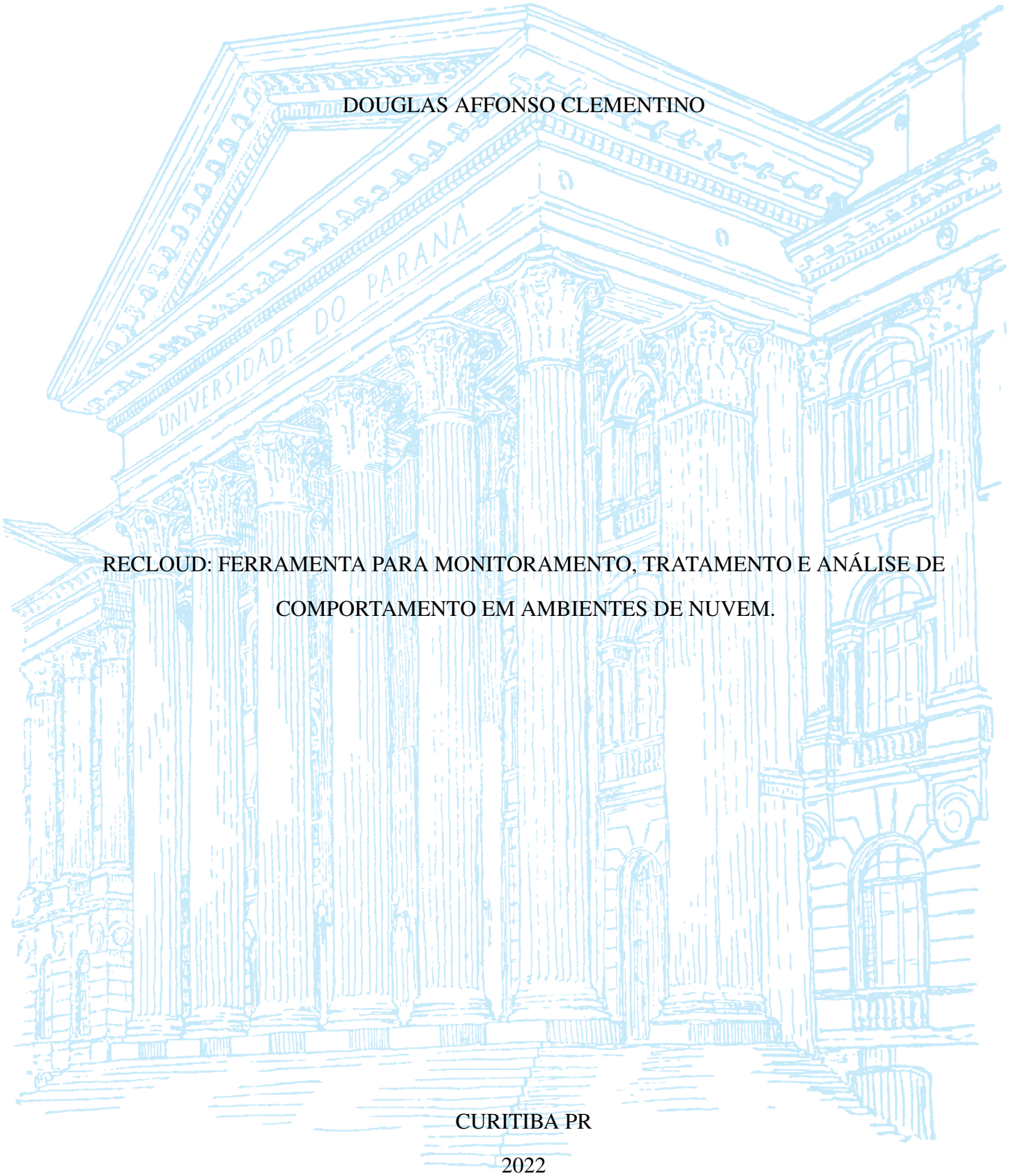
UNIVERSIDADE FEDERAL DO PARANÁ

DOUGLAS AFFONSO CLEMENTINO



RECLOUD: FERRAMENTA PARA MONITORAMENTO, TRATAMENTO E ANÁLISE DE COMPORTAMENTO EM AMBIENTES DE NUVEM.

CURITIBA PR

2022

DOUGLAS AFFONSO CLEMENTINO

RECLOUD: FERRAMENTA PARA MONITORAMENTO, TRATAMENTO E ANÁLISE DE COMPORTAMENTO EM AMBIENTES DE NUVEM.

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Marco Antonio Zanata Alves.

CURITIBA PR

2022

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Conclusão de Curso 2

Título do Trabalho: RECloud: Ferramenta para monitoramento, tratamento e análise de comportamento em ambientes de nuvem

Autor(es):

GRR 20175877 Nome: Douglas Affonso Clementino

GRR _____ Nome: _____

Apresentação: Data: 22 / 09 / 22 Hora: 14h Local: Laboratório HiPES

Orientador: Dr. Marco Antonio Zanata Alves

Membro 1: Dra. Simone Dominico

Membro 2: MSc. Vinicius Fülber Garcia

(nome) (assinatura)

AVALIAÇÃO – Produto escrito		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Conteúdo	(00-40)				
Referência Bibliográfica	(00-10)				
Formato	(00-05)				
AVALIAÇÃO – Apresentação Oral					
Domínio do Assunto	(00-15)				
Desenvolvimento do Assunto	(00-05)				
Técnica de Apresentação	(00-03)				
Uso do Tempo	(00-02)				
AVALIAÇÃO – Desenvolvimento					
Nota do Orientador	(00-20)		**********	**********	
NOTA FINAL		**********	**********	**********	100

Os pesos indicados são sugestões.

Conforme decisão do colegiado do curso de Ciência da Computação, a entrega dos documentos comprobatório de trabalho de Conclusão de Curso 2 deve deve respeitar os seguintes procedimentos: o orientador deve abrir um processo no Sistema Eletrônico de Informações (SEI – UFPR); Selecionar o tipo: *Graduação: Trabalho Conclusão de Curso*; informar os interessados: nome do aluno e o nome do orientador; anexar esta ata escaneada e a versão final do PDF da monografia do aluno; Tramitar o processo para CCOMP (Coordenação de Ciência da Computação).

AGRADECIMENTOS

Gostaria de agradecer aqueles que durante todo o processo de desenvolvimento deste projeto estiveram ao meu lado, se pondo como bussolas em momentos de dúvida, dando suporte em minhas decisões, apontando meus erros quando eu não conseguia vê-los, e me amparando perante minhas frustrações.

Um agradecimento especial a minha noiva, Alexia, por me ajudar ativamente em múltiplas etapas do desenvolvimento do trabalho e por aturar as várias discussões, muitas vezes unilaterais, sobre o projeto.

Agradeço a meus pais, Maria e Jamilson, por seus esforços e dedicação para me apresentarem oportunidades desde a minha infância, dando suporte e auxílio às minhas escolhas mesmo quando não às compreendiam por completo.

Finalmente, agradeço ao sistema público de educação, que utilizo desde minha infância e que possibilitou o meu desenvolvimento acadêmico e como ser critico. Pois sou produto da interação com professores e alunos que conheci durante as etapas da minha vida.

RESUMO

No contexto atual, onde cada vez mais é exigido o processamento de grandes quantidades de dados, de forma distribuída e sob demanda, houve a ascensão da computação em nuvem. Apesar de fundamentada em conceitos estabelecidos à décadas no meio acadêmico e empresarial, a computação em núvem está ganhando cada vez mais espaço no cotidiano do público geral. Dessa forma, tendo como alvo o modelo de serviço de *Infrastructure as a Service* (IaaS), este trabalho propõe a ferramenta *RECloud*, que tem como objetivo efetuar o monitoramento da utilização de CPU, memória e tráfego de pacotes de rede efetuado por máquinas virtuais de um sistema de nuvem, registrando esses dados em traços de monitoramento ao longo do tempo. Além disso, o sistema também é capaz de converter tais traços de monitoramento em traços de visualização (seguindo o formato Pajé, o qual também descreve todo o contexto monitorado) e traços de reprodução (simplificando o formato Pajé, descrevendo atividades realizadas por máquinas virtuais de forma individual), possibilitando a análise e manipulação de traços que descrevam ambiente de nuvem monitorado. Dessa forma, visando o ambiente de nuvem *IaaS*, busca-se criar um ferramental que estimule a análise e sobretudo o caráter reprodutivo de experimentos realizados por desenvolvedores e pesquisadores. Assim, *RECloud* realizará o monitoramento e análise de ambientes de nuvem, possibilitando, no futuro, a sua integração com ferramentas de geração de cargas sintéticas para reprodutibilidade de experimentos. Para a análise das ferramentas, serão efetuados experimentos a fim de mensurar as suas limitações de precisão, o seu impacto sobre ambiente monitorado, o desempenho de processo de conversão de traços e uma possível aplicação para traços de visualização. Ao final, serão apresentadas conclusões sobre as capacidades e limitações da ferramenta, assim como possibilidades para trabalhos futuros.

Palavras-chave: Computação em nuvem. Infrastructuere as a Service. Monitoramento.

ABSTRACT

Currently, where the processing of large amounts of data is increasingly required, in a distributed and on-demand way, there has been the rise of cloud computing. Despite being based on concepts established for decades in the academic and business world, cloud computing is gaining more and more space in the daily life of the general public. Thus, targeting the Infrastructure as a Service (IaaS) service model, this work proposes the RECloud tool, which aims to monitor the CPU, memory, and network packet traffic performed by virtual machines of a cloud system, recording this data in monitoring traces over time. Furthermore, the system is also able to convert such monitoring traces into visualization traces (considering the Pajé format, which also describes the entire monitored context) and reproduction traces (a simplified Pajé format, describing activities of virtual machines individually), enabling the analysis and manipulation of traces that describe a monitored cloud environment. Thus, focusing on the IaaS cloud environment, we seek to create a tool that stimulates analysis, especially the reproductive character of accomplished developers and researchers. In this way, RECloud will carry out the monitoring and analysis of cloud environments. In the future, it may be integrated with tools for generating synthetic loads for reproducibility of experiments. To analyze the tools, tests will be executed to assess its limitations over precision, overload over the monitored environment, performance on the trace conversion process, and some applications for trace visualization. Finally, conclusions will be drawn about the capabilities and limitations of the tool, as well as possibilities for future works.

Keywords: Cloud computing. Infrastructuere as a Service. Monitoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

CPU	*Central Processing Unit*
GMG	Gerente de Monitoramento Geral
GML	Gerente de Monitoramento Local
IaaS	Infrastructure as a Service
MPI	*Message Passing Interface*
NIST	*National Institute of Standards and Technology*
NTP	*Network Time Protocol*
PaaS	*Platform as a Service*
SaaS	*Software as a Service*
SO	Sistema Operacional

SUMÁRIO

1 INTRODUÇÃO

Nas décadas de 1960 e 1970 foi proposto e consolidado o conceito de virtualização de sistemas, visando permitir que grandes servidores pudessem prover ambientes computacionais lógicos completos a mais de um usuário simultaneamente, fazendo uso dos recursos providos por estas máquinas de forma mais eficiente (Maziero, 2020). Porém, apesar da proposta da virtualização prometer a melhor utilização dos recursos de sistemas computacionais, o barateamento do custo do *hardware* e a diminuição na dimensão física das máquinas (que deixaram de ocupar ambientes completos e passaram a limitar-se a área de uma mesa) promovidas nas décadas posteriores, fez com que o conceito de compartilhamento fosse temporariamente descartado, uma vez que era possível que cada usuário possuísse sua própria estação de trabalho física.

Entretanto, na década de 90, com o advento da linguagem *Java* (Oracle, 2022) a virtualização de *hardware* entrou em destaque, recuperando e popularizando parte das propostas originais para virtualização. Por meio da *Java Virtual Machine* tornou-se possível efetuar o desenvolvimento de aplicações multiplataforma, sendo responsabilidade das máquinas virtuais *Java* efetuarem a interface entre a aplicação desenvolvida e o ambiente em que esta estaria sendo executada.

Devido ao aprimoramento das redes de computadores e a popularização de dispositivos de computação de diversas capacidades e propósitos (*desktops*, *notebooks*, *smartphones* e dispositivos *IoT*), o ideal almejado para o surgimento da virtualização atingiu outros patamares. Isso posto, observou-se a possibilidade de utilizar o poder computacional dos servidores para atender a uma nova variedade de serviços solicitados pela sociedade, com a intenção de suprir demandas de usuários que fazem uso da tecnologia em seu dia a dia.

Sendo assim, a fim de atender a demanda de diversas esferas sociais, em 2006 foi determinado o conceito *Cloud Computing* (computação em nuvem) (H.Yang e Tate, 2012) para descrever provisão de serviços de processamento sob demanda através da internet. Neste sistema o usuário possuiria apenas a interface para comunicação e o processamento seria realizado em servidores de forma concorrente, onde múltiplas demandas de usuários seriam atendidas de forma isolada por um ou vários servidores.

Hoje, serviços providos por plataformas de nuvem já estão estabelecidos em nossa sociedade e só tendem a crescer. Isso se justifica dada a grande demanda de comunicação e processamento de dados, provido pela adição e integração de dispositivos e processos que vem migrando para a área tecnológica. Dessa forma, buscando fomentar o desenvolvimento das plataformas de nuvem, é necessário que se crie ferramental adequado a esse ambiente crescente, proporcionando a pesquisadores e desenvolvedores os meios para a evolução não somente mercadológica mas principalmente acadêmica deste novo arquétipo da computação.

1.1 PROBLEMAS

Dada a ascensão na aplicação de tecnologias em nuvem evidencia-se a necessidade de pesquisar por novas maneiras de melhorar tanto o sistema de gerenciamento de nuvem quanto a implantação de serviços sobre esta plataforma. Para desenvolver tais estudos, na maioria dos casos, é necessário o acesso a uma plataforma de nuvem pública ou privada. Esse acesso pode ser feito como administrador ou com usuário.

O primeiro cenário, no qual se utiliza de nuvens públicas, mostra-se muito restritivo, pois acessar como administrador tais plataformas envolve uma série de preocupações relacionadas a segurança da estrutura de nuvem. Além disso, acessar tais ambientes como usuário é uma operação que exige um custo financeiro, o que é agravado se os experimentos a serem executados levarem muitas horas para serem concluídos.

No segundo cenário, acessando um ambiente de nuvem privada, tanto o acesso como administrador e como o de usuário comum é facilitado, permitindo forma de simples a implementação e avaliação de novas pesquisas e ferramentas definidas sobre o ambiente de nuvem. No entanto, tal cenário privado carece da variabilidade das aplicações conforme ocorre em ambientes de nuvem pública, não proporcionando aos pesquisadores a devida avaliação de suas propostas em cenários de maior concorrência e variedade de cargas de trabalho.

Considerando esses aspectos, observa-se que um novo mecanismo é necessário para capturar atividades realizadas em cenários reais (servidores de nuvem pública) e reproduzir essas atividades em um ambiente mais controlado e acessível (servidor de nuvem privada), reforçando a pesquisa na área sob a perspectiva da reprodutibilidade, exigida tanto no desenvolvimento quanto no compartilhamento de experimentos científicos.

Assim, constata-se a necessidade de definir uma ferramenta que efetue o monitoramento da utilização de recursos de tais ambientes de nuvem, de forma que esses dados possam ser formatados em traços que representem o ambiente monitorado e possibilitem a visualização e manipulação dessas informações. Por consequência, viabilizando o posterior desenvolvimento de uma ferramenta geradora de carga de trabalho sintético, dirigida por traços que representam as cargas de trabalho reais, ajustando o uso dos recursos de CPU, memória e latência de rede conforme solicitado. Isso posto, tornaria-se possível avaliar não apenas novas técnicas de aplicação sobre ambientes de nuvem, mas também fornecer uma maneira de replicar e comparar resultados de diferentes autores.

1.2 OBJETIVOS

Esse trabalho tem como objetivo geral fornecer um método para reunir e sincronizar várias métricas sobre a utilização de recursos em um sistema nuvem do modelo *Infrastructure as a Service* (*IaaS*) ao longo do tempo. Assim, possibilitando visualização e manipulação sobre os dados extraídos de ambiente monitorado.

Buscando atender a esse objetivo, é proposta a ferramenta *RECloud*, que visa 2 frentes principais:

- Um sistema distribuído que permita a coleta do uso de CPU e memória dos ambientes virtualizados através dos *hypervisor* utilizado em cada uma das máquinas físicas monitoradas, além do monitoramento do tráfego de rede sobre as interfaces de rede de máquinas físicas pela perspectiva de pacotes, armazenando informações capturadas em formas de traço de monitoramento, que deverão descrever a utilização dos recursos ao longo do tempo;

- Um sistema conversão de traços, que permita analisar traços de monitoramento e transforma-los em traços de visualização (formato Pajé) e traços de reprodução (simplificação de formato Pajé), possibilitando análise e manipulação destes traços que descrevem ambientes monitorados.

Como trabalho futuro, propõe-se o desenvolvimento de sistema distribuído que seja capaz de, guiado por traços de reprodução previamente gerados, redefinir ambiente de nuvem

descrito por tais traços. Dessa forma, possibilitando ao usuário replicar ambiente virtual em máquinas físicas indicadas e efetuando a aplicação de tais cargas descritas ao decorrer do tempo.

1.3 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 deste trabalho será apresentada a fundamentação teórica para a construção das ferramentas propostas e do ambiente em que estas estarão situadas (modelos de virtualização de sistemas e seus componentes, sistemas de nuvem).

O Capítulo 3 tratará do estado da arte, abordando algumas ferramentas e trabalhos que se propõem a realizar integralmente ou parcialmente funções relacionadas ao monitoramento e a aplicação de cargas sobre sistemas de nuvem.

O Capítulo 4 será dedicado a descrever o ambiente, o método e as ferramentas utilizadas para efetuar o monitoramento e a geração de traços propostos por *RECloud*.

No Capítulo 5 serão apresentados métodos para avaliação das ferramentas propostas e os resultados gerados pelo processo.

E finalmente, no Capítulo 6 serão expostas as conclusões finais sobre o desenvolvimento do trabalho, considerações sobre as abordagens escolhidas e sugestões para trabalhos futuros sobre o tema.

2 FUNDAMENTAÇÃO TEÓRICA

De acordo com *National Institute of Standards and Technology* (*NIST*) (Mell e Grance, 2011), plataformas que ofereçam serviços de nuvem são caracterizados pela oferta de serviços sob demanda, por meio da Internet de forma onipresente a partir do gerenciamento de um agrupado de recursos físicos. Além disso, a disponibilização de recursos deve ser feita de forma elástica e mensurada. Uma nuvem pode ser caracterizada pelo modelo de implantação adotado, assim como qual o modelo de serviço disponibilizado (nível de abstração e controle dos usuários sobre os serviços oferecidos) (Mell e Grance, 2011) .

Dessa forma, o modelo implantação das nuvens é dividido em 4 categorias principais: i) nuvens privadas, onde a acessibilidade aos seus serviços é provida apenas a um seleto grupo de pessoas ou empreendimento, sendo utilizadas geralmente em empresas e universidades para realização de atividades internas; ii) as nuvens comunitárias, que são compartilhadas por um restrito conjunto de pessoas ou grupos com um mesmo propósito; iii) as nuvens públicas, que comumente são geridas por empresas que provêm serviços de processamento e hospedagem ao público geral, efetuando cobrança das atividades de formas variadas (como período de disponibilidade de serviço, utilização de recursos, entre outros); iv) e por último, existem as nuvens mistas que são compostas por duas ou mais categorias distintas, compartilhando ambiente, configuração ou dados entre elas.

