R. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.* Wiley, 1991 (Wiley professional computing). – I–XXVII, 1–685 S. – ISBN 978–0–471–50336–1.

[19] Jawarneh, I. M. A. ; Bellavista, P. ; Foschini, L. ; Martuscelli, G. ; Montanari, R. ; Palopoli, A. ; Bosi, F. QoS and performance metrics for container-based virtualization in cloud environments. In *Proceedings of the 20th International Conference on Distributed Computing and Networking*, (2019), pp. 178–182.

[20] Katzan Jr, H. et al. On an ontological view of cloud computing. *Journal of Service Science (JSS)* 3, 1 (2010).

[21] Kernel *Subsystem Trace Points: kmem.* https://www.kernel.org/doc/html/v4.18/trace/events-kmem.html, 2019. – Acessado em : 26/03/2019.

[22] Khazaei, H. ; Barna, C. ; Beigi-Mohammadi, N. ; Litoiu, M. Efficiency analysis of provisioning microservices. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, (2016), pp. 261–268.

[23] Kourai, K. ; Chiba, S. A fast rejuvenation technique for server consolidation with virtual machines. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, (2007), pp. 245–255.

[24] Leitão, B. Depuração de kernel com SystemTap. *Linux Magazine* 1, 67 (2010), 58–64.

[25] Lenk, A. ; Klems, M. ; Nimis, J. ; Tai, S. ; Sandholm, T. What's inside the Cloud? An architectural map of the Cloud landscape. In *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, (2009), pp. 23–31.

[26] Li, Z. ; Kihl, M. ; Lu, Q. ; Andersson, J. A. Performance overhead comparison between hypervisor and container based virtualization. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, (2017), pp. 955–962.

[27] Lilja, D. J. *Measuring Computer Performance: A Practitioner's Guide*. New York : Cambridge University Press, 2000. – ISBN 0–521–64105–5.

[28] Liu, B. ; Chang, X. ; Liu, B. ; Chen, Z. Performance Analysis Model for Fog Services under Multiple Resource Types. In *2017 International Conference on Dependable Systems and Their Applications (DSA)*, (2017), pp. 110–117.

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

[30] Matos, R. ; Araujo, J. ; Alves, V. ; Maciel, P. Characterization of software aging effects in elastic storage mechanisms for private clouds. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, (2012), pp. 293–298.

[31] Mell, P. ; Grance, T. et al. The NIST definition of cloud computing. (2011).

[32] Morabito, R. ; Kjällman, J. ; Komu, M. Hypervisors vs. lightweight virtualization: a performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, (2015), pp. 386–393.

[33] Nelson, W. B. A bibliography of accelerated test plans. *IEEE Transactions on Reliability* 54, 2 (2005), 194–197.

[34] NetworkManager *NetworkManager: NetworkManager Reference Manual*. `https://developer.gnome.org/NetworkManager/stable/NetworkManager.html`, 2019. – Acessado em : 18/02/2019.

[35] Oliveira, F. ; Costa, D. ; Vale, G. ; Beserra, D. ; Araujo, J. Fragmentação de Memória em Sistemas Linux. *V Escola Regional de Informática de Mato Grosso* (2014).

[36] Pahl, C. Containerization and the paas cloud. *IEEE Cloud Computing* 2, 3 (2015), 24–31.

[37] Piraghaj, S. F. ; Dastjerdi, A. V. ; Calheiros, R. N. ; Buyya, R. A framework and algorithm for energy efficient container consolidation in cloud data centers. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, (2015), pp. 368–375.

[38] Piraghaj, S. F. ; Dastjerdi, A. V. ; Calheiros, R. N. ; Buyya, R. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience* 47, 4 (2017), 505–521.

[39] Prasad, V. ; Cohen, W. ; C., E. F. ; Hunt, M. ; Keniston, J. ; Chen, B. Locating System Problems Using Dynamic Instrumentation. *In Linux Symposium* (2005).

[40] Preeth, E. ; Mulerickal, F. J. P. ; Paul, B. ; Sastri, Y. Evaluation of Docker containers based on hardware utilization. In *2015 International Conference on Control Communication & Computing India (ICCC)*, (2015), pp. 697–700.

[41] Rista, C. ; Teixeira, M. ; Griebler, D. ; Fernandes, L. G. Evaluating, Estimating, and Improving Network Performance in Container-based Clouds. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, (2018), pp. 00514–00520.

[42] Salah, T. ; Zemerly, M. J. ; Yeun, C. Y. ; Al-Qutayri, M. ; Al-Hammadi, Y. Performance comparison between container-based and VM-based services. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, (2017), pp. 185–190.

[43] Scheepers, M. J. Virtualization and containerization of application infrastructure: A comparison. In *21st twente student conference on IT* 1, (2014), pp. 1–7.