No que se refere aos modelos de serviço oferecidos pelas nuvens, apesar de serem encontradas várias especificidades, eles se encaixam em 3 categorias principais, *Infrastructure as a Service* (*IaaS*), *Platform as a Service* (*PaaS*) e *Software as a service* (*SaaS*). A primeira, *IaaS* proporcionar aos usuários um ambiente de computação completo, entregando a ele acesso a um Sistema Operacional (SO) onde este terá acesso privilegiado, podendo efetuar operações sobre esta máquina de forma livre. Em segundo, *PaaS*, onde se disponibiliza ambiente para implementação de aplicações por parte do usuário de acordo com as ferramentas definidas em configuração de serviço de nuvem (como hospedagem de sites, hospedagem de microsserviços, entre outros), dessa forma usuário tem controle sobre as funcionalidades disponibilizadas e os dados gerados. Por último, temos *SaaS*, onde são oferecidos aos usuários acesso a aplicações executadas em ambientes de nuvem, dando liberdade sobre a manipulação da ferramenta assim como os dados sobre elas operados (serviços de edição de texto ou planilhas acessados via navegador, por exemplo).

Assim, a fim de prover esses serviços e efetuar o controle e disponibilidade de recursos computacionais de forma eficiente, utiliza-se do paradigma de virtualização. A virtualização é aplicada sobre servidores utilizados na hospedagem das nuvens, a fim de efetuar o particionamento lógico dos recursos físicos presentes nessas máquinas, possibilitando maior flexibilidade e segurança no gerenciamento das nuvens.

Dessa forma, para a oferta de serviços de *IaaS*, utiliza-se de Máquinas Virtuais de Sistema. Esse tipo de virtualização foi definida pela proposta de Popek e Goldberg (1974), onde a partir da utilização de um *hypervisor*, permite-se a definição de um ambiente virtual convidado completo sobre outro ambiente hospedeiro. Assim, torna-se possível o compartilhamento e o gerenciamento recursos de sistema convidado através do sistema hospedeiro.

O componente *hypervisor*, que é responsável por possibilitar a integração e o controle de múltiplas máquinas virtuais, pode ser de 2 tipos, *bare-metal* ou *hosted*. O tipo I, *bare-metal*, executa diretamente sobre o *hardware*, efetuando a virtualização de recursos de forma direta. Já o tipo II, *hosted* é executado como um processo que opera sobre um SO que por sua vez tem acesso

direto ao *hardware*, dessa forma o processo de virtualização tem não somente o *hypervisor* mas também o SO hospedeiro como camadas de abstração sobre o *hardware*.

Finalmente, para que o processo de virtualização de sistemas seja implementado, é necessário não apenas suporte lógico mas também suporte de *hardware* (como *Intel VT* e *AMD-V*) que possibilite ao *hypervisor* e SO hospedeiro efetuarem o processo de virtualização. Além disso, dada a importância atual desses serviços, aprimoramentos como *Intel Extended Page Tables* (Bhatia, 2009) ou *AMD-V Nested Page Table* (Virtualization, 2008) foram promovidos para substituir em *hardware* as operações previamente realizadas em *software* por *hypervisors*, proporcionando maior velocidade a ambientes virtuais.

Portanto, dada as variações no ferramental de operacionalização de nuvens *IaaS*, este trabalho se dedicará a construir a sua ferramenta de monitoramento sobre sistema de nuvem que faça uso de *hypervisor* tipo *hosted* que, por possuir um SO base, tonará viável a instanciação de processos implementados pela nossa proposta.

3 ESTADO DA ARTE

Neste capítulo, discutiremos os principais trabalhos correlatos a proposta *RECloud*, avaliando e comparando as propostas e as funcionalidades oferecidas por cada um deles.

Durante o processo de revisão bibliográfica, foram encontrados diversos trabalhos que se relacionavam à pontos específicos ao encontrados em nossa proposta, como monitoramento, conversão de arquivos de monitoramento em traços ou reprodução. A única exceção encontrada foi a proposta de Tarasov et al. (2012), que visa a aplicar cargas sintéticas guiadas por traços que descrevem atividades reais, porém este aborda somente a métrica I/O para dispositivos de bloco, métrica essa que não é visada neste trabalho.

Desta forma, os demais trabalhos correlatos foram classificados em 3 grupos:

- Os que se propõem a efetuar o monitoramento e coleta dos dados de ambiente de nuvem;
- Aqueles que propões a aplicação de cargas sintéticas sobre ambientes de nuvem;
- E os que colaboraram para a definição da arquitetura dos sistemas propostos neste projeto.

Quanto ao monitoramento, temos Hillbrecht e Bona (2012) e Xu e Yang (2011), que propõem processos de monitoramento e gestão de máquinas virtuais em tempo real de um sistema de nuvem *IaaS* de forma remota. Em ambos os trabalhos faz-se uso do *Simple Network Management Protocol* (SNMP) e das ferramentas de coleta de informações e de controle dos *hypervisors* através da biblioteca *LibVirt* (LibVirt, 2022). Outro trabalho que compartilha deste propósito é o protótipo definido por (Corradi et al., 2012), que visa um sistema *Publisher-Subscriber* com troca de dados no formato *Data Distribution Service* (DDS), efetuando o monitoramento mútuo de máquinas físicas a fim de melhorar a distribuição na criação e na migração de máquinas virtuais em um ambiente de nuvem *IaaS*. Além disso, estes são os únicos trabalhos listados que se preocupam com a manipulação de dados extraídos em rede, entretanto, diferentemente de *RECloud*, estes trabalham apenas com a mensuração do tráfego de dados, não trabalhando com a orientação dos pacotes.

Por outro lado, Righi et al. (2019) propõe a ferramenta *middleware* ProElastic, que efetua o gerenciamento proativo do número de núcleos de processamento reservados por um usuário de um sistema de nuvem *IaaS* público. Nele, é realizada a pré alocação e desalocação deste recurso considerando o consumo do usuário, desempenho esperado e custo financeiro. Em sua proposta, a ferramenta é submetida a cargas flexíveis geradas a partir do processo de integrações numéricas no modelo Newton-Cotes, de forma a simular diferentes cargas de trabalho de processamento em CPU sobre o sistema.

Alguns outros trabalhos contribuem conceitualmente como (Kao e Iyer, 1992), onde foca-se na geração de cargas de trabalho I/O e que propõe o conceito de cargas *job-unspecific* para métricas de consumo de recursos de um sistema forma geral (desconsiderando o processo que as gerou).

Finalmente, trabalhos como o de Hamza et al. (2014) e os já citados Tarasov et al. (2012) e Kao e Iyer (1992) que propõem soluções para monitoramento e criação de cargas sintéticas relacionadas a I/O e que poderão ser utilizados em trabalhos futuros que abordem essa métrica.

Tabelas 3.1 e 3.2 resumem abordagens e objetivos dos trabalho analisados. A primeira destacando as métricas abordadas e a segunda as funcionalidades propostas sobre estas métricas em cada um dos trabalhos. Coloca-se em contraste trabalhos correlatos e o trabalho proposto.

Tabela 3.1: Correlação métricas abordadas em propostas de trabalho. Em cabeçalho, cada número corresponde respectivamente à: 1-Hillbrecht e Bona (2012); 2-Xu e Yang (2011); 3-Corradi et al. (2012); 4-Righi et al. (2019); 5-Kao e Iyer (1992); 6-Hamza et al. (2014); 7-Tarasov et al. (2012).

	1	**2**	**3**	**4**	**5**	**6**	**7**	**RECloud**
Relacionado a CPU	•	•	•	•				•
Relacionado a memória	•	•	•	•				•
Relacionado a rede	•	•						•
Relacionado a I/O					•	•	•	

Tabela 3.2: Correlação de funcionalidades propostas. Em cabeçalho, cada número corresponde respectivamente à: 1-Hillbrecht e Bona (2012); 2-Xu e Yang (2011); 3-Corradi et al. (2012); 4-Righi et al. (2019); 5- Kao e Iyer (1992); 6-Hamza et al. (2014); 7-Tarasov et al. (2012).

	1	**2**	**3**	**4**	**5**	**6**	**7**	**RECloud**
Monitoramento em Traços						•		•
Conversão de Traços					•	•	•	•
Coleta de Informações de Ambiente	•	•	•					•
Monitoramento de Sistema	•	•	•	•				•
Sistema Distribuído	•	•	•	•	•			•
Geração de Carga de Trabalho Sintética					•		•	
Visualização de Ambiente Monitorado								•

4 PROPOSTA

Neste capítulo, serão abordados detalhes sobre o desenvolvimento, funcionamento, formatos de arquivo e tecnologias utilizadas para execução de todo o processo proposto pela ferramenta *RECloud*. Será discutido desde o sistema distribuído dedicado a efetuar o monitoramento do ambiente de nuvem até a conversão de arquivos de traço monitoramento em arquivos de traço de visualização e reprodução.

4.1 MONITORAMENTO

Nesta seção serão discutidas as configurações do sistema das máquinas físicas que compõem o ambiente de nuvem *IaaS* alvo. Além disso, também serão analisados todos os mecanismos criados e utilizados a fim de realizar o processo de monitoramento desse sistema, incluindo os programas gestores, a topologia do sistema de monitoramento, as ferramentas desenvolvidas e utilizadas e metodologia para realização deste processo. Também serão explorados os arquivos resultantes do processo de monitoramento, como o que descreve a estrutura do ambiente monitorado (no formato *json*), e os que tratam das métricas monitoradas, sendo eles os arquivos de traço de monitoramento de rede (binário em formato *pcap-savefile* (Tcpdump Group, 2022a)) e traço de monitoramento de tempo de processamento em CPU e utilização da memória (formato texto) gerados durante o processo de monitoramento. Ademais, será tratado do processo de armazenamento, coleta e agregação dos arquivos gerados pelo processo de monitoramento.

Na Figura 4.1 é possível observar os principais componentes tanto do ambiente de nuvem (como as máquinas físicas, o *hypervisor* e suas máquinas virtuais gerenciadas) quanto os principais componentes do sistema de monitoramento como o Gerente de Monitoramento Geral (GMG, discutido na Seção 4.1.2), o Gerente de Monitoramento Local (GML, discutido na Seção 4.1.3), as aplicações de sonda de CPU e memória (Seção 4.1.4) e a sonda de Rede (Seção 4.1.5) assim como os traços de monitoramento gerados pelo processo.

4.1.1 Configurações do Ambiente Monitorado

Considerando as máquinas física e as máquinas virtuais sobre elas criadas em um ambiente de nuvem *IaaS*, abordaremos todos os mecanismos definidos para viabilizar não somente a virtualização de sistemas completos sobre as máquinas físicas, mas também possibilitar o processo e monitoramento de suas máquinas virtuais.

Primeiramente, tratando-se da virtualização, dentre as opções oferecidas e previamente discutidas no Capítulo 2, utiliza-se da combinação de *hypervisors* tipo II KVM/QEMU (*Kernel-based Virtual Machine* e *Quick EMUlator*). Dessa forma, o KVM, um módulo de *kernel* para SOs Linux, permite ao SO hospedeiro, instanciado sobre máquina física, funcionar como um *hypervisor*, definindo e gerenciando os núcleos virtuais (*vCPU*). Enquanto o segundo *hypervisor*, QEMU, é utilizado para efetuar a emulação do processador e demais dispositivos das utilizados pelas máquinas virtuais. Esta combinação foi escolhida não somente por ser nativa dos SOs Linux (dado que cada máquina física é gerida pelo SO Ubuntu 16.04 Server) mas também pela sua simplicidade e popularidade.

Outro ponto é a utilização da ferramenta *chrony* (Curnow e Lichvar, 2021), que opera sobre o protocolo de rede *Network Time Protocol* (NTP). Assim, cada máquina física opera como um cliente NTP, efetuando cada uma delas solicitações a um mesmo servidor NTP local (máquina

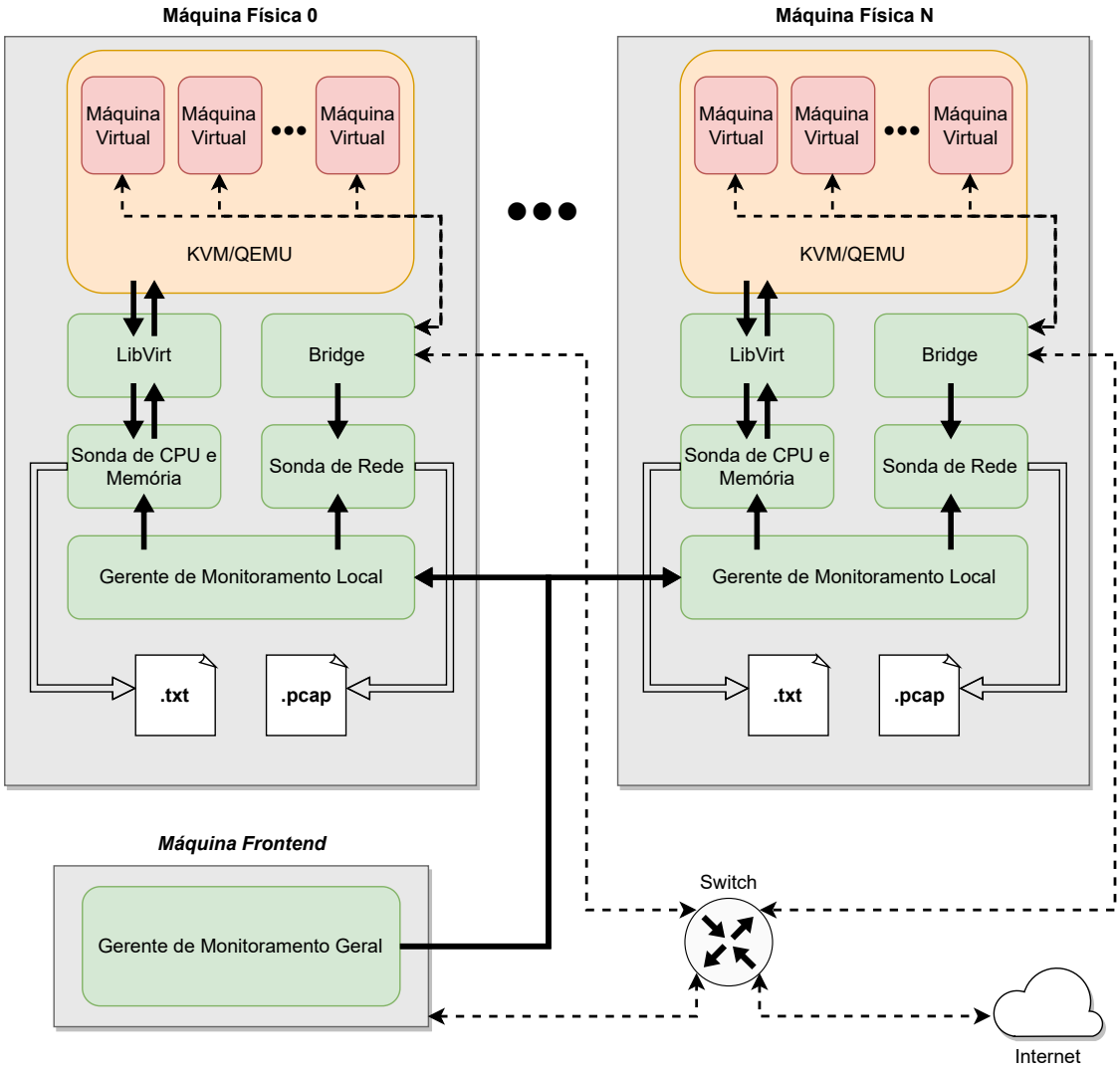


Figura 4.1: Diagrama de arquitetura do sistema distribuído de monitoramento. Na imagem, setas preenchidas indicam troca de mensagens entre componentes de sistema de monitoramento, as setas pontilhadas indicam tráfego de pacotes e as setas ocas representam escrita em arquivo.

frontend), que contém o horário de referência, mantendo a sincronia entre os relógios de cada um dos sistemas clientes e o servidor. Dessa forma, como indicado na Figura 4.2, foi definido um sistema hierárquico para manutenção na atualização dos horários das máquinas físicas (Indicadas como Cliente NTP em diagrama) que efetuam solicitações a máquina *frontend* (Indicadas como Servidor NTP em diagrama) em busca de um horário correto e, esta máquina *frontend*, por sua vez, também opera como cliente, solicitando o horário correto de um servidor NTP externo. Assim, mesmo que o servidor NTP em internet não seja alcançado a fim de corrigir o horário dos sistemas com o horário mundial, todas as máquinas físicas estarão sincronizadas pelo horário estabelecido por servidor NTP local. Essa funcionalidade permite que o usuário possa, por conta própria, ler os arquivos gerados pelo processo de monitoramento e traçar correlações entre os dados extraídos das diferentes fontes monitoradas (máquinas físicas diferentes) de forma mais precisa possível. Assim, se garante a sincronicidade dos traços de monitoramento extraídos de deferentes fontes.

Outra configuração operada sobre os ambientes de máquinas físicas foi a configuração do dispositivo de rede físico em modo *bridge*, de forma que este pudesse ser utilizado como ponte

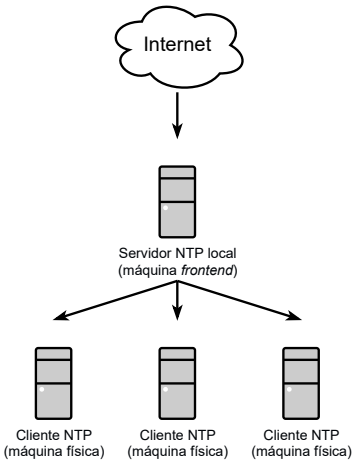


Figura 4.2: Diagrama de arquitetura NTP implementada.

pelos dispositivos de rede virtuais definidos para cada uma das máquinas virtuais, substituindo a configuração padrão que utiliza NAT (*Network Address Translation*). A partir definição da *bridge*, cada máquina virtual receberá um endereço IP válido para servidor DHCP local, tornando os pacotes que tenham as máquinas virtuais como origem ou destino rastreáveis para uma mesma rede local.

Outro ponto é a possibilidade da inicialização de comunicações externas a cada uma das máquinas virtuais, posto que dispositivos sob uma rede NAT, por padrão, só podem estabelecer conexões com outros dispositivos em mesma rede NAT ou com dispositivos externos desde que a conexão seja inicializada pelo dispositivo sob a rede NAT. Ou seja, utilizando da rede NAT, máquinas virtuais pertencentes a diferentes máquinas físicas não poderiam estabelecer conexões entre si, pois pertencem a redes NAT diferentes, problema não existente a partir da definição das *bridges*.

4.1.2 Gerente de Monitoramento Geral

Nesta seção, será apresentado o Gerente de Monitoramento Geral (GMG), cujo objetivo é orquestrar o processo de monitoramento numa perspectiva macro. Ele será responsável por efetuar conexão com as máquinas físicas indicadas para o monitoramento, gerenciando os processos e arquivos gerados durante o procedimento. O programa que representa o GMG é indicado no repositório como *startAllMonitoring.py* e foi desenvolvido na linguagem *python3.9*.

Inicialmente, o processo pode receber como entrada até 4 parâmetros, sendo 2 deles obrigatórios e 2 opcionais. Quanto aos obrigatórios, primeiramente temos *host_file*, onde o usuário deverá indicar o caminho para arquivo contendo a lista com máquinas físicas que serão monitoradas, onde cada linha deverá indicar o nome ou endereço IP da máquina alvo. O segundo parâmetro é *output_folder*, onde deverá ser indicado pelo usuário o caminho para diretório onde todos os arquivos gerados pelo processo de monitoramento serão organizados e armazenados.

Quanto aos opcionais, temos o parâmetro indicado pela *flag -d*, com a qual o usuário poderá indicar um valor de ponto flutuante, que representará o intervalo de tempo (*delay*) definido entre cada amostragem efetuada pela sonda de CPU e memória. Caso não seja indicado explicitamente algum valor, é definido por padrão o intervalo de 0,5 segundos. Para sonda de rede, a identificação e registro dos pacotes é feito assim que estes são identificados em interface de rede, processo discutido em detalhes na Seção 4.1.5.

O outro parâmetro opcional é representado pela *flag -t*, com a qual o usuário poderá indicar a quantidade de tempo desejada, em segundos, para o período de monitoramento.

Após o tratamento e validação dos argumentos de entrada, será solicitado ao usuário as credenciais SSH (*Secure Shell*) utilizadas nas máquinas físicas. Dessa forma, possibilitando efetuar o gerenciamento dos processos de monitoramento em cada um dos dispositivos. Esse usuário deverá deter as mesmas configurações em todas as máquinas físicas, além de possuir permissões administrativas. A utilização dos privilégios de administrador restringem-se apenas para a coleta de metadados dos ambientes monitorados, resguardando informações de usuários de máquinas virtuais.

Uma vez que as credenciais sejam informadas, será efetuada a leitura do arquivo *host_file*, recuperando o endereço de todas as máquinas que se deseja monitorar. Posteriormente, efetua-se a tentativa de conexão com máquinas físicas a serem monitoradas através do protocolo SSH utilizando das credenciais previamente informadas.

Com a conexão estabelecida entre máquina *frontend* e todas as máquinas físicas a serem monitoradas, será definida uma instância do programa *environment.py* em cada um dos terminais monitorados. Este processo deverá efetuar a coleta de metadados referentes a cada uma das máquinas físicas, assim como cada uma de suas máquinas virtuais em execução, contendo identificadores e valores referentes aos recursos reais e virtuais no formato *JSON*. A Figura 4.3 exemplifica um arquivo *environment.json* gerado para uma máquina física *MF1* onde seu *hostname* e IP estão evidenciados. Além disso, o vetor *virtual_machines* possui uma entrada para o objeto relacionada a máquina virtual *MV1*, onde são informados o seu endereço IP assim como suas informações de gerenciamento (id, UUID, mac, vnic) e seus recursos (quantidade de memória física dedicada e de núcleos de processamento).

```
{
   "hostname": "MF1",
   "ip": "10.254.227.240",
   "virtual_machines": [
      {
         "name": "MV1",
         "id": 1,
         "UUID": "445a0dbc-abc5-4f8e-8b97-418672938d78",
         "mac": "52:54:00:90:00:b8",
         "vnic": "vnet0",
         "ip": "10.254.227.210",
         "max_memory": 1003520,
         "vcpus": 1
      }
   ]
}
```

Figura 4.3: Exemplo de arquivo *environment.json* para máquina física *MF1*.

Importante atentar que foi necessária a utilização da ferramenta de exploração de redes *Nmap* (Lyon, 2022), isso se deve ao endereçamento IP das máquinas virtuais estar incumbido ao servidor DHCP local, e não mais ao *hypervisor* das máquinas físicas (devido ao processo de definição das *bridges* previamente discutido). Dessa forma, escaneia-se a rede local em busca dos endereços das máquinas que a compõem, possibilitando o processo de correlacionar o endereço MAC gerado para as placas de rede virtuais de cada máquina virtual (gerenciado por *hypervisor* de cada máquina física) aos endereços encontrados em rede local. Quanto as demais informações, é possível extraí-las do *hypervisor* através da biblioteca *libvirt*.

Após a geração dos arquivos *environment.json*, envia-se comando a interface *chronyc* de cada uma das máquina físicas. Isso indicará ao *daemon chronyd* que o horário local deve

ser atualizado imediatamente e não gradualmente como ele opera por padrão (utilizando-se do protocolo NTP como descrito na Seção 4.1.1).

Dessa forma, após a solicitação de sincronia a todas as máquinas físicas, o GMG irá invocar em cada uma delas o Gerente de Monitoramento Local (que será discutido na Seção 4.1.3), enviando a ele também o intervalo de tempo indicado pelo usuário para a amostragem da sonda de CPU e memória. Uma vez que este processo seja invocado, caberá ao GML gerenciar as sondas que, de fato, efetuaram a coleta dos dados. Assim, uma vez definidas todas as instâncias do GML, o processo GMG enviará periodicamente solicitações ao SO das máquinas físicas (em intervalos de 1 segundo), verificando se cada um dos monitores locais continua em funcionamento. O GMG permanece neste estado até que usuário indique o fim do processo de monitoramento (enviando sinal de finalização SIGINT ou SIGTERM) ou, no caso da utilização de parâmetro *-t*, até que o intervalo para o processo de monitoramento indicado chegue ao fim.

Uma vez que seja informado ao GMG que o processo de monitoramento deve ser finalizado, este deverá enviar sinal *SIGTERM* a todos os processos GML, indicando que estes devem ser finalizados. Verifica-se se todos os processos GML foram finalizados (armazenando propriamente todos os dados de monitoramento). Após isso, inicializa-se o processo de agregação dos arquivos gerados em cada uma das máquinas físicas, incluso o já abordado *environment.json* assim como *cpu_mem_output.txt* e *network_output.pcap*, que serão discutidos em Seções 4.1.4 e 4.1.5, respectivamente.

Além disso, uma vez que todos os arquivos sejam recuperados, é gerado o arquivo *environments.json*, que será um compilado de todos os arquivos *environment.json*. Nele, será agregado em vetor *hosts* as informações de cada uma das máquinas físicas monitoradas. Na Figura 4.6 é possível observar um exemplo de arquivo *environments.json*, descrevendo todo o ambiente monitorado, contendo a máquinas físicas *MF1* e *MF2* e suas máquinas virtuais *MV1* e *MV2*, respectivamente.

Ao final, será definido um diretório experiment_[DATA/HORA], onde DATA/HORA são a data e o horário de início de monitoramento, sob o diretório indicado em *output_folder*. Nele serão organizados todos os arquivos produtos do processo de monitoramento de todas as máquinas físicas. Os arquivos serão armazenados de forma análoga ao exemplificado na Figura 4.4, indicando diretório relativo a experimento (*experiment_10-05-2022_16:37:27*) contendo diretórios destinado as máquinas físicas monitoradas (*MF1* e *MF2*) cada qual armazenando arquivos resultantes do processo de monitoramento de seu ambiente. Além disso, também revela-se o arquivo *environments.json*, contendo compilado de informações de ambientes de máquina físicas monitoradas.

Ainda, na Figura 4.5 define-se um diagrama de sequência para as trocas de mensagens e processos descritos para funcionamento do Gerente de Monitoramento Geral. Este diagrama contém, referência a diagrama com mesmo propósito destinado a Gerente de Monitoramento Local, definido na Figura 4.7.

4.1.3 Gerente de Monitoramento Local

Nesta seção será abordado o Gerente de Monitoramento Local, componente do sistema distribuído incumbido de gerir o processo de monitoramento em cada máquina físicas. Ele efetuará o controle e a inicialização das sondas de coleta da utilização dos recursos de CPU e memória (explorado na Seção 4.1.4) das máquinas virtuais, assim como da sonda de rede (explorado na Seção 4.1.5), que efetuará a captura dos pacotes trafegados pela interface de rede de máquina física em modo *bridge*. O programa que representa o GMG é indicado no repositório como *startMonitoring.py* e foi desenvolvido na linguagem *python3.9*.

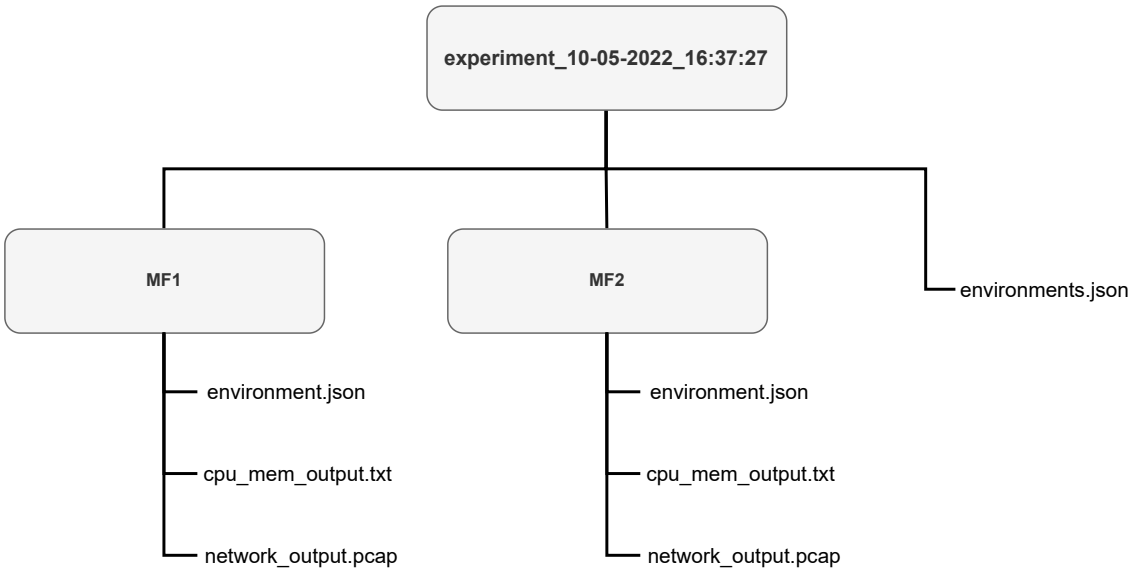


Figura 4.4: Diagrama exemplo para organização de arquivos resultantes de processo de monitoramento.

Ao inicializar o Gerente de Monitoramento Local, é possível utilizar-se de 2 parâmetros opcionais. O primeiro, indicado pela *flag -d*, possibilita ao usuário estabelecendo o valor desejado para o intervalo entre as coletas efetuada pela sonda de CPU e memória, indicando um valor em ponto flutuante na escala de segundos. Caso não seja definido um valor, o intervalo entre amostragens será de 0,5 segundos.

Quanto ao segundo parâmetro opcional, indicado pela *flag -e*, é utilizado para possibilitar ao usuário indicar o caminho para o arquivo *environmt.json*, de forma que os endereços IP de máquinas virtuais listados nele sejam utilizados como filtro para captura de pacotes pela sonda de rede. No caso da ausência de *-e*, sonda de rede deverá coletar todos os pacotes que trafeguem através da interface de rede monitorada.

Após o tratamento dos argumentos de entrada, verifica-se se o processo está rodando com privilégios de administrador, necessário para o pleno funcionamento da sonda de rede *tcpdump* (Tcpdump Group, 2022b). Em seguida, valida-se a existência do diretório de saída, preparando-o para uma nova rodada de monitoramento.

Ao final dos preparativos para o processo de monitoramento local, serão invocadas as sondas para a coleta de dados utilizando o módulo *Popen*, nativo em *python3.9*, sendo indicadas as *flags* apropriadas (exploradas cada qual em sua respectiva seção) assim como a localização de saída de arquivos gerados. Além disso, cada uma das sondas é invocada contendo prioridade máxima (*niceness* = -20 em sistemas Linux). Dessa forma, sempre que necessário e possível, eles serão escalonados ao processador e executados, enfatizanto a precisão do processo de amostragem, o que pode acarretar em impacto a demais aplicações em execução no ambiente de máquina física.

Uma vez que as sondas sejam inicializadas, processo GML entrará em laço, verificando em intervalos de 1 segundo se os processos de sonda estão em funcionamento, aguardando sinal indicando fim de processo de monitoramento.

Ao receber um sinal SIGINT ou SIGTERM, o GML efetuará os preparativos de finalização de monitoramento. Dessa forma, será enviado um sinal SIGTERM a cada um dos processos sonda, indicando a eles que o processo de monitoramento deve ser finalizado. Uma vez que todos os processos de sonda sejam finalizados, o GML finalizará sua execução.

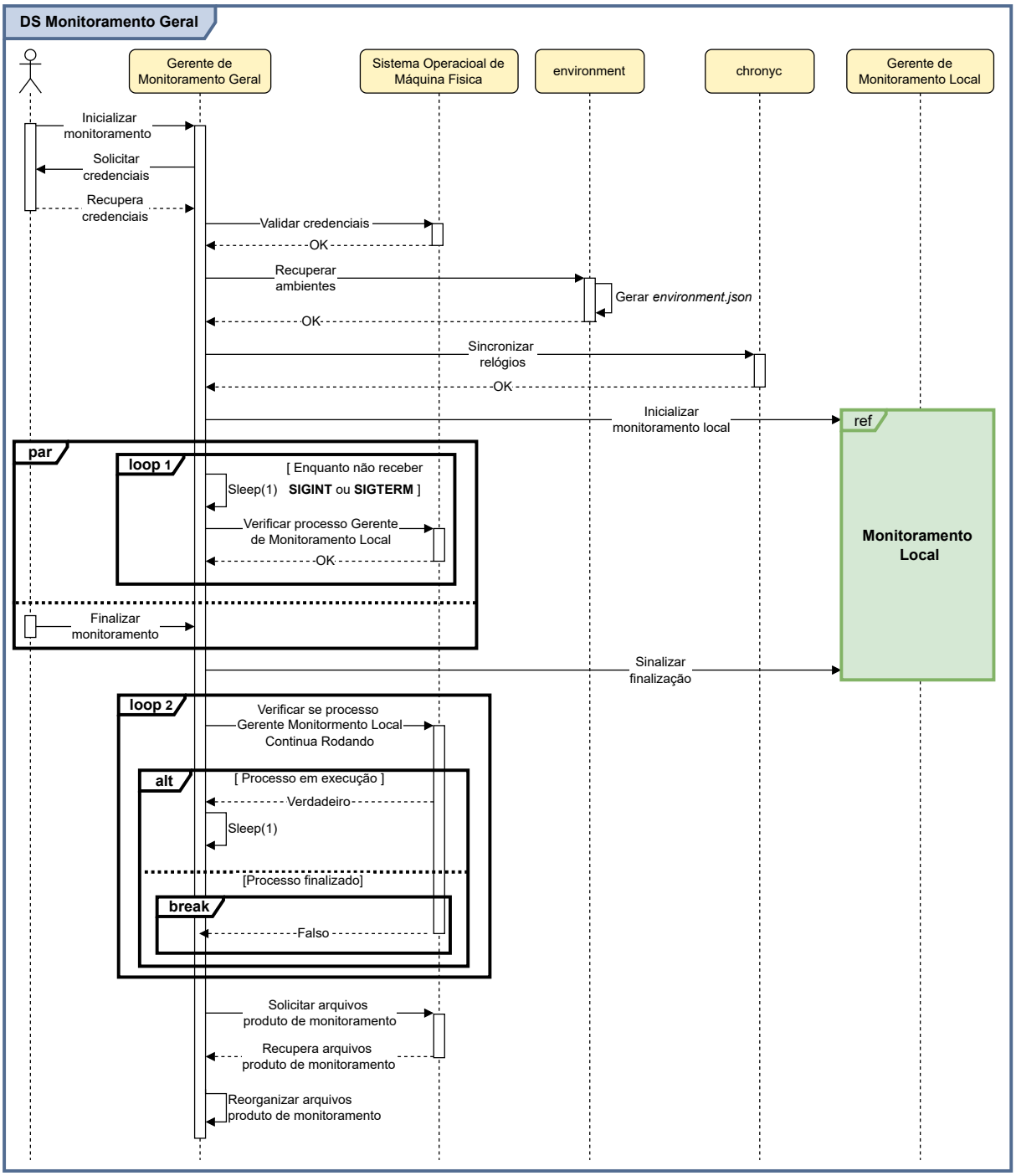


Figura 4.5: Diagrama de sequência para fluxo de eventos e comunicações de processo Gerente de Monitoramento Geral.

Na Figura 4.7, é possível observar o diagrama de sequência descrevendo fragmento para o processo de funcionamento de Gerente de Monitoramento Local. Comunicações *Inicializar monitoramento local* e *Sinalizar finalização* provêm do processo descrito para Gerente de Monitoramento Geral (Seção 4.1.2) e é exemplificado em diagrama de sequência contido na Figura 4.5.

```
{
   "hosts": [
      {
         "hostname": "MF1",
         "ip": "10.254.227.1",
         "virtual_machines": [
            {
               "name": "MV1",
               "id": 1,
               "UUID": "445a0dbc-abc5-4f8e-8b97-418672938d78",
               "mac": "52:54:00:90:00:b8",
               "vnic": "vnet0",
               "ip": "10.254.227.2",
               "max_memory": 1003520,
               "vcpus": 1
            }
         ]
      },
      {
         "hostname": "MF2",
         "ip": "10.254.227.3",
         "virtual_machines": [
            {
               "name": "MV2",
               "id": 1,
               "UUID": "11e80994-e4be-4eb0-95f9-7dafd5ad657a",
               "mac": "52:54:00:26:fc:aa",
               "vnic": "vnet0",
               "ip": "10.254.227.4",
               "max_memory": 1003520,
               "vcpus": 1
            }
         ]
      }
   ]
}
```


Figura 4.6: Exemplo de arquivo *environments.json*, que agregará todos os arquivos *environments.json* gerados durante um processo de monitoramento.

4.1.4 Sonda de CPU e Memória

Nesta seção, será descrita a construção e funcionamento das sondas de CPU e memória, cujo objetivo é efetuar a coleta do consumo destas métricas realizada por cada uma das máquinas virtuais sendo executadas em máquina física alvo. A Sonda de CPU e memória é indicada no repositório como *cpuMemMonitor*. Ela foi desenvolvida na linguagem C em padrão C11 e compilada pelo compilador GCC em versão 5.4.0. Esta versão contém alteração para funções de horário relacionadas a estrutura *timespec*, utilizada no desenvolvimento da sonda.

No processo de compilação foram utilizadas as *flags -lvirt* e *-lm* possibilitando acesso as funcionalidades da biblioteca de gerenciamento de máquinas virtuais *libvirt* (LibVirt, 2022) e da biblioteca de funções matemáticas *libm* (Foundation, 2022a) respectivamente. Além dessas, também foi utilizada a *flag* de otimização *-O3* (Foundation, 2022c).