[44] Seo, K.-T. ; Hwang, H.-S. ; Moon, I.-Y. ; Kwon, O.-Y. ; Kim, B.-J. Performance comparison analysis of linux container and virtual machine for building cloud. *Advanced Science and Technology Letters* 66, 105-111 (2014), 2.

[45] Silva, F. H. R. Avaliação de desempenho de Contêineres Docker para aplicações do Supremo Tribunal Federal. (2017).

[46] Tadesse, S. S. ; Malandrino, F. ; Chiasserini, C.-F. Energy consumption measurements in docker. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)* 2, (2017), pp. 272–273.

[47] Vaquero, L. M. ; Rodero-Merino, L. ; Caceres, J. ; Lindner, M. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev.* 39, 1 (2008), 50–55.

[48] Xavier, M. G. ; De Oliveira, I. C. ; Rossi, F. D. ; Dos Passos, R. D. ; Matteussi, K. J. ; De Rose, C. A. A performance isolation analysis of disk-intensive workloads on container-based clouds. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, (2015), pp. 253–260.

[49] Ye, K. ; Jiang, X. ; Chen, S. ; Huang, D. ; Wang, B. Analyzing and modeling the performance in xen-based virtual cluster environment. In *2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, (2010), pp. 273–280.

[50] Zeng, H. ; Wang, B. ; Deng, W. ; Zhang, W. Measurement and evaluation for docker container networking. In *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, (2017), pp. 105–108.

[51] Zhang, Q. ; Cheng, L. ; Boutaba, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications* 1, 1 (2010), 7–18.

[52] Zhang, Q. ; Liu, L. ; Pu, C. ; Dou, Q. ; Wu, L. ; Zhou, W. A comparative study of containers and virtual machines in big data environment. *arXiv preprint arXiv:1807.01842* (2018).

Apêndice

Apêndice A

Cargas de Trabalho

A.1 Cargas de Trabalho Cenário #1

```
#!/bin/bash
# Universidade Federal Rural de Pernambuco - UAG
# Uname Research Group
# Felipe Oliveira 09/02/2019

# container name
container_name="d"
# number of containers
number_containers=20
# wait time to inicialize containers
#wait_initialize=
# wait time before kill all containers (seconds)
wait_kill=7200 # 2 hours
#time to check if kill or reboot command will be executed
wait_check_reboot=300 # 5 minutes

while [ True ]
do
    # create containers
    i=0
    while [ $i -lt $number_containers ]
    do
       # echo " docker run -d --name $container_name$i httpd"
        docker create --name $container_name$i --memory 256m -i httpd

        i=$(( $i + 1 ))
    done
```

```
    i=0
    time_up=0
    #starts containers
    while [ $i -lt $number_containers ]
    do
       # echo " docker run -d --name $container_name$i httpd"
        docker start $container_name$i
        i=$(( $i + 1 ))
    done
    # restarts containers
    while [ $time_up -lt $wait_kill ]
    do
        time_up=$(( $time_up + $wait_check_reboot ))
        # wait...
        sleep $wait_check_reboot
        i=0
        while [ $i -lt $number_containers ]
        do
            docker restart $container_name$i
            i=$(( $i + 1 ))
        done
    done
    # wait ...
    sleep $wait_check_reboot
    i=0
    # stop containers
    while [ $i -lt $number_containers ]
    do
        docker stop $container_name$i
        i=$(( $i + 1 ))
    done

    i=0
    # remove containers
    while [ $i -lt $number_containers ]
    do
        docker rm $container_name$i
        i=$(( $i + 1 ))
    done
done
```

A.2 Cargas de Trabalho Cenário #2

```
#!/bin/bash
# container name
service_name="web-server"
# number max of containers, 40 containers
number_max=41
#number min of containers
number_min=1
wait_check=30
number_step=5
n=1

#cria o servico com uma replica local
docker service create --name $service_name --replicas=1 -p 80:80 httpd

while [ True ]
do
    time_up=0

    while [ $n -lt $number_max ]
    do
        n=$(( $n + $number_step ))
                # scale up
#       echo "docker service scale $service_name=$n"
                docker service scale $service_name=$n
        sleep $wait_check
    done

    while [ $n -gt $number_min ]
    do
        n=$(( $n - $number_step))
                # scale down
#       echo "docker service scale $service_name=$n"
                docker service scale $service_name=$n
        sleep $wait_check
    done
done
```