Para a manipulação do tempo e seus cálculos, utiliza-se da estrutura *timespec* (descreve tempo em segundos em *tv_sec* e nanosegundos em *tv_nsec*) e da função *clock_gettime* (utilizada

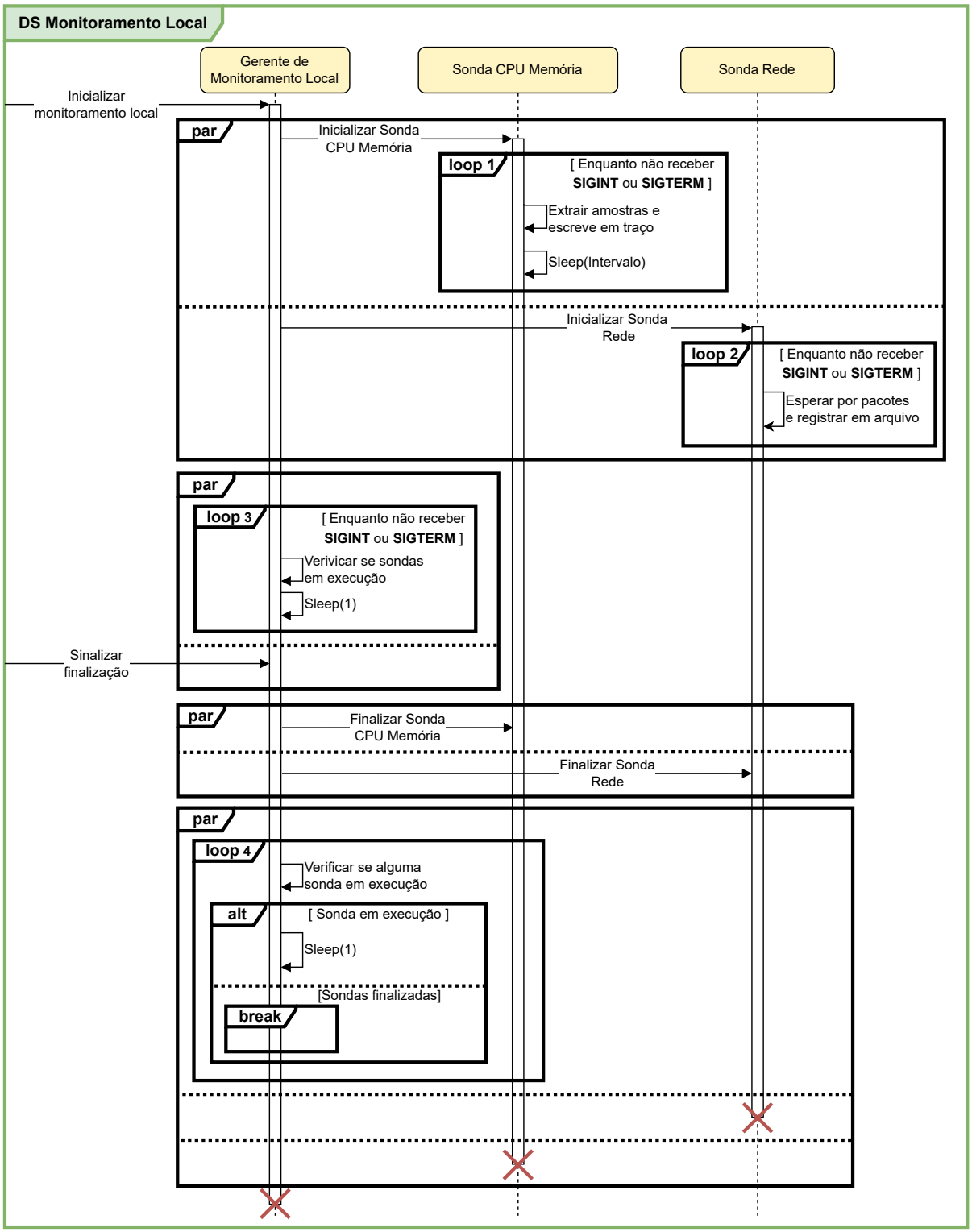


Figura 4.7: Diagrama de sequência para fluxo de eventos e comunicações de processo Gerente de Monitoramento Local.

para recuperação de tempo do sistema em estrutura *timespec*, relativo a *Epoch*) da biblioteca *time* (Foundation, 2022b).

O processo de monitoramento de CPU e memória pode receber 2 argumentos opcionais. Como primeiro, indicado pela *flag -d*, o usuário poderá indicar um valor de ponto flutuante para o intervalo de tempo, em segundos entre cada amostragem dos valores de CPU e memória. Este

valor será armazenado e manipulado através da variável *delay* e, caso não seja informado, terá como padrão o valor 0,5 segundos.

O segundo argumento opcional indica a saída de dados durante o processo de monitoramento, ele é representado pela *flag -o* pela qual o usuário poderá indicar o caminho para arquivo em formato texto onde ele deseja que sejam armazenadas as informações. No caso da ausência do parâmetro *-o*, o processo utilizará da saída padrão (*stdout*).

Após o tratamento dos argumentos de entrada, utiliza-se da função *libvirtConnect* da biblioteca *libvirt* (LibVirt, 2022) para estabelecer conexão com o *hypervisor* (enviando-se o parâmetro *"qemu:///system"*, indicando QEMU local). Em caso de sucesso, será retornado ponteiro para estrutura de dados *virConnect*, utilizada para comunicação com *hypervisor*.

Em seguida, através da função *getActiveDomains* serão recuperados ponteiros para estruturas *virDomain* para cada uma das máquinas virtuais (ou domínios) ativas em uma máquina física monitorada. Assim, possibilitando comunicação individual com cada uma delas. Além disso, habilita-se para monitoramento de memória o *balloon driver* de cada domínio, de forma que essa métrica possa ser extraída de cada máquina virtual monitorada.

Após a definição do valor de todas as variáveis, inicializa-se o laço de monitoramento, onde a cada intervalo de segundos, indicado por *delay*, serão extraídas as métricas de cada máquina virtual do ambiente. O pseudocódigo indicado no Algoritmo 1 descreve o processo de monitoramento.

Algoritmo 1 Laço de monitoramento:

```
 1: acum_err ← 0.0
 2: ts_atual ← recuperaTempoAtual()
 3: imprimeCabecalhoMonitoramento(ts_atual)
 4: while not SIGTERM do
 5:     ts_coleta ← recuperaTempoAtual()
 6:     for dominio ∈ dominios do
 7:         dominio.usoCPU ← recuperaUsoCPU(dominio)
 8:         dominio.usoMem ← recuperaUsoMem(dominio)
 9:     end for
10:     imprimeTempoColeta(ts_coleta)
11:     for dominio ∈ dominios do
12:         imprimeDadosDomionio(dominio)
13:     end for
14:     ts_invervalo ← calculaProximoDelay(ts_coleta, ts_anterior, delay, &acum_err)
15:     ts_anterior ← ts_coleta
16:     aguarda(ts_intervalo)
17: end while
18: ts_atual ← recuperaTempoAtual()
19: imprimeRodapeMonitoramento(ts_atual)
```

Primeiramente inicializa-se a variável acumuladora de erro *acum_err*. Em seguida recupera-se o tempo atual do sistema em variável ts_atual, imprimindo-o juntamente com cabeçalho em saída previamente determinada, indicando o tempo de início do monitoramento. Após isso, inicializa-se o laço de monitoramento, que será mantido até que o processo receba um sinal *SIGTERM*.

Já dentro do laço de monitoramento, primeiramente recupera-se o tempo de coleta em variável *ts_coleta* (indicado na linha 5). Posteriormente, itera-se sobre todos os domínios ativos (máquinas virtuais em execução), recuperando os valores de uso de CPU e memória.

Para estipular o uso de CPU (representado pela chamada de função *recuperaUsoCPU()* na linha 7), primeiramente recupera-se informações de *domínio* utilizando da função *virDomainGetInfo* de biblioteca *Libvirt*. Será retornado por essa função um ponteiro para estrutura *virDomainInfo* contendo o campo *cpuTime*, que é formado por número inteiro que indica a quantidade de tempo de utilização de CPU (ou seja, em um processador contendo com 2 núcleos, *cpuTime* indicará a soma do tempo de uso em ambos) desde a inicialização do *domínio*. Após isso efetua-se o cálculo:

$$uso\_CPU = \frac{(cpuTime\_atual - cpuTime\_prev)}{(t\_coleta\_atual - t\_coleta\_prev)} \tag{4.1}$$

Onde *cpuTime_atual* indica o valor de coleta atual de *cpuTime*, *cpuTime_prev* indica o valor para amostragem anterior à amostra atual de *cpuTime*, *t_coleta_atual* representa o tempo da coleta atual de *cpuTime* em nanosegundos e *t_coleta_prev* o tempo da coleta anterior à coleta atual em nanosegundos. Dessa forma, será armazenado em *uso_CPU* a soma da porcentagem do tempo de utilização de todas as CPUs do domínio entre a amostragem anterior e a atual. Ao final, armazena-se os valores de *cpuTime_atual* e *t_coleta_atual* em *cpuTime_prev* e *t_coleta_prev*, respectivamente e retorna-se o valor de *uso_CPU*.

Para os valores que caracterizam a utilização da memória (indicado pela chamada de função *recuperaUsoMem()* na linha 8), esses são obtidos a a partir da chamada de função *virDomainMemoryStats()* de biblioteca *Libvirt* para cada domínio, onde será retornado ponteiro para vetor de estrutura *virDomainMemoryStatStruct*. Neste vetor estarão contidas as entradas para várias estatísticas relacionadas ao estado atual de memória da máquina virtual, sendo que as estatísticas retornadas dependem diretamente das informações disponibilizadas pelo *hypervisor* e podem diferir de um ambiente de virtualização para outro. Dessa forma, considerando o *hypervisor QEMU*, serão extraídas as estatísticas representadas pelas macros *VIR_DOMAIN_MEMORY_STAT_UNUSED* e *VIR_DOMAIN_MEMORY_STAT_AVAILABLE* que conterão valores para memória disponível e memória total do domínio respectivamente. Desse modo são feitos os seguintes cálculos:

$$uso\_memoria = memoria\_total - memoria\_disponivel \tag{4.2}$$

$$uso\_memoria\_perc = \frac{uso\_memoria}{memoria\_total} \times 100 \tag{4.3}$$

Como indicado Equação 4.2, o valor de *uso_memoria* será calculado a partir da diferença entre a memória total disponível à máquina virtual (*memoria_total*) da memória livre (memoria_disponivel). Em outras palavras, *uso_memoria* representará o valor absoluto, em kB, da soma entre memória e memória em *buffers* de cada domínio.

Por outro lado, na Equação 4.3, calcula-se o valor de *uso_memoria_perc*, que representará a porcentagem da utilização da memória em relação ao total disponível ao domínio.

Importante atentar que, enquanto os valores de cada amostragem para CPU indicam a porcentagem de uso deste recurso entre os momentos de amostragem, as métricas relacionadas a memória indicam uso do recurso no momento da amostragem.

Após extraídas as métricas de cada uma das máquinas virtuais ativas em máquina física monitorada, registra-se o momento (*timestamp*) de extração, assim como as métricas de CPU e memória para cada domínio (indicado nas linhas 11 à 13).

Em seguida, como indicado na linha 14, chama-se a função *calculaProximoDelay()*, que retornará o tempo de espera necessário até a próxima rodada de amostragens. De forma a ater-se ao intervalo de segundos previamente estipulado em variável *delay*. Para calcular tal

valor, considera-se o intervalo de tempo entre as coletas atual a anterior (indicadas em *ts_coleta* e *ts_anterior*, respectivamente). Como indicado na Equação 4.4 , primeiramente calcula-se o erro acumulado (indicado por *acum_err*) entre intervalos de amostragem durante o processo de monitoramento sempre buscando definir intervalo de espera compatível com valor informado em *delay*. Após atualização de erro acumulado, com indicado na Equação 4.5 calcula-se o tempo em que o processo ficará em espera até próximo intervalo de amostragem, considerando o intervalo ideal entre amostragens indicado em *delay* somado ao erro acumulado durante o processo de monitoramento em *acum_err*. Finalmente, após calcular novo intervalo, o valor é retornado por *calculaProximoDelay()* e armazenado em *ts_invervalo*.

$$acum\_err = acum\_err + (delay - (ts\_coleta - ts\_anterior)) \tag{4.4}$$

$$intervalo = delay + acum\_err \tag{4.5}$$

Em seguida, armazena-se o tempo de coleta atual *ts_coleta* na variável *ts_anterior*, guardando-o para próxima rodada de amostragens e, posteriormente inicializando-se intervalo de espera indicado em *ts_intervalo*.

Uma vez que o processo receba um sinal de finalização *SIGTERM*, encerra-se o laço de monitoramento. Nesse momento será coletado o tempo atual, que indicará na linha de rodapé (função *imprimeRodapeMonitoramento()*) o momento de finalização de processo de monitoramento.

Após isso, desativa-se o modo de monitoramento de memória de *balloon driver* de todos os domínios, finalizam-se as conexões estabelecidas com o *hypervisor* e com máquinas virtuais, liberando todas as estruturas utilizadas para armazenamento dos dados.

```
1  Tue 2022-05-10 16:37:27.360726067; ********* Start Monitoring **********
2  Tue 2022-05-10 16:37:27.360789133; MV1 330772 34.11 0.00
3  Tue 2022-05-10 16:37:27.863849198; MV1 330772 34.11 30.00
4  Tue 2022-05-10 16:37:28.363651755; MV1 330772 34.11 40.00
5  Tue 2022-05-10 16:37:28.863587821; MV1 775777 80.00 100.00
6  Tue 2022-05-10 16:37:29.363641068; MV1 775777 80.00 100.00
7  Tue 2022-05-10 16:37:29.863602301; MV1 775777 80.00 100.00
8  Tue 2022-05-10 16:37:30.363559597; MV1 775777 80.00 100.00
9  Tue 2022-05-10 16:37:30.863818796; MV1 330756 34.10 0.00
10 Tue 2022-05-10 16:37:31.363569779; MV1 330756 34.10 0.00
11 Tue 2022-05-10 16:37:31.863843037; MV1 330724 34.10 0.00
12 Tue 2022-05-10 16:37:31.884126141; ********* Stopping Monitoring **********
```

Figura 4.8: Exemplo de traço de monitormento de CPU máquina virtual exemplo *MV1*.

Na Figura 4.8 exemplifica-se o traço resultante do processo de monitoramento de máquina física *MF1*, contendo uma máquina virtual *MV1*. A esquerda do caractere *';'* em todas as linhas indica-se o momento de inicialização do processo (linha 1), coleta das amostras (linhas 2 à 11) e finalização de monitoramento (linha 12). Em todas as linhas de amostragem, após *timestamp*, observa-se entrada para informações extraídas de máquina virtual, vindo primeiro o nome da máquina, seguido pela quantidade de memória em uso dado em kB, quantidade de memória em uso dado em porcentagem e percentual de CPU em uso. Neste exemplo é apresentado somente a atividade de máquina virtual (domínio) *MV1* porém, na existência de múltiplas máquinas virtuais monitoradas, cada linha de amostragem apresentaria múltiplas entradas para máquinas virtuais separadas por caractere *';'*.

4.1.5 Sonda de Rede

Nesta seção, será tratada a aplicação utilizada como sonda de rede durante o monitoramento do ambiente, registrando o tráfego de pacotes realizado em cada uma das máquinas físicas. Para realização desta tarefa, utilizou-se a aplicação *tcpdump* (Tcpdump Group, 2022b), que efetuará o monitoramento de uma interface de rede em modo *bridge*, capturando pacotes que trafeguem por tal interface e registrando em arquivo de saída binário no formato *pcap-savefile* (também documentado como *pcap* ou *libcap*).

O formato de arquivo *pcap-savefile* (Tcpdump Group, 2022a) é dividido em duas partes. Primeiramente insere-se um cabeçalho ao arquivo, como indicado em Tabela 4.1, onde *Magic number* contém uma codificação de 4 bytes indicando qual será a representação temporal dos *timestamps* (em segundos e nanosegundos ou em segundos e microsegundos), seguindo *endianness* do sistema em que foi gerado (possibiltando definir tanto *endianness* de sistema onde arquivo foi escrito como o do sistema em que arquivo está sendo lido). Em seguida temos *Major version* e *Minor version*, cada qual contendo 2 bytes representando a versão do formato *pcap-savefile* em que arquivo foi escrito (verificando compatibilidade). Após isso, apresentam-se os campos *Time zone offset* e *Time stamp accuracy*, ocupando 4 bytes cada e contendo valor 0, sendo reservados para uso futuro. Lógo após, campo *Snapshot length* indicará o tamanho máximo de bytes registrados por pacote capturado. E finalmente *Link-layer header type*, contendo codificação para o tipo de enlace do dispositivo monitorado (definido pelo protocolo utilizado em camada de enlace de interface monitorada).

Tabela 4.1: Exemplo de cabeçalho por arquivo em formato *pcap-savefile*. Extraído de Tcpdump Group (2022a)

Magic number	
Major version	Minor version
Time zone offset	
Time stamp accuracy	
Snapshot length	
Link-layer header type	

A segunda parte contém as entradas para os pacotes monitorados, cada qual contendo seu próprio cabeçalho de pacote, como é descrito na Tabela 4.2. O primeiro campo, *Time stamp, seconds value*, contém 4 bytes e indicará o momento, em segundos, da captura do pacote em relação a 1 de Janeiro de 1970 00:00:00 UTC. O segundo campo *Time stamp, microseconds or nanoseconds value*, também contendo 4 bytes, complementará o primeiro campo, indicando a porção em nanosegundos ou microssegundos (a depender de *Magic number*) a partir de *Time stamp, seconds value* em que pacote foi capturado. Em seguida *Length of captured packet data* contém 4 bytes e indicará o tamanho original do pacote capturado (*headers* + *payload*). Finalmente *Un-truncated length of the packet data*, também de tamanho 4 bytes, indicará a quantidade de bytes registrada do pacote em arquivo de saída (menor ou igual a *Snapshot length*).

Tabela 4.2: Exemplo de cabeçalho por pacote em formato *pcap-savefile*. Extraído de Tcpdump Group (2022a)

Time stamp, seconds value
Time stamp, microseconds or nanoseconds value
Length of captured packet data
Un-truncated length of the packet data

Assim, ao se inicializar a aplicação *tcpdump* os parâmetros *-i*, *-w*, *-U* e *-s* são indicados, alinhando o processo de monitoramento da ferramenta com os objetivos do projeto: extrair e registrar o tráfego de pacotes tendo como origem ou destino as máquinas virtuais monitoradas, registrando origem, destino, tamanho e tempo de envio de cada um destes pacote.

O primeiro parâmetro, *-i* é utilizado para definir a interface de rede de máquina física que será monitorada e sempre virá seguido pelo nome desta interface. Deste modo, como como previamente estabelecido na Seção 4.1.1, definira-se como interface de rede a ser monitorada a que está em modo *bridge* e é utilizada por todas as interfaces de rede virtuais, onde estará centralizado o tráfego de pacotes efetuados por todas as máquinas virtuais geridas pela máquina física.

O parâmetro *-w* deverá ser seguido por caminho válido para um arquivo, indicando que processo de monitoramento de rede deverá registrar eventos (tráfego de pacotes) em tal arquivo informado. O arquivo indicado será um binário organizado no formato *pcap-savefile*.

Quanto ao parâmetro *-U*, em combinação com *-w*, indicará que novas capturas de pacote não devem ser acumuladas em *buffer* interno de processo, sendo escritas em arquivo de saída imediatamente.

Com parâmetro *-s*, estabelece-se o valor de *snaplen*, indicando a quantidade máxima de bytes do pacote capturado (*headers* + *payload*) que devem ser registradas, truncando bytes registrados caso tamanho de pacotes exceda *snaplen*. Dado que, em cada entrada de cabeçalho por arquivo teremos o momento de captura de pacote em campos (momento de envio) *Time stamp, seconds value* e conjunto com *Time stamp, microseconds or nanoseconds value* e o tamanho em bytes do pacote em *Length of captured packet data*, basta recuperar a origem e o destino de cada pacotes trafegado. Ou seja, basta garantir que o cabeçalho IP (contendo IP de origem e destino do pacote) seja registrado a cada pacote. Dessa forma, utilizou-se o valor 96 para *snaplen*, garantindo que os primeiros 96 bytes de cada pacote sejam registrados em arquivo de saída, armazenando cabeçalho IP tanto nas versões IPv4 quanto IPv6 de todos os pacotes.

Finalmente, utilizando-se do parâmetro *host*, é possível indicar à sonda a lista de endereços IP das máquinas virtuais a serem monitoradas. Dessa forma, somente pacotes que possuem origem ou destino como sendo algum dos endereços IP listados será registrado em arquivo de saída.

Importante atentar para o fato de que a ferramenta de sonda de rede efetuará a captura de pacotes diretamente da *bridge* previamente definida. Dessa forma, pacotes cuja origem seja alguma máquinas pertencentes a máquina física monitorada (própria máquina física ou máquinas virtuais hospedadas) não passaram por fragmentação, ou seja, poderão ter tamanhos maiores que a *Maximum Transmission Unit* (MTU) definida. Porém, pacotes que cheguem à *bridge*, cuja origem seja externa a máquina física, deverão sofrer fragmentação caso tenham tamanho maior que MTU, resultando em várias comunicações de tamanho menor ou igual a MTU definida.

4.2 CONVERSÃO DE TRAÇOS

Nesta seção, será abordada a conversão dos arquivos de traços de monitoramento de rede (*pcap-savefile*) e traço de monitoramento de CPU e memória, descritos na seção anterior, para dois outros formatos de traço: i) traço de visualização no formato *Pajé* (Schnorr et al., 2015), que busca representar todas as máquinas e eventos de interesse do ambiente monitorado e; ii) traço de reprodução, uma simplificação do formato *Pajé*, visando o processo de geração de cargas sintéticas de trabalho guiadas.

Serão exploradas os dados extraídos dos arquivos de traço de monitoramento, os formatos dos arquivos resultantes do processo assim como a sua organização.

4.2.1 Leitura de Traços de Monitoramento

Como descrito na seção 4.1, a etapa de monitoramento produz 3 tipos de arquivos:

- O arquivo *environmnets.json*, que contém a estrutura geral do ambiente monitorado de forma hierárquica, começando pelas máquinas física até as máquinas virtuais.
- Arquivos de texto *cpu_mem_output.txt* para cada máquina física, contendo as amostragens da utilização de CPU e de memória de cada máquina virtual em intervalos de tempo regulares.
- Arquivos *network_output.pcap* para cada máquina física, registrando o tráfego de pacotes pelas *bridges* definidas em cada máquina física e utilizadas tanto pela máquina física quando pelas máquinas virtuais.

Dessa forma, o programa *converter.py* (definido sobre a versão 3.9 da linguagem *Python*) seguirá as seguintes etapas para a extração, conversão e unificação dos dados contidos nestes arquivos de traço de monitoramento.

Primeiramente será efetuada a leitura do arquivo *environmnets.json*, de forma a recuperar e armazenar em um dicionário os dados referentes ao ambiente monitorado (hierarquia e metadados de máquinas físicas e máquinas virtuais). Em seguida, são definidos todos os caminhos para os arquivos traço de monitoramento a serem lidos (usuário deverá indicar caminho de diretório de saída de arquivos resultantes de processo de monitoramento) e caminhos para arquivos de traço gerados pelo processo de conversão.

Após recuperadas e definidas todas as informações relativas a cada elemento do ambiente monitorado, serão inicializadas rodadas de leitura de arquivos de traço de monitoramento e escrita de traços de reprodução e visualização para cada máquina virtual, seguindo a ordem definida em *environments.json*.

4.2.1.1 Conversão de Traços de CPU e Memória

Como definido na seção 4.1.4, cada arquivo *cpu_mem_output.txt* conterá primeiramente um cabeçalho, indicando o início do monitoramento e o tempo corrente, as demais linhas contendo as amostras extraídas de cada máquina virtual em intervalos regulares, e finalmente um rodapé, indicando o fim do monitoramento.

Dessa forma, considerando uma máquina virtual (como por exemplo **MV1**), ao efetuar a leitura do traço de sua máquina física (**MF1**), primeiramente será definido como tempo inicial o momento em que é inicializado o monitoramento (indicado na linha de cabeçalho) de forma que todos os traços gerados conterão o intervalo de tempo entre esse tempo de início até o tempo indicado em *cpu_mem_output.txt* para a coleta de cada amostra.

Assim, para cada intervalo de coleta de amostra, serão definidas duas entradas no arquivo *MV1.trace* (no formato de traço de reprodução dedicado a atividade de máquina **MV1**). A primeira indicando o consumo de CPU e a segunda indicando o consumo de memória para o intervalo, seguindo o seguinte modelo:

```
[Código PajeSetVariable] [Timestamp] [Nome MV] CPU [Porcentagem]
[Código PajeSetVariable] [Timestamp] [Nome MV] MEM [kB]
```

Onde `[Código PajeSetVariable]` indica o código para definição de valor de variável em formato *Pajé* (indicado pelo valor 8, previamente definido); `[Timestamp]` é o intervalo de tempo entre amostragem analisada e início de monitoramento; `[Nome MV]` o nome

da máquina virtual (extraído de *environmnets.json*); o nome da variável a ter seu valor atualizado (CPU / MEM); e `[Porcentagem]` e `[kB]` o valor da variável (consumo médio de CPU para o intervalo ou utilização de memória em momento de coleta em kB).

Um exemplo do arquivo *MV1.trace* pode ser observado na Figura 4.9, onde as linhas 3 e 4 indicam os valores das variáveis CPU e MEM da máquina virtual *MV1* para um mesmo intervalo de amostragem (0.000063s à 0.503123s).

```
1  8 6.3e-05 MV1 CPU 0.00
2  8 6.3e-05 MV1 MEM 330772
3  8 0.503123 MV1 CPU 30.00
4  8 0.503123 MV1 MEM 330772
5  8 1.002925 MV1 CPU 40.00
6  8 1.002925 MV1 MEM 330772
7  8 1.502861 MV1 CPU 100.00
8  8 1.502861 MV1 MEM 775777
9  8 2.002915 MV1 CPU 100.00
10 8 2.002915 MV1 MEM 775777
11 8 2.502876 MV1 CPU 100.00
12 8 2.502876 MV1 MEM 775777
13 8 3.002833 MV1 CPU 100.00
14 8 3.002833 MV1 MEM 775777
15 8 3.503092 MV1 CPU 0.00
16 8 3.503092 MV1 MEM 330756
17 8 4.002843 MV1 CPU 0.00
18 14 3.736526 LINK root MV1 90 MV1:MV2|0
19 8 4.002843 MV1 MEM 330756
20 8 4.503117 MV1 CPU 0.00
21 8 4.503117 MV1 MEM 330724
```

Figura 4.9: Exemplo de arquivo de traço de reprodução para máquina virtual exemplo *MV1*

4.2.1.2 Conversão Traços de Tráfego de Rede

Para efetuar a conversão dos arquivos de traço de monitoramento de rede utilizou-se da biblioteca *Scapy* (Biondi e the Scapy community, 2021) para linguagem *Python*, possibilitando a leitura dos arquivos de *pcap-savefile*.

Assim, para uma máquina virtual, busca-se o arquivo *.pcap* relacionado ao monitoramento da *bridge* de sua máquina física (por exemplo, **MF1**). Dessa forma, itera-se sobre as entradas de pacotes registrados neste arquivo, primeiramente filtrando por aqueles que possuem cabeçalho IP, de forma que seja possível identificar as máquinas envolvidas na comunicação. Após isso, buscam-se pacotes cujo endereço IP de origem seja a máquina de interesse (por exemplo, a máquinas virtual **MV1**), caso este tipo de pacote seja encontrado, é definido uma nova entrada no arquivo de traço de execução no formato:

```
[Código PajeStartLink] [Timestamp] LINK root [Nome MV] [Tamanho] [Key]
```

Onde `[Código PajeStartLink]` indica o código para definição de início de um *link* em formato *Pajé* (indicado pelo valor 14, previamente definido); `[Timestamp]` é o intervalo de tempo entre a captura do pacote e início de monitoramento (utiliza-se o mesmo de *cpu_mem_output.txt*); `LINK e root`, valores estáticos para tipo de LINK e *container* (definições *Pajé*); `[Nome MV]` o nome do *container* (máquina virtual) onde foi inicializada a comunicação origem do pacote); `[Tamanho]` indicando o tamanho total do pacote

(cabeçalhos + *payload*); `[Key]`, *string* única que marcará a correspondência entre traços de envio de mensagem (*PajeStartLink*) e recebimento da mesma mensagem (*PajeEndLink*, que será definido somente posteriormente em arquivo *root.trace*), essa etiqueta segue o formato *NOME_ORIGEM:NOME_DESTINO|ORDEM*, sendo *NOME_ORIGEM* o nome da origem do pacote, *NOME_DESTINO* o nome do destino do pacote e *ORDEM* a ordem da mensagem enviada entre as máquinas.

Quanto a ordem, traços indicando o tráfego de pacotes são registrador em arquivo de traço entre entradas que indicam o consumo de CPU e de memória que compreendem o intervalo de tempo no qual aquele pacote foi enviado.

Isso pode ser identificado na Figura 4.9, onde o envio de um pacote definido na linha 18 registrado no tempo 3.736526 segundos ocorre no intervalo de consumo de CPU e memória nas linhas 17 e 18 (entre 3.503092 e 4.002843 segundos).

Caso seja identificado um pacote cuja origem ou destino seja desconhecida (ou seja, o qual o IP não esteja contido em arquivo *environmnets.json*), é definida um nova entrada de traço no arquivo *unknown_host.trace*. Isso indicará a existência de troca de pacotes entre máquina virtual de ambiente monitorado e algum elemento externo.

4.2.1.3 Traços para Visualização

Após a geração de todos os traços de reprodução, caso o programa *converter.py* seja inicializado com parâmetro *-g*, também será realizada a geração do arquivo *root.trace* (traço de visualização em formato *Pajé*). Neste arquivo, estará contido o cabeçalho identificando todas os eventos *Pajé* necessários para interpretação de eventos do traço, a definição dos *containers* (máquinas físicas e virtuais), variáveis (CPU e MEM) e LINK (LINK). Também é realizada a inicialização de todos os *containers* (definindo a hierarquia das máquinas físicas e virtuais) e variáveis. Dessa forma, todos os traços de reprodução previamente definidos são adicionados ao arquivo *root.trace*, efetuando não apenas a cópia das entradas, mas também adicionando o recebimento (*PajeEndLink*, indicado pelo código 14) correspondente a cada entrada de envio de pacote registrada (*PajeStartLink*, indicado pelo código 15).

O arquivo *root.trace* relativo aos de exemplos pode ser visualizado no Apêndice A. Nele, primeiramente definem-se os eventos *pajé* (linhas 1 à 52), posteriormente definindo *containers*, as variáveis e evento LINK (linhas 58 à 65), definindo a hierarquia dos *containers* e inicializando as variáveis CPU e MEM (linhas 70 à 79) e finalmente agregando traços (linhas 85 ao fim). Importante notar a presença não somente do *PajeEndLink* (código 14) mas também de sua contraparte *PajeStartLink* (código 15), indicadas nas linhas 102 e 103.

Ao final deste processo, uma vez que o arquivo *root.trace* atende aos requisitos do formato de traço *Pajé*, é possível utilizar da ferramenta ViTE (Coulomb et al., 2022) para efetuar a visualização da representação gráfica do ambiente monitorado, seus componentes e dos eventos registrados.

A Figura 4.10 exemplifica a visualização utilizando o programa *ViTE* do arquivo *root.trace* para exemplo discutido. Nela é possível observar a disposição das máquinas seguindo a hierarquia do ambiente monitorado, onde *root* é o *container* base (*PajeCreateContainer* base para comportar demais *containers*), seguido das máquinas físicas *MF1* e *MF2* e suas máquinas virtuais, *MV1* e *MV2*, respectivamente, além do *container Unknown*, que é utilizado para representar comunicações entre o ambiente monitorado e máquinas externas a ele. Ademais, à direita dos *containers* é possível observar o consumo dos recursos de CPU e memória (Linhas horizontais) ao decorrer do tempo de monitoramento (indicado por linha do tempo em parte superior da imagem), assim como uma a troca de uma mensagem tendo origem em máquina virtual *MV1* e destino em máquina virtual *MV2*. Finalmente, ao se clicar na área de observação,

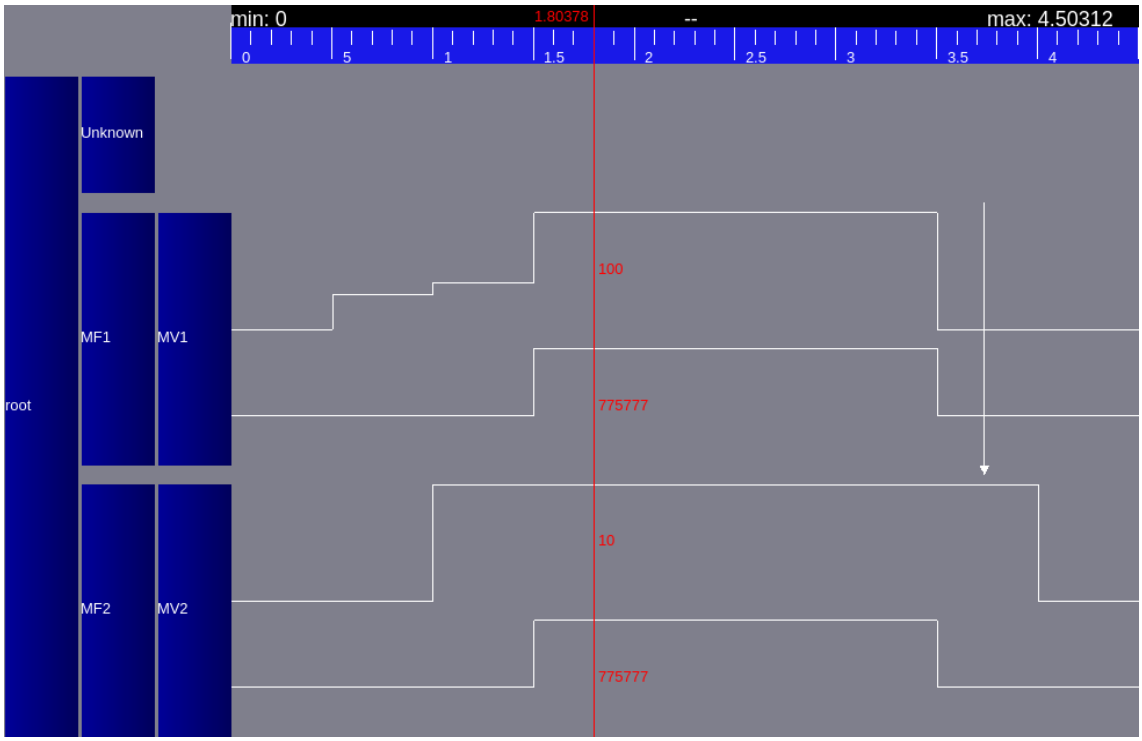


Figura 4.10: Visualização ViTE para *root.trace* de exemplo.

é possível observar valores definido para as variáveis CPU e memória de cada uma das máquinas virtuais, indicado pela linha vertical vermelha, em tempo 1.80378, indicando consumo 100% de CPU e 77577 kB de memória para *MV1*e 10% de CPU e 77577 kB de memória para *MV2*.

5 VALIDAÇÃO DA PROPOSTA

Neste capítulo serão exploradas as capacidades das ferramentas propostas através de 4 avaliações:

- **Distorção de Medidas:** abordada na Seção 5.1, onde será explorada a assertividade no período de coleta de dados para a ferramenta de monitoramento proposta. Neste experimento, será averiguado um eventual deslocamento entre o intervalo de tempo ideal previsto entre amostragens e o intervalo de tempo real entre amostragens.
- **Sobrecusto de Monitoramento:** abordado na Seção 5.2, será analisado o impacto do sistema de monitoramento sobre a execução de processos em máquinas virtuais de ambiente monitorado. Neste experimento, será estudado eventuais impactos que o ambiente de monitoramento possa gerar na execução das aplicações do ambiente de nuvem.
- **Conversão de Arquivos:** abordado na Seção 5.3, serão avaliadas as métricas de espaço de armazenamento e tempo para o processo de conversão traços de monitoramento em traços de reprodução e visualização. Neste experimento, serão analisados os arquivos gerados durante o monitoramento bem como seu pós-processamento.
- **Comportamento de Aplicações:** abordado na Seção 5.4, será analisada a visualização gráfica resultante de processo de monitoramento. Neste experimento, será verificado se é possível a extração de sub-traços isolados, considerando um ambiente contendo múltiplas máquinas virtuais pertencentes a diferentes usuários.

Para o desenvolvimento dos experimentos, serão utilizadas 8 máquinas físicas para hospedagem de máquinas virtuais. Cada uma dessas máquinas físicas contém processador Intel(R) Core(TM) 2 Duo E8400 executando na frequência de 3.00GHz e 2 módulos de 2 GiB de memória DRR-3 1066 MHz (totalizando 4 GiB de memória RAM). O SO instalado é Ubuntu 16.04.7 LTS versão servidor. Estas máquinas serão identificadas como *MojitoX* sendo *X* um inteiro utilizado como identificador e que varia entre 1 a 8.

Para a configuração das máquinas virtuais, cada uma estará configurada com o 1 Ubuntu 16.04.7 LTS versão de servidor, sendo alocado 1 núcleo de processamento virtual e 2 GiB de memória. Cada máquina virtual será identificada como *testvmX-Y* onde *X* corresponde a sua máquina física *X* e *Y* serve como seu identificador de máquina virtual contida dentro da máquina física *X* (variando de 1 a 2).

Além das máquinas físicas *Mojito* 1 a 8, utilizadas como hospedeiras, também estará presente nos experimentos a máquina *Mojito0* contendo processador Intel(R) Core(TM) i5-2400 CPU executando na frequência de 3.10GHz e 2 módulos de 2 GiB de memória DRR-3 1333 MHz (totalizando 4 GiB de memória total). Esta máquina será utilizada como *frontend* no processo de monitoramento e sua configuração será relevante apenas em experimento Conversão e Arquivos (Seção 5.3), sendo esta utilizada no processo de conversão.

Para realização dos experimentos **Sobrecusto de Monitoramento** e **Comportamento de Aplicações**, foram selecionadas 3 as aplicações MPI do conjunto disponível pelo *NAS Parallel Benchmarks* (NPB) em sua versão 3.1.3 (NASA, 2022): i) *Embarrassingly Parallel* (EP) que efetuará operações de ponto flutuante em cada um dos seus processos, minimizando a comunicação entre eles (Bailey et al., 1991); ii) *Conjugate Gradient* (CG), onde será calculada

uma aproximação para o menor autovalor de uma matriz definida positiva simétrica esparsa (Bailey et al., 1991); iii) *Data Traffic* (DT), que realizará troca de pacotes entre seus processos seguindo padrão de um grafo predeterminado (Mallón et al., 2009). Dessa forma, cada uma das aplicações foi selecionada dada sua característica de sobrecarga de CPU, memória e tráfego de dados, respectivamente, como exposto por Blanche e Lundqvist (2015) e NASA (2022).

Finalmente, a fim de diminuir o impacto da variação da frequência dos processadores durante os experimentos, foi utilizada da ferramenta *cpupower* (Brown e Renninger, 2011) para definir cada um dos processadores em modo *performance*. Dessa forma, fixando a frequência de operação na máxima oferecida (3.0 GHz para máquinas utilizadas na hospedagem e 3.1 GHz para máquina *frontend*).

5.1 DISTORÇÃO DE MEDIDAS

Nesta seção, será apresentado o experimento com o objetivo de observar a variação entre o intervalo de tempo ideal e intervalo de tempo real entre amostragens de CPU e memória, verificando também o impacto que outras cargas de trabalho nos ambientes virtuais possam exercer sobre essa medida. Dessa forma, foram definidos 2 cenários de experimentos, o primeiro contendo o mínimo de recursos ativos utilizados em ambiente monitorado, e a segundo com grande utilização de recursos computacionais pelas máquinas virtuais.