Apêndice B

Scripts de Monitoramento

B.1 Monitor de CPU

```
#!/bin/bash
echo "usr nice sys iowait soft data hora" > monitor-cpu.txt
while [ True ]
do
cpu=`mpstat | grep all`
usr=`echo $cpu | awk '{print $3}'`
nice=`echo $cpu | awk '{print $4}'`
sys=`echo $cpu | awk '{print $5}'`
iowait=`echo $cpu | awk '{print $6}'`
soft=`echo $cpu | awk '{print $8}'`
# Obtem o tempo atual, formato RFC3339: AAAA-MM-DD HH:MM:SS
tempo=`date --rfc-3339=seconds`
# Separa data e hora do tempo obtido
data=`echo $tempo | cut -d\  -f1`
hora=`echo $tempo | cut -d\  -f2 | awk 'BEGIN{FS="-"}{print $1}'`
echo "$usr $nice $sys $iowait $soft $data $hora" >> monitor-cpu.txt

#echo "$usr $nice $sys $iowait $soft"
sleep 60
done
```

B.2 Monitor de Memória

```
#!/bin/bash
# Script para monitoramento da rede

# Escreve no arquivo o cabecalho com as informacoes
```

```
echo "metrica RX-enp63s0 TX-enp63s0 date time" > monitor-rede.txt
#Tempo de experimento 30*480 = 14.400 segundos = 240 minutos == 4 horas
e=1
while [ True ] #480 medicoes
do
# Obtem os dados da interface enp63s0
d_enp63s0=`/sbin/ifconfig "enp63s0"`
rxENS4=`echo $d_enp63s0 | grep "RX" | grep "bytes" | awk {'print $5'}`
txENS4=`echo $d_enp63s0 | grep "TX" | grep "bytes" | awk {'print $5'}`

# Obtem o tempo atual, no formato RFC3339: AAAA-MM-DD HH:MM:SS
tempo=`date --rfc-3339=seconds`
# Separa data e hora do tempo obtido
data=`echo $tempo | cut -d\  -f1`
hora=`echo $tempo | cut -d\  -f2| awk 'BEGIN{FS="-"}{print $1}'`

# Escreve no arquivo as informacoes do trafego de rede
echo $e $rxENS4 $txENS4 $data $hora >> monitor-rede.txt

e=`expr $e + 1`

sleep 60 # Tempo entre as medicoes

done
```

B.3 Monitor de Disco

```
#!/bin/bash

echo "usado data hora" > monitor-disco.txt
while [ True ]
do

disco=`df | grep /dev/sda2`
usado=`echo $disco | awk '{print $3}'`
 # Obtem o tempo atual, formato RFC3339: AAAA-MM-DD HH:MM:SS
tempo=`date --rfc-3339=seconds`
# Armazena somente os campos de interesse
num=`ps aux | awk '{if ($8~"Z"){print $0}}' | wc -l`
# Separa data e hora do tempo obtido
data=`echo $tempo | cut -d\  -f1`
```

```
hora=‘echo $tempo | cut -d\  -f2 | awk 'BEGIN{FS="-"}{print $1}'‘
echo "$usado $data $hora" >> monitor-disco.txt
sleep 60
done
```

B.4 Monitor de Processo

```
#!/bin/bash

echo "cpu mem vmrss vsz threads swap" > monitor-processo-c.txt

while [ True ]
do
pid=‘ps aux | grep [c]ontainerd | grep [u]nix | awk '{print $2}'‘
#echo $pid
dados=‘pidstat -u -h -p $pid -T ALL -r 1 1 | sed -n '4p'‘
thread=‘cat /proc/$pid/status | grep Threads | awk '{print $2}'‘
cpu=‘echo $dados | awk '{print $7}'‘
mem=‘echo $dados | awk '{print $13}'‘
vmrss=‘echo $dados | awk '{print $12}'‘
vsz=‘echo $dados | awk '{print $11}'‘
swap=‘cat /proc/$pid/status | grep Swap | awk '{print $2}'‘

echo "$cpu $mem $vmrss $vsz $thread $swap" >> monitor-processo-c.txt

sleep 60
done
```

B.5 Monitor de Processos Zumbis

```
#!/bin/bash
# Script para monitoramento de processos zumbis

# Escreve o cabecalho de identificacao dos dados

echo "metrica num_zumbis data hora" > monitor-zumbis.txt

#Tempo de experimento 30*480 = 14.400 segundos = 240 minutos == 4 horas

e=1
while [ True ] #480 medicoes
```

```
do
    # Obtem o tempo atual, no formato RFC3339: AAAA-MM-DD HH:MM:SS
    tempo=`date --rfc-3339=seconds`

    # Armazena somente os campos de interesse
    num=`ps aux | awk '{if (($8~"Z"){print $0}}' | wc -l`

    # Separa data e hora do tempo obtido
    data=`echo $tempo | cut -d\  -f1`
    hora=`echo $tempo | cut -d\  -f2| awk 'BEGIN{FS="-"}{print $1}'`

    # Mostre na tela as informacoes capturadas pelos script
   # echo $num $data $hora

    # Escreve no arquivo as informacoes do disco
    echo $e $num $data $hora >> monitor-zumbis.txt

    # Incremento contador
    e=`expr $e + 1`

    sleep 60

done
```