No primeiro cenário, foi definida apenas 1 máquina virtual por máquina física, estando tanto as máquinas virtuais quantas as máquinas física em modo ocioso, executando o mínimo de processos possível, ou seja, apenas aqueles requeridos pelo SO.

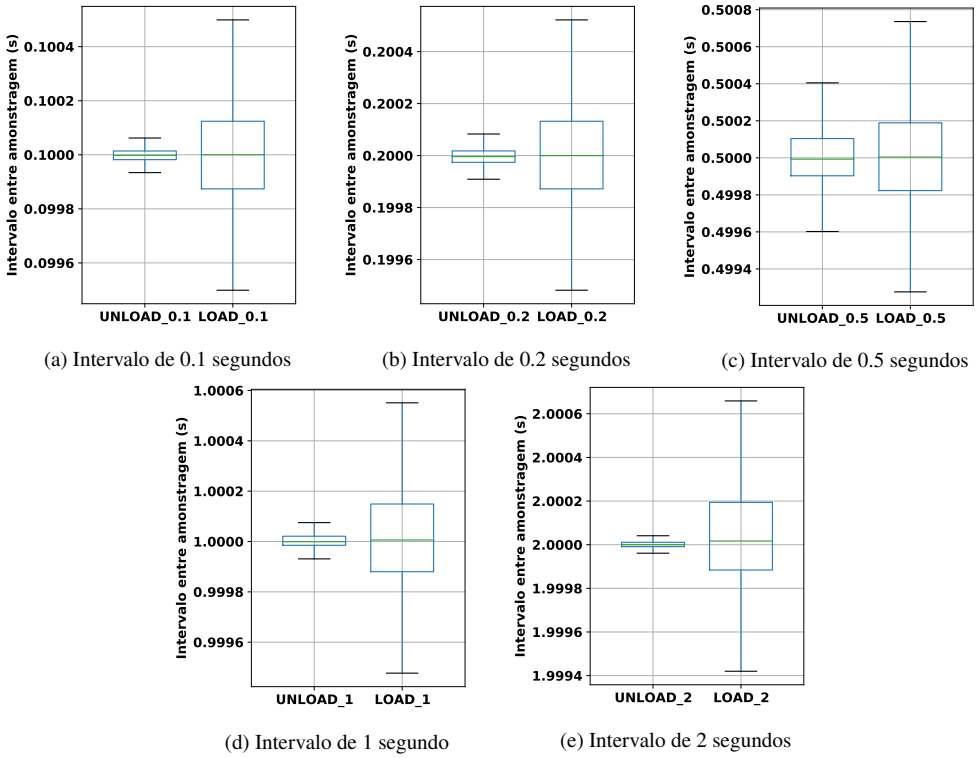
Para a segundo cenário, foram definidas 2 máquinas virtuais por máquina física, cada qual executando o programa *stress-ng* (King, 2022) de forma que cada máquina virtual estresse 1 núcleo de processamento e 2 GiB de memória (perspectiva das máquinas virtuais).

Para cada um desses cenários, foram realizados processos de monitoramento, variando o intervalo de tempo entre as amostragens de CPU e memória entre os intervalos de 0,1, 0,2, 0,5, 1 e 2 segundos. Para cada combinação de cenário e intervalo de amostragem, foram realizadas 1000 execuções do processo de monitoramento (a fim de eliminar possíveis anomalias em resultados), cada qual durando aproximadamente 60 segundos (1 minuto).

Finalizado o processo de monitoramento, para cada combinação de cenário e intervalo de amostragem, calcula-se o intervalo de tempo entre amostragens de CPU e memória em cada processo de monitoramento. Após isso, se comparam os resultados de cada cenário (ambiente ocioso e ambiente carregado) com os respectivos intervalos de monitoramento estabelecidos.

Tabela 5.1: Métricas para variações de monitoramento sobre aplicação CG. Tabela evidenciando Média, Desvio Padrão e Coeficiente de variação para cada experimento em forma ocioso (indicado por *UNLOAD_X*) e sob carga (indicado por *LOAD_X*) para os intervalos de monitoramento 0.1, 0.2, 0.5, 1 e 2 segundos.

Experimento	Média (s)	Desvio padrão (s)	Coeficiente de variação (%)
UNLOAD_0.1	0,100007	0,000200	0,199933
LOAD_0.1	0,100018	0,017724	17,721206
UNLOAD_0.2	0,200014	0,000256	0,128104
LOAD_0.2	0,200033	0,019927	9,961720
UNLOAD_0.5	0,500034	0,000351	0,070202
LOAD_0.5	0,500086	0,015032	3,005866
UNLOAD_1	1,000074	0,000406	0,040623
LOAD_1	1,000165	0,001451	0,145059
UNLOAD_2	2,000147	0,000580	0,028975
LOAD_2	2,000334	0,001675	0,083750



(a) Intervalo de 0.1 segundos (b) Intervalo de 0.2 segundos (c) Intervalo de 0.5 segundos

(d) Intervalo de 1 segundo (e) Intervalo de 2 segundos

Figura 5.1: Gráficos analisando distorção em intervalos de amostragem, onde são comparados par a par a cada uma dos experimentos para distintos intervalos de monitoramento em para cenários ocioso (indicado por *UNLOAD_X*) com sua contraparte sob carga (indicado por *LOAD_X*).

Observando os gráficos indicados na Figura 5.1, é possível identificar que apesar do aumento no coeficiente de variância no intervalo entre amostragens da aplicação de sonda de CPU e memória, quando se comparam experimentos em ambientes sem sobrecarga (*UNLOAD_X*) com seus pares sob cargas de trabalho intensa (*LOAD_X*), os gráficos para cada uma das partes se sobrepõem, indicando equivalência estatística entre os pares de ambientes monitorados. Além disso, em conjunto com tabela 5.1, constata-se que a média de intervalos de amostragens atem-se a intervalo solicitado para cada um dos experimentos, ocorrendo erro abaixo de 0,0004 segundos para todas as médias. Outro ponto interessante é a variância, onde no pior caso (*LOAD_0.1*), obteve-se uma variância de 17,72% para cenário sob carga de trabalho intensa, além de ser evidente a diminuição do coeficiente de variação conforme o intervalo entre amostragens é aumentado considerando-se os cenários com e sem carga.

Dessa forma, podemos concluir que, apesar do impacto das cargas de trabalho aplicadas em máquinas virtuais de ambiente monitorado, a aplicação de sonda de CPU e memória se mantém estável em todos os intervalos de monitoramento propostos.

5.2 SOBRECUSTO DE MONITORAMENTO

Nesta seção, serão abordados experimentos com o objetivo de mensurar os impactos do processo de monitoramento sobre processos executados em máquinas virtuais, avaliando as métricas de tempo e troca de contexto em cenários com e sem o processo de monitoramento para 3 aplicações MPI.

Foi utilizada a ferramenta *perf* (Linux Kernel Organization, 2022) para extração das métricas *task-clock* (indicando o tempo total em que processo estava sendo executado em processador), *context-switches* (indicando a quantidade de trocas de contexto sofridas por cada

processo), *cpu-migrations* (indicando a quantidade de migrações de núcleo sofridas por processo) e *time elapsed* (indicando o tempo decorrido entre o início e o fim do processo). A aplicação *perf* foi executada sobre a chamada do processo gerente de processos MPI, contido em máquina virtual *testvm1-1*, responsável por gerenciar demais processos relativos a execução de cada aplicação MPI executados na *testvm1-1* assim como nas demais máquinas virtuais estabelecidas.

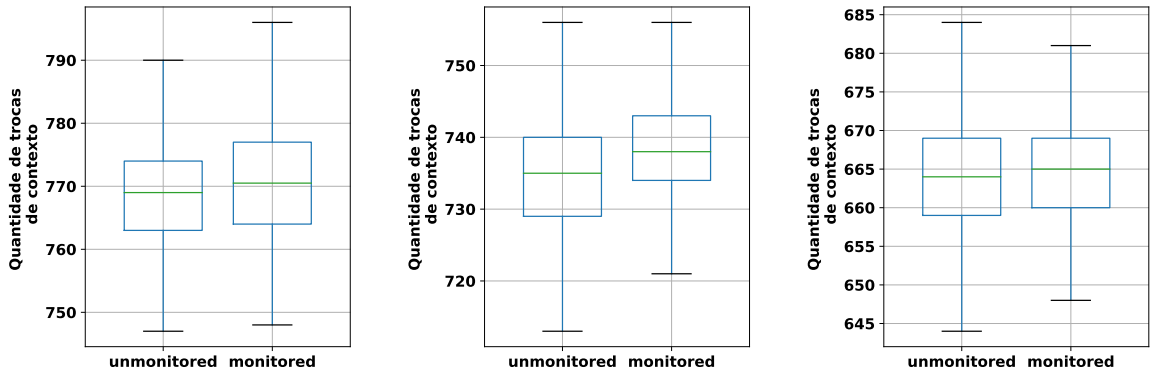
Foram estabelecidos 2 cenários para o ambiente. No primeiro cenário, cada máquina hospedeira contém apenas 1 máquina virtual ativa (*testvmX-1*), de forma a sobrarem recursos para aplicação de monitoramento em cada uma das máquinas físicas. Nesse cenário são utilizadas 8 máquinas virtuais para execução de processos MPI. Para o segundo cenário, foram estabelecidas 2 máquinas virtuais (*testvmX-1* e *testvmX-2*) por máquina física , de forma a estabelecer concorrência por recursos entre as máquinas virtuais e os processos da *RECloud*.

Foram utilizadas as aplicações EP, CG e DT do conjunto de aplicações NAS. As aplicações EP e CG foram configuradas dados de entrada tamanho A, e DT para tamanho S utilizando grafo BH para cenário com 8 máquinas virtuais e classe W com grafo BH para cenário com 16 máquinas virtuais.

Para o processo de sonda de CPU e memória, foi fixado o intervalo para amostragens em 0,5 segundos.

Para cada combinação de cenário, aplicação e estado de ambiente (sob sistema de monitoramento ou sem monitoramento) foram realizadas 1000 execuções das aplicações, a fim de evitar o impacto de possíveis variações nos resultados finais.

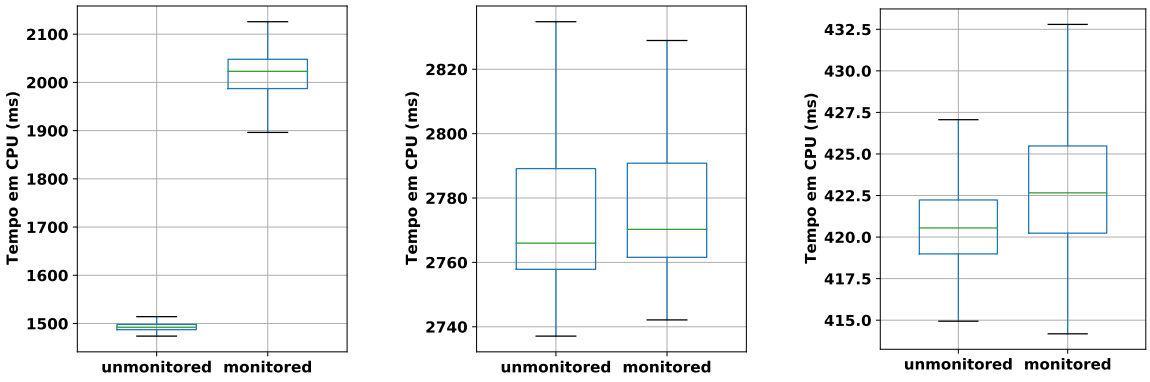
Ao final do processo, foram coletadas as métricas fornecidas através do *perf* para cada combinação de cenário, aplicação e presença de monitoramento. Após isso, comparam-se os resultados para ambientes com e sem a presença da ferramenta monitora.



(a) Trocas de contexto CG tamanho A (b) Trocas de contexto EP tamanho A (c) Trocas de contexto DT tamanho S grafo BH

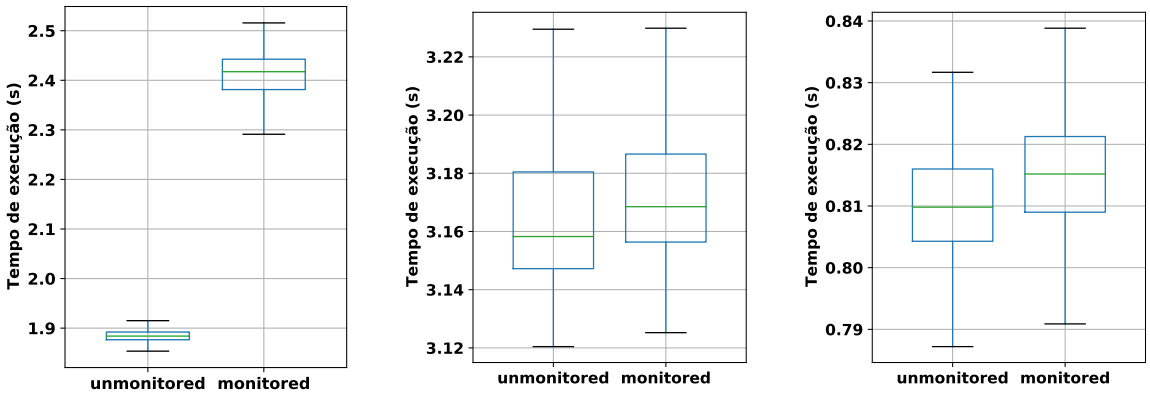
Figura 5.2: Gráficos analisando impacto de monitoramento sobre troca de contexto para 8 máquinas virtuais. São comparados par a par a quantidade de trocas de contexto (evento *context-switches* de *perf*) para aplicações CG de tamanho A, EP de tamanho A, e DT de tamanho S para grafo BH em ambiente monitorado contendo 8 máquinas virtuais, 1 por máquina física.

A partir da análise dos resultados, utilizando conjuntos de gráficos indicados nas Figuras 5.2 e 5.5, observa-se que, considerando pares de experimentos para utilização ou não da ferramenta de monitoramento, apesar de elevação de trocas de contexto em mediana (exceto para EP e DT com 16 processos como indicado nas Figuras 5.5(c) e 5.5(b)), os resultados ainda se sobrepões, indicando equivalência estatística entre os pares. Porém, quando analisam-se as métricas de Tempo em CPU (conjuntos de gráficos na Figura 5.3 e Figura 5.6) e Tempo de execução (conjuntos de gráficos nas Figuras 5.4 e 5.7) nota-se que, para diversas aplicações EP e DT apesar do aumento na mediana de tempo, tanto em CPU quanto total, ainda existe equivalência estatística entre cenários com e sem monitoramento. Entretanto, quando observam-se os gráficos



(a) Tempo CPU para CG tamanho A (b) Tempo CPU para EP tamanho A (c) Tempo CPU para DT tamanho S grafo BH

Figura 5.3: Gráficos analisando impacto de monitoramento sobre tempo em CPU para 8 máquinas virtuais. São comparados par a par o tempo em CPU (evento *task-clock* de *perf*) de contexto para aplicações CG de tamanho A, EP de tamanho A, e DT de tamanho S para grafo BH em ambiente monitorado contendo 8 máquinas virtuais, 1 por máquina física.



(a) Tempo total para CG tamanho A (b) Tempo total para EP tamanho A (c) Tempo total para DT tamanho S grafo BH

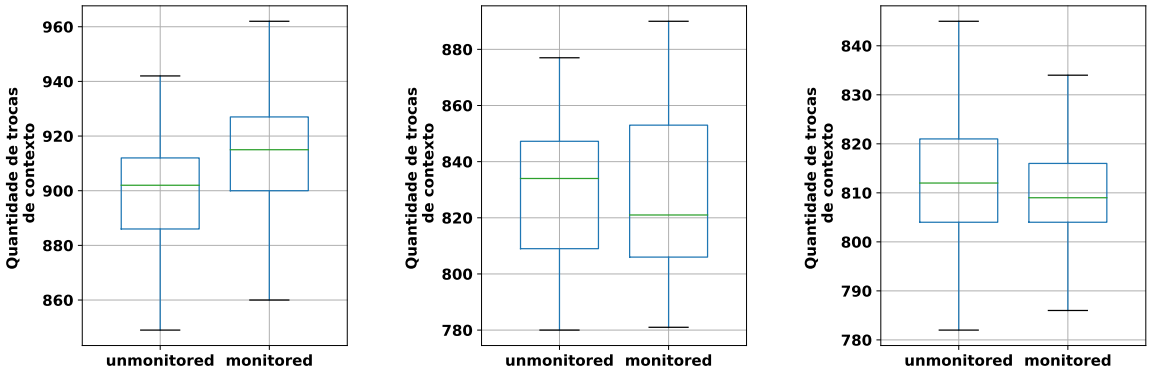
Figura 5.4: Gráficos analisando impacto de monitoramento sobre tempo total de execução para 8 máquinas virtuais. São comparados par a par o tempo total de execução (*elapsed time* de *perf*) para aplicações CG de tamanho A, EP de tamanho A, e DT de tamanho S para grafo BH em ambiente monitorado contendo 8 máquinas virtuais, 1 por máquina física.

de tempo relacionado a aplicação CG, verifica-se total disjunção entre os resultados obtidos, evidenciando grande impacto do processo de monitoramento nas sobre as execuções de CG independente da quantidade de máquinas virtuais ativas em máquina física.

Buscando explorar a diferença observada, estabeleceu-se um novo cenário de testes, a fim de apurar o impacto do processo de monitoramento sobre a aplicação CG. Dessa forma, definiu-se ambiente contendo 16 máquinas virtuais (2 por máquina física) e variou-se o intervalo entre amostragens de CPU e memória em 0.1, 0.2, 0.5 1 e 2 segundos de intervalo. Foram executadas iterações de execução da aplicação CG para cada variação de intervalo de tempo, extraindo-se as métricas *task-clock* de e *elapsed-time* com a ferramenta *perf*.

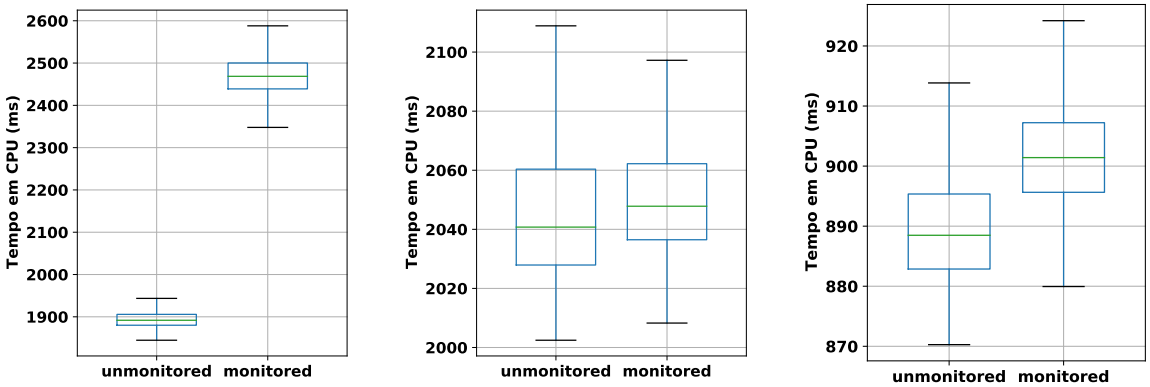
Assim, após extraídas métricas para os diferentes cenários, comparou-se cada experimento com a atuação de ferramenta de monitoramento com cenário equivalente (16 máquinas virtuais, 2 por máquina hospedeira) e sem a presença do monitor.

Desse modo, por meio da Figura 5.8, é possível observar que, com o aumento do intervalo entre amostragens de sonda de CPU, atenua-se o efeito negativo do processo de monitoramento sobre execução de aplicação CG. Porém, também é possível constatar que o ganho de desempenho



(a) Trocas de contexto CG tamanho A (b) Trocas de contexto EP tamanho A (c) Trocas de contexto DT tamanho W grafo BH

Figura 5.5: Gráficos analisando impacto de monitoramento sobre trocas de contexto para 16 máquinas virtuais. São comparados par a par a quantidade de trocas de contexto (evento *context-switches* de *perf*) para aplicações CG de tamanho A, EP de tamanho A, e DT de tamanho W para grafo BH em ambiente monitorado contendo 16 máquinas virtuais, 2 por máquina física.



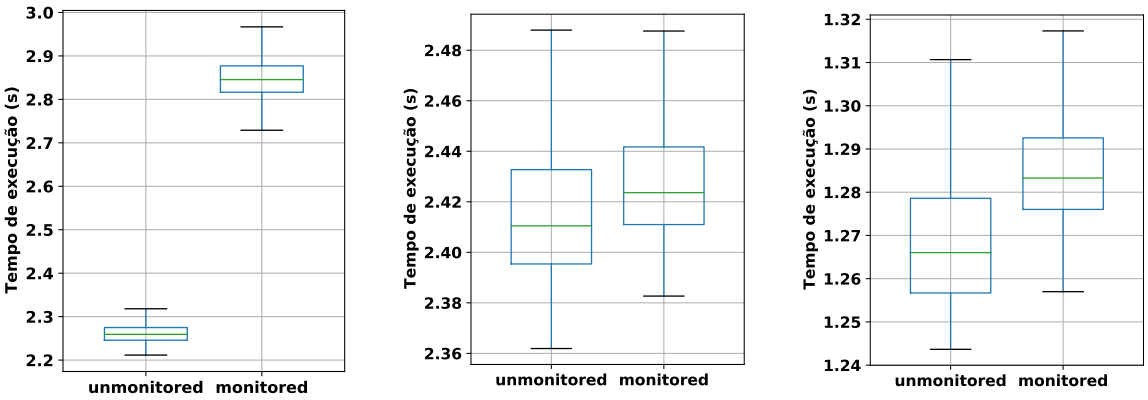
(a) Tempo CPU de CG tamanho A (b) Tempo CPU de EP tamanho A (c) Tempo CPU de DT tamanho W grafo BH

Figura 5.6: Gráficos analisando impacto de monitoramento sobre tempo em CPU para 16 máquinas virtuais. São comparados par a par o tempo em CPU (evento *task-clock* de *perf*) de contexto para aplicações CG de tamanho A, EP de tamanho A, e DT de tamanho W para grafo BH em ambiente monitorado contendo 16 máquinas virtuais, 2 por máquina física.

Tabela 5.2: Tempo total de execução de aplicação CG para 16 processos variando-se o intervalo de amostragem. Nela indica-se o tempo total (segundos) de execução de aplicação CG de tamanho A para 16 máquinas virtuais variando-se cenário sem presença de monitor e cenários contendo monitor em variados intervalos de amostragem de CPU e memória.

Intervalo de monitoramento (s)	**Média (s)**	**Desvio padrão (s)**	**Coeficiente de variação (s)**	**Acréscimo médio em tempo de execução (%)**
unmonitored	2,266091	0,062458	0,027562	0
0,10	2,928549	0,065430	0,022342	29,23
0,20	2,876928	0,054601	0,018979	26,95
0,50	2,846040	0,045460	0,015973	25,59
1,00	2,819699	0,064417	0,022845	24,43
2,00	2,805962	0,051488	0,018350	23,82

devido ao aumento do intervalo de monitoramento tende a estagnação em patamares acima do desempenho observado para aplicação CG sem a presença das sondas. Isso é corroborado por resultados apresentados na tabela 5.2, onde por meio de *Intervalo de monitoramento (s)* e *Acréscimo médio em tempo de execução (%)* é possível perceber que o aumento no valor



(a) Tempo Total para CG tamanho A

(b) Tempo Total para EP tamanho A

(c) Tempo Total para DT tamanho W grafo BH

Figura 5.7: Gráficos analisando impacto de monitoramento sobre tempo total de execução para 16 máquinas virtuais. São comparados par a par o tempo total de execução (*elapsed time* de *perf*) para aplicações CG de tamanho A, EP de tamanho A, e DT de tamanho W para grafo BH em ambiente monitorado contendo 16 máquinas virtuais, 2 por máquina física.

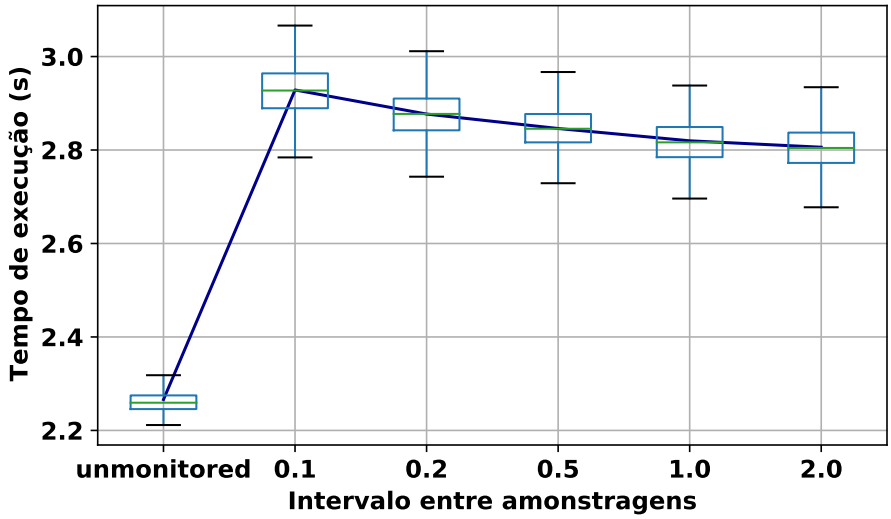


Figura 5.8: Gráfico de tempo total de execução de aplicação CG tamanho A comparando cenários com e sem monitoramento (*unmonitored*). O ambiente contém 16 máquinas virtuais, sendo 2 por máquina física.

do intervalo entre amostragens feitas à sonda de CPU e memória não é proporcionalmente acompanhado pelo ganho de desempenho de aplicação CG.

Sendo assim, conclui-se que apesar de sonda de CPU e memória afetar desempenho das aplicações executadas em máquinas virtuais, em cenários cuja aplicação executada relacione-se com características de aplicação CG (longa duração e grande tráfego de pacotes), a sonda de rede utilizada por monitor (*tcpdump*) tende a exercer maior estresse sobre o ambiente monitorado que o processo da sonda de CPU e Memoria.

5.3 CONVERSÃO DE ARQUIVOS

Nesta seção, será explorado o processo de conversão de arquivos em traços de monitoramento para traços de reprodução e visualização. Serão abordados o tamanho dos arquivos utilizados como entrada, o tamanho dos arquivos obtidos como saída e o tempo de conversão destes arquivos.

Para a execução desta tarefa, foram utilizadas 16 máquinas virtuais, 2 por máquina física. A partir disso, estabeleceu-se 3 cenários para monitoramento: i) cada máquina virtual executa um processo da aplicação MPI CG de classe B; ii) cada máquina virtual executa um processo da aplicação MPI EP de classe B; iii) cada máquina virtual executa um processo da aplicação MPI DT de classe W com grafo BH. Em todos os cenários foi estabelecido 0,5 segundos como intervalo entre amostragens para sonda de CPU e memória.

Dessa forma, uma vez que todos os arquivos de monitoramento estivessem a disposição, foram efetuadas as conversões sobre os arquivos traço de monitoramento de cada um dos experimentos. Utilizou-se do programa *perf* em conjunto com o processo de conversão a fim de efetuar capturar as métricas *task-clock* (indicando o tempo de processo escalonado em processador) e *time elapsed* (indicando o tempo decorrido entre o início e o fim do processo).

Após o processo de conversão, foram extraídos o tamanho total dos arquivos de traço de monitoramento de CPU e memória, traço de monitoramento de rede, traço de execução e traço visualização por experimento. Além disso, foi feita a contagem do total de pacotes capturados por experimento, assim como a quantidade total de linhas de arquivos de traço de monitoramento de CPU e memória.

Para o processo de conversão, foi utilizada a implementação alternativa para *Python 3*, *Pypy* em sua versão 3.7 (The PyPy Team, 2022) a fim de obter melhores resultados para a execução da aplicação de conversão de traços.

Reiterando que, como indicado em introdução de capítulo, o processo de conversão foi efetuado pela máquina *frontend mojito0*, que possui configuração sutilmente superior as máquinas de físicas utilizadas para osa outros experimentos.

Tabela 5.3: Resultados de experimentos para processo de conversão de traços. Apresentam-se expostos os resultados para processo de conversão de traços de Monitoramento para traços de reprodução e traços de visualização relativos ao monitoramentos de ambiente de nuvem executando aplicações CG de tamanho B, EP de tamanho B e DT de tamanho W utilizando grafo BH.

Experimento	**cg,B**	**ep,B**	**dt,W BH**
Tamanho total traços monitoramento (MB)	857,82	0,71	0,97
Taxa conversão (MB/s)	0,93	0,11	0,16
Relação tamanho entrada (MB) / saída (MB)	3,09	1,53	1,48
Número de mensagens capturadas (pcap)	8269010	7227	9528
Quantidade de amostras CPU e MEM	640	168	72
Tempo de monitoramento (s)	39,50	10,00	4,00

Utilizando a tabela 5.3 é possível quantificar algumas métricas para o processo de conversão traços dos diferentes cenários. Primeiramente, utilizando *Relação tamanho entrada(MB)/saída (MB)*, observamos a redução na quantidades de dados necessária para representação das atividades monitoradas. Isso se apresenta em todos os experimentos mas principalmente para experimento CG, onde traços de reprodução somados a traços de visualização ocupam 3 vezes menos espaço que traços de monitoramento, otimizando a representação das atividades. Outro ponto é a *Taxa conversão (MB/s)*, onde o experimento *cg.B* obteve o melhor desempenho, processando uma média de 0.9256 MB por segundo. Assim, utilizando gráfico da Figura 5.9 é possível inferir que o processo de conversão dos traços de rede pode ser proporcionalmente mais rápido que a conversão dos traços de CPU e memória, já que quanto maior a quantidade de pacotes por amostras de CPU e memória, maior é a taxa de conversão total.

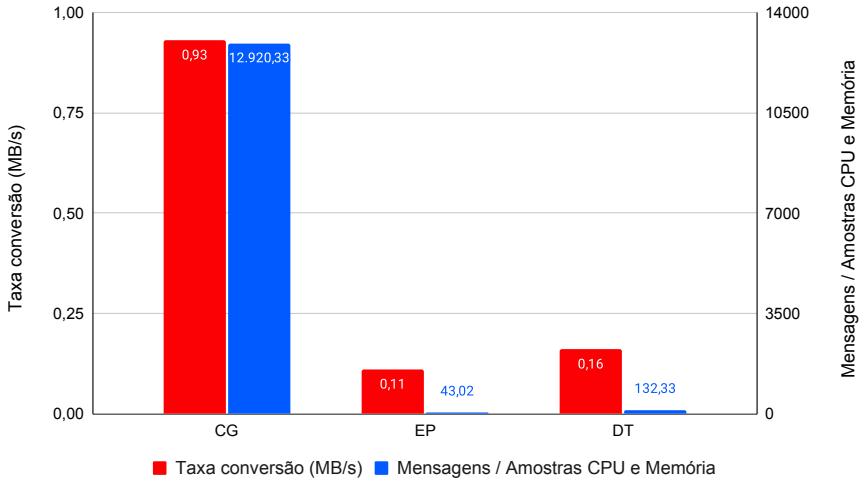


Figura 5.9: Gráfico correlacionando Taxa de Conversão e Proporção de Mensagens/Amostras CPU e memória.

5.4 COMPORTAMENTO DE APLICAÇÕES

No experimento abordado nesta seção, será analisado o comportamento de um ambiente de nuvem monitorado através visualização das atividades registradas. Será utilizada a ferramenta de visualização gráfica ViTE, tendo como entrada arquivo *root.trace*, produto de processos de monitoramento e conversão, discutido em Seção 4.2.1.3.

Para a execução desse experimento, foram definidas 2 máquinas virtuais por máquina física. As máquinas virtuais foram divididas em 2 grupos, o grupo A contendo todas as máquinas com terminação 1 (*testvmX-1*) e o grupo B contendo todas as máquinas com terminação 2 (*testvmX-2*), de forma que cada máquina física contivesse uma máquina virtual pertencente a cada grupo. Assim, emulou-se um cenário onde dois usuários (A e B) estariam com suas máquinas virtuais escalonadas lado a lado, distribuídos em diversas máquinas físicas.

Dessa forma, foi definido para cada um dos grupos uma aplicação MPI do conjunto NPB, onde máquinas do grupo A executaram a aplicação CG de classe B para 8 processos e as máquinas do grupo B executaram a aplicação EP de classe B também para 8 processos.

Primeiramente, inicializou-se o processo de monitoramento tendo como *frontend* a máquina *mojito0* e como máquinas hospedeiras monitoradas as máquinas físicas *mojito1* à *mojito8*. Em seguida iniciou-se a execução do processo CG em máquina virtual *testvm1-1*, chamando o processo em todas as máquinas de grupo A assim como o processo EP em máquina virtual *testvm1-2* e demais máquinas do grupo B.

Uma vez que ambos os processos terminasse suas execuções, finalizou-se o processo de monitoramento, recuperando os arquivos de traço de monitoramento para todas as máquinas físicas. Após isso, foram gerados arquivos de traço de execução assim como arquivo de traço de visualização.

Finalmente, utilizou-se o programa ViTE para efetuar a visualização gráfica do traço de visualização *root.trace*, extraindo-se as seguintes visões do mesmo processo.

Considerando as visões extraídas, utilizando o programa *ViTE* a partir de arquivo de traço de visualização *root.trace* para o experimento, é possível observar que, mesmo em um cenário de monitoramento de nuvem aparentemente caótico, como pode ser observado na Figura 5.10, é possível ao usuário manipular a visualização dos *containers* monitorados como indicado na Figura 5.11, onde são omitidas as setas indicando troca de mensagens e é possível identificar padrão no consumo de CPU e de memória das máquinas virtuais, onde máquinas

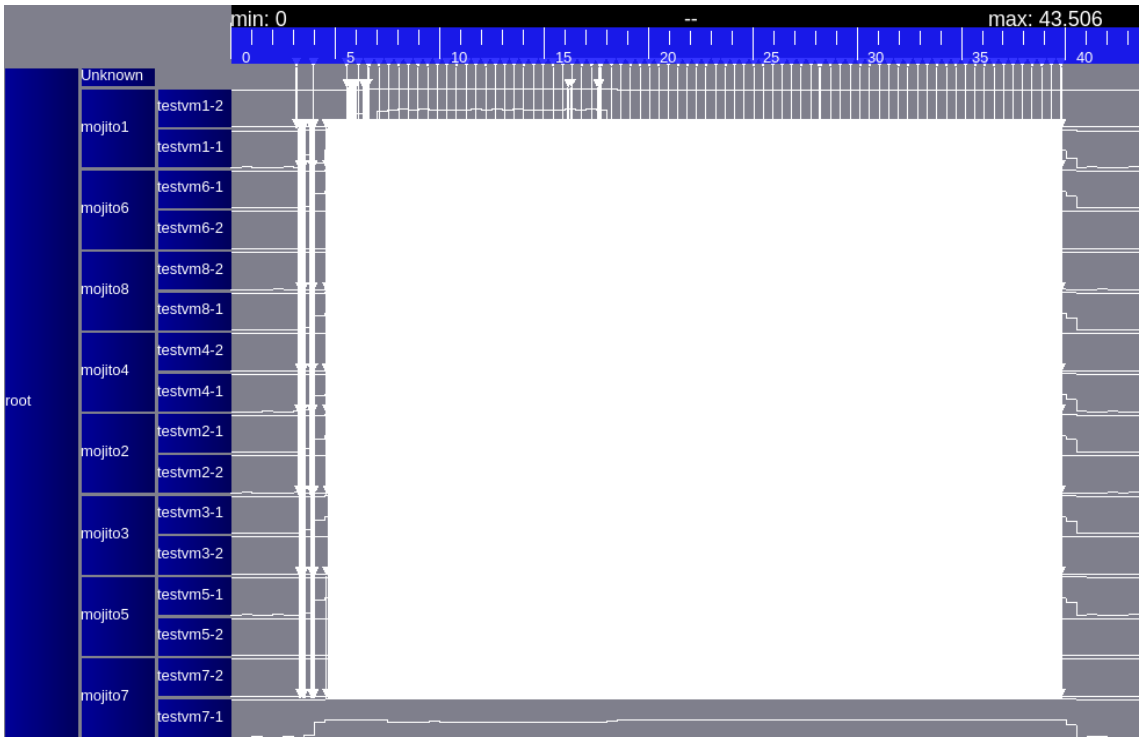


Figura 5.10: Visualização completa de experimento utilizando ViTE, contendo todos os elementos e métricas monitoradas.

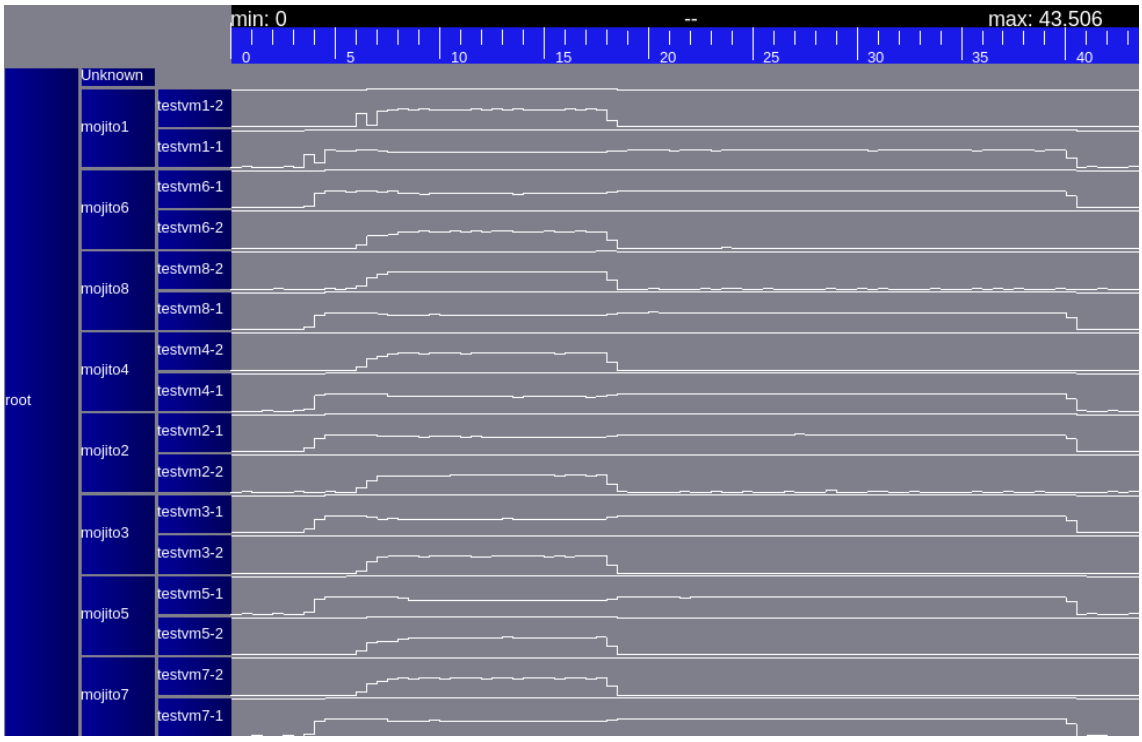


Figura 5.11: Visualização completa de experimento utilizando ViTE, contendo todos os elementos e omitindo setas representando comunicações.

virtuais *testvmX-1* efetuam de consumo de recursos em tempos semelhantes assim como máquinas virtuais *testvmX-2*.

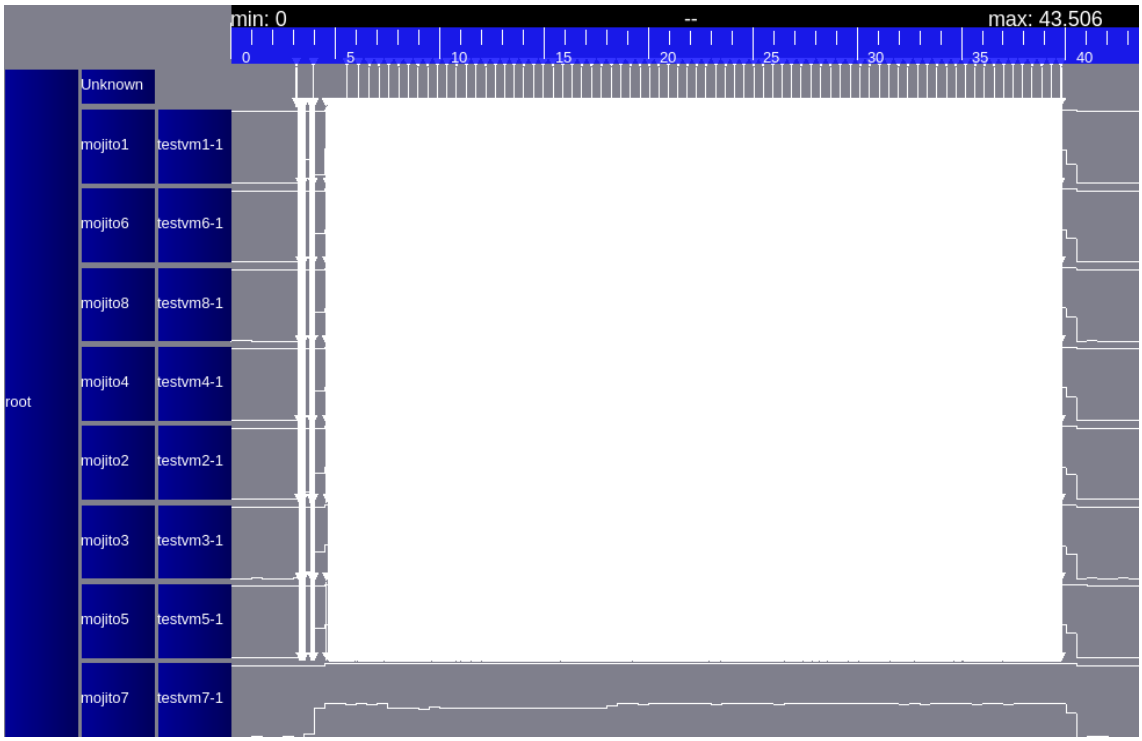


Figura 5.12: Visualização parcial de experimento utilizando ViTE, contendo apenas máquina físicas e virtuais componentes de grupo A.

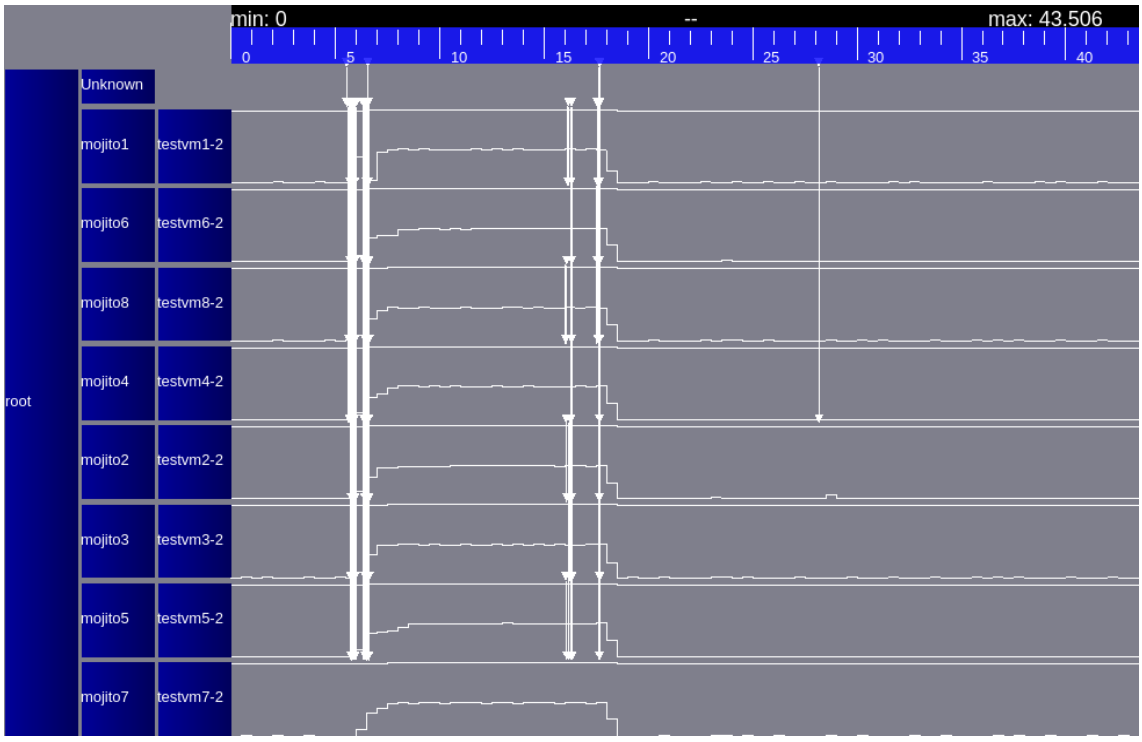


Figura 5.13: Visualização parcial de experimento utilizando ViTE, contendo apenas máquina físicas e virtuais componentes de grupo B.

Além disso, é possível ao usuário identificar visualmente máquinas comunicantes, como os indicados nas Figuras 5.12 e 5.13, possibilitando a identificação visual de máquinas virtuais que venham a ser controladas por um mesmo usuário em sistema de nuvem monitorado.

6 CONCLUSÕES E TRABALHOS FUTURO

Este trabalho propôs a ferramenta RECloud com o objetivo de efetuar o monitoramento de ambientes de nuvem *IaaS*, efetuando a captura do consumo de CPU, memória e do trafego de pacotes de rede exercido por máquinas virtuais em traços de monitoramento ao longo do tempo. Além disso, a partir dos traços de monitoramento gerados, converte-los em traços de reprodução e visualização, que descrevam o comportamento observado em nuvem monitorada pela perspectiva de máquinas virtuais individuais assim como de todo o ambiente.

Ao final, a partir dos resultados obtidos, é possível afirmar que a ferramenta *RECloud* é capaz de efetuar a captura de atividades *job-unspecific* referentes ao consumo de CPU, memória e tráfego de rede realizadas por máquinas virtuais de um ambiente de nuvem em traços de monitoramento.

Dessa forma, também foi demonstrada a capacidade de conversão de traços de monitoramento em traços de visualização em formato Pajé (Schnorr et al., 2015), possibilitando que comportamentos observados em ambiente monitorado sejam utilizados como entrada por ferramentas que aceitem este formato (como a ferramenta *ViTE* utilizada para confecção deste trabalho) ou sejam manipulados por usuário, dado que o formato é aberto. Além disso, a partir de traços de reprodução gerados ao final do processo de conversão, fica em aberto a possibilidade de desenvolvimento de novos trabalhos que busquem simular ambiente de nuvem monitorado, a fim de possibilitar a reprodução e compartilhamento de experimentos realizados em ambiente de nuvem, fomentando o quesito reprodutibilidade destes experimentos.

Outro ponto a ser aprofundado a partir deste trabalho é a análise de performance da ferramenta proposta, aplicando-a a ambientes de nuvem cujas máquinas físicas possuam novas tecnologias de suporte a virtualização via *hardware*, como Intel *Extended Page Tables* (Bhatia, 2009) ou AMD-V *Nested Page Table* (Virtualization, 2008). Assim, efetuando sua avaliação em conjunto a componentes que promovem ganho de performance a ambientes virtualizados, e que não estavam presentes no ambiente utilizado para experimentos contidos neste trabalho.

Ainda, quanto ao aperfeiçoamento dos mecanismos propostos. Dada a limitação de sistema de monitoramento atual ater-se apenas a um ambiente de nuvem estático (sem modificação em conjunto de máquinas virtuais), observa-se a possibilidade de aprimoramento em sistema de monitoramento, de forma que esse seja capaz de identificar e tratar de eventos comuns em ambientes de nuvem como a criação, inicialização, desligamento e migração de máquinas virtuais.

Quanto ao quesito performance, como pode ser visto nos resultados de experimentos, apesar de que na maioria dos cenários propostos a ferramenta de monitoramento demonstrar comportamento estável, foi possível observar impacto significativo da ferramenta quando monitoramento ambiente em que aplicação CG de conjunto NPB estivera em execução. Isso indica espaço para aprimoramento tanto de sonda de CPU e memória desenvolvida, como também a oportunidade para definição de uma nova sonda de rede especializada para projeto, efetuando a substituição de ferramenta *tcpdump*. Além disso, para a ferramenta de conversão de traços, dada a sua construção sobre a linguagem *Python*, fica evidente uma possível migração de linguagem, de forma a obter melhores resultados de desempenho para a conversão de traços em *single-thread* tanto quanto possibilitar a sua execução de forma paralela *multi-thread*.

Finalmente, ficam a disposição os códigos utilizados para a definição das ferramentas propostas em https://github.com/Daffc/RECloud.

REFERÊNCIAS

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V. e Weeratunga, S. K. (1991). The nas parallel benchmarks summary and preliminary results. Em *Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*.

Bhatia, N. (2009). Performance evaluation of intel ept hardware assist. *VMware, Inc*.

Biondi, P. e the Scapy community (2021). Scapy project. `https://www.tcpdump.org/`. Acessado em 30/04/2022.

Blanche, A. e Lundqvist, T. (2015). Addressing characterization methods for memory contention aware co-scheduling. *J. Supercomput.*, 71(4):1451–1483.

Brown, L. e Renninger, T. (2011). Cpupower(1). `https://man.archlinux.org/man/cpupower.1.en`. Acessado em 10/07/2022.

Corradi, A., Foschini, L., Povedano-Molina, J. e Lopez-Soler, J. M. (2012). Dds-enabled cloud management support for fast task offloading. Em *2012 IEEE Symposium on Computers and Communications (ISCC)*, páginas 67–74.

Coulomb, K., Faverge, M., Jazeix, J., Lagrasse, O., Marcoueille, J., Noisette, P., Redondy, A., Thibault, S. e Vuchener, C. (2022). Visual trace explorer (ViTE). `http://vite.gforge.inria.fr`. Acessado em 30/04/2022.

Curnow, R. e Lichvar, M. (2021). Chrony - introduction. `https://chrony.tuxfamily.org/`. Acessado em 30/04/2022.

Foundation, F. S. (2022a). 19 mathematics. `https://www.gnu.org/software/libc/manual/html_node/Mathematics.html`. Acessado em 30/04/2022.

Foundation, F. S. (2022b). 21 date and time. `https://www.gnu.org/software/libc/manual/html_node/Date-and-Time.html`. Acessado em 30/04/2022.

Foundation, F. S. (2022c). 3.11 options that control optimization. `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html#Optimize-Options`. Acessado em 30/04/2022.

Hamza, O., Boukhobza, J., Singhoff, F. e Rubini, S. (2014). A Multilevel I/O Tracer for Timing and Performance Analysis of Storage Systems in IaaS Cloud. Em *3rd IEEE Real-Time and Distributed Computing in Emerging Applications (REACTION)*.

Hillbrecht, R. e Bona, L. (2012). A snmp-based virtual machines management interface. Em *Utility and Cloud Computing, IEEE Internatonal Conference on*, páginas 279–286, Los Alamitos, CA, USA. IEEE Computer Society.

H.Yang e Tate, M. (2012). A descriptive literature review and classification of cloud computing research. *Communications of the Association for Information Systems*, 31(1):2.

Kao, W.-I. e Iyer, R. (1992). A user-oriented synthetic workload generator. Em *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*, páginas 270–277.

King, C. I. (2022). stress-ng. `https://github.com/ColinIanKing/stress-ng`. Acessado em 10/07/2022.

LibVirt (2022). LibVirt, the virtualization API. `http://libvirt.org/`. Acessado em 30/04/2022.

Linux Kernel Organization (2022). perf: Linux profiling with performance counters. `https://perf.wiki.kernel.org/index.php/Main_Page`. Acessado em 10/07/2022.

Lyon, G. (2022). Nmap network scanning. `https://nmap.org/book/`. Acessado em 30/04/2022.

Mallón, D. A., Taboada, G. L., Touriño, J. e Doallo, R. (2009). Npb-mpj: Nas parallel benchmarks implementation for message-passing in java. Em *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*.

Maziero, C. (2020). Sistemas operacionais: Conceitos e mecanismos. `http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-livro.pdf`.

Mell, P. e Grance, T. (2011). The NIST Definition of Cloud Computing. *National Institute of Standards and Technology Special Publication 800-145*.

NASA (2022). NAS Parallel Benchmarks. `https://www.nas.nasa.gov/software/npb.html`. Acessado em 10/07/2022.

Oracle (2022). What is java technology and why do i need it? `https://www.java.com/en/download/help/whatis_java.html`. Acessado em 16/07/2022.

Popek, G. J. e Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Commun. ACM*.

Righi, R. D. R., Rodrigues, V., Nardin, I., Costa, C., Alves, M. A. Z. e Pillon, M. A. (2019). Towards providing middleware-level proactive resource reorganisation for elastic HPC applications in the cloud. *Int. Journal of Grid and Utility Computing*, 10:76–92.

Schnorr, L. M., de O. Stein, B., Faverge, M., Trahay, F. e de Kergommeaux, J. C. (2015). Pajé-trace file format. `https://raw.githubusercontent.com/schnorr/pajeng/master/doc/lang-paje/lang-paje.pdf`.

Tarasov, V., Kumar, S., Ma, J., Hildebrand, D., Povzner, A., Kuenning, G. e Zadok, E. (2012). Extracting flexible, replayable models from large block traces. Em *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, página 22, USA. USENIX Association.

Tcpdump Group (2022a). Man page of pcap-savefile. `https://www.tcpdump.org/manpages/pcap-savefile.5.html`. Acessado em 30/04/2022.

Tcpdump Group (2022b). Tcpdump & libcap. `https://www.tcpdump.org/`. Acessado em 30/04/2022.

The PyPy Team (2022). PyPy. `https://www.pypy.org/`. Acessado em 10/07/2022.

Virtualization, A. (2008). Amd-v nested paging. *White paper.[Online] Available: http://sites. amd. com/us/business/it-solutions/virtualization/Pages/amd-v. aspx*.

Xu, L. e Yang, J. (2011). A management platform for eucalyptus-based iaas. Em *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, páginas 193–197.

APÊNDICE A – EXEMPLO *ROOT.TRACE*

```
%EventDef PajeDefineContainerType 0
%      Alias string
%      Type string
%      Name string
%EndEventDef
%EventDef PajeDefineVariableType 1
%      Alias string
%      Type string
%      Name string
%      Color color
%EndEventDef
%EventDef PajeDefineLinkType 4
%      Alias string
%      Type string
%      StartContainerType string
%      EndContainerType string
%      Name string
%EndEventDef
%EventDef PajeCreateContainer 6
%      Time date
%      Alias string
%      Type string
%      Container string
%      Name string
%EndEventDef
%EventDef PajeDestroyContainer 7
%      Time date
%      Type string
%      Name string
%EndEventDef
%EventDef PajeSetVariable 8
%      Time date
%      Container string
%      Type string
%      Value double
%EndEventDef
%EventDef PajeStartLink 14
%      Time date
%      Type string
%      Container string
%      StartContainer string
%      Value string
%      Key string
%EndEventDef
%EventDef PajeEndLink 15
%      Time date
%      Type string
%      Container string
%      EndContainer string
%      Value string
%      Key string
%EndEventDef

```

```
# ---------------------------------------
# ---------- Defining Entities ----------
# ---------------------------------------

0 ROOT 0 "ROOT"
0 NODE ROOT "NODE"
0 VM NODE "VM"

1 MEM VM "Memory" "0 0 0"
1 CPU VM "CPU" "0 0 0"

4 LINK 0 VM VM "LINK"

# ---------------------------------------
# -- Initializing Container & Variables --
# ---------------------------------------
6 0.0 root ROOT 0 root
6 0.0 Unknown NODE root Unknown
6 0.0 MF1 NODE root MF1
6 0.0 MV1 VM MF1 MV1
8 0.0 MV1 MEM 0
8 0.0 MV1 CPU 0
6 0.0 MF2 NODE root MF2
6 0.0 MV2 VM MF2 MV2
8 0.0 MV2 MEM 0
8 0.0 MV2 CPU 0

# ----------------------------------------
# --- Aggregating Virtual Machines Data ---
# ----------------------------------------
# --- MV1 ---
8 6.3e-05 MV1 CPU 0.00
8 6.3e-05 MV1 MEM 330772
8 0.503123 MV1 CPU 30.00
8 0.503123 MV1 MEM 330772
8 1.002925 MV1 CPU 40.00
8 1.002925 MV1 MEM 330772
8 1.502861 MV1 CPU 100.00
8 1.502861 MV1 MEM 775777
8 2.002915 MV1 CPU 100.00
8 2.002915 MV1 MEM 775777
8 2.502876 MV1 CPU 100.00
8 2.502876 MV1 MEM 775777
8 3.002833 MV1 CPU 100.00
8 3.002833 MV1 MEM 775777
8 3.503092 MV1 CPU 0.00
8 3.503092 MV1 MEM 330756
8 4.002843 MV1 CPU 0.00
14 3.736526 LINK root MV1 90 MV1:MV2|0
15 3.736526 LINK root MV2 90 MV1:MV2|0
8 4.002843 MV1 MEM 330756
8 4.503117 MV1 CPU 0.00
8 4.503117 MV1 MEM 330724

# --- MV2 ---
8 6.3e-05 MV2 CPU 0.00
8 6.3e-05 MV2 MEM 330772
8 0.503123 MV2 CPU 0.00
```

```
8 0.503123 MV2 MEM 330772
8 1.002925 MV2 CPU 10.00
8 1.002925 MV2 MEM 330772
8 1.502861 MV2 CPU 10.00
8 1.502861 MV2 MEM 775777
8 2.002915 MV2 CPU 10.00
8 2.002915 MV2 MEM 775777
8 2.502876 MV2 CPU 10.00
8 2.502876 MV2 MEM 775777
8 3.002833 MV2 CPU 10.00
8 3.002833 MV2 MEM 775777
8 3.503092 MV2 CPU 10.00
8 3.503092 MV2 MEM 330756
8 4.002843 MV2 CPU 0.00
8 4.002843 MV2 MEM 330756
8 4.503117 MV2 CPU 0.00
8 4.503117 MV2 MEM 330724


# ------------------------------------------------
# --- Aggregating Unknown Host Communications ---
# ------------------------------------------------
```