



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

Giulia Ciprandi

Monitoramento na Computação de Névoa e Borda do Contínuo
Nuvem-Coisas: Um Estudo Comparativo entre PCP e Prometheus

Florianópolis
2026

Giulia Ciprandi

Monitoramento na Computação de Névoa e Borda do Contínuo
Nuvem-Coisas: Um Estudo Comparativo entre PCP e Prometheus

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Eletrônica do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Eletrônica.
Orientador: Prof. Dr. Odorico Machado Mendizabal

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Ciprandi, Giulia

Monitoramento na Computação de Névoa e Borda do Contínuo Nuvem-Coisas : Um Estudo Comparativo entre PCP e Prometheus / Giulia Ciprandi ; orientador, Odorico Machado Mendizabal, 2026.

92 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia Eletrônica, Florianópolis, 2026.

Inclui referências.

1. Engenharia Eletrônica. 2. Computação em Borda. 3. Computação em Névoa. 4. Contínuo Nuvem- Coisas. 5. Monitoramento de Desempenho. I. Mendizabal, Odorico Machado. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. III. Título.

Giulia Ciprandi

Monitoramento na Computação de Névoa e Borda do Contínuo
Nuvem-Coisas: Um Estudo Comparativo entre PCP e Prometheus

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Bacharel em Engenharia Eletrônica” e aprovado em sua forma final pelo Curso de Graduação em Engenharia Eletrônica.

Florianópolis, 03 de julho de 2026.



Documento assinado digitalmente
Fernando Rangel de Sousa
Data: 30/06/2026 14:47:54-0300
CPF: ***.649.114-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Dr. Fernando Rangel de Sousa
Coordenador do Curso

Banca Examinadora:



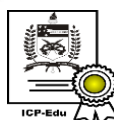
Documento assinado digitalmente
Odorico Machado Mendizabal
Data: 29/06/2026 21:24:10-0300
CPF: ***.941.170-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Dr. Odorico Mendizabal
Orientador



Documento assinado digitalmente
PATRICIA DELLA MEA PLENTZ
Data: 01/07/2026 22:08:48-0300
CPF: ***.243.700-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof^a. Dr^a. Patricia Della Méa Plentz
Avaliador(a)
Universidade Federal de Santa Catarina



Documento assinado digitalmente
Richard Demo Souza
Data: 30/06/2026 14:52:07-0300
CPF: ***.267.379-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Dr. Richard Demo Souza
Avaliador(a)
Universidade Federal de Santa Catarina

À minha mãe Doralice das Graças Benedito e meu pai
Olivio Ciprandi, por dar-me asas e ladrilhar o caminho. E
à professora Daniela Suzuki, que segurou a minha mão por
um instante apenas, breve e imenso.

AGRADECIMENTOS

Agradeço à minha mãe Doralice das Graças Benedito, que me criou com tanto carinho e ensinou-me a curiosidade pelo mundo, ao meu pai Olivio Ciprandi, pelas brincadeiras de criança, piadocas e suporte ao longo de toda a minha vida. Entrei neste curso em 2014 mas não o concluí, retornei como egressa, e ao longo desta trajetória de 12 anos, há incontáveis amigas e amigos a quem agradeço. Em especial, Clarissa Vaz Ferro e Letícia Ventura dos Santos — com vocês é que aprendi a ser no mundo; à Carolina Lemos e Tamara Busato, com quem compartilho ambições confidências; ao João Pedro da Silva Segundo, que encorajou esta empreitada ensandecida. Agradeço à paciência e disponibilidade do meu orientador, Odorico Mendizabal, aos membros da banca, aos membros da coordenação, aos membros do Centro Acadêmico, aos funcionários da secretaria, aos técnicos e terceirizados, ao professor Marcio Cherem Schneider e ao Laboratório de Circuitos Integrados (LCI), a Mobilis - Veículos Elétricos, e à própria a Universidade Federal de Santa Catarina, que — posso dizer com segurança após estes 12 anos — é tão viva quanto nós.

*“Para quem quer se soltar
Invento o cais
Invento mais que a solidão me dá
Invento Lua nova a clarear
Invento o amor
E sei a dor
De me lançar
Eu queria ser feliz
Invento o mar
Invento em mim
O sonhador
Para quem quer me seguir
Eu quero mais
Tenho o caminho do que sempre quis
E um Saveiro pronto pra partir
Invento o cais
E sei a vez
De me lançar”*

(Milton Nascimento e Ronaldo Bastos, 1972)

RESUMO

A proliferação de dispositivos conectados à Internet, estimada em 29,3 bilhões em 2023, impulsionou o desenvolvimento de paradigmas de computação distribuída que complementam a computação em nuvem, como a computação em borda e em névoa. Esses paradigmas, sintetizados pelo conceito de contínuo nuvem-coisas, aproximam o processamento dos dispositivos finais e atendem a requisitos que a arquitetura centralizada da nuvem não consegue satisfazer. Nesse cenário, o monitoramento de sistemas é uma funcionalidade importante para a orquestração de serviços, pois coleta o estado atualizado dos recursos e habilita as decisões de gerenciamento; contudo, o próprio sobrecusto introduzido pelas ferramentas de monitoramento pode comprometer a operação em dispositivos com capacidade computacional limitada. Este trabalho faz uma revisão sobre os paradigmas computacionais relevantes na literatura, com foco na computação em borda e em névoa. Neste contexto, busca também análises sobre monitoramento e métricas relevantes, bem como soluções de monitoramento já existentes, definindo taxonomias relevantes para ambas.

À partir disso, realiza-se uma avaliação experimental, utilizando como metodologia um *benchmark*, entre duas ferramentas de monitoramento de sistemas Linux, o *Performance Co-Pilot* (PCP) e a *stack* Prometheus, incluindo *Node Exporter*, avaliando o sobrecusto que cada uma impõe sobre uma máquina com recursos constritos, visando contemplar casos de nós borda. Os experimentos foram conduzidos no *testbed* Emulab, utilizando o *Yahoo! Cloud Serving Benchmark* (YCSB) para gerar cargas transacionais sobre o sistema de banco de dados MySQL 8.4, em quatro perfis de carga de trabalho e dois tamanhos de *buffer pool* (128 MB e 1 GB), comparando três configurações: sem monitoramento (*baseline*), com *pmlogger* e com o *Node Exporter*. Os resultados indicam que ambas as ferramentas impõem sobrecusto limitado, com aparente tendência de piora em termos de vazão e latência, porém da mesma ordem de grandeza que a variabilidade natural entre execuções independentes do mesmo cenário. Como encaminhamento para trabalhos futuros, recomenda-se a execução de múltiplas repetições por cenário para estimativa de variância, o aumento da granularidade de *threads* nas varreduras, avaliação mais precisa do sobrecusto da latência e avaliação de bases de dados NoSQL.

Palavras-chave: Computação em Borda. Computação em Névoa. Contínuo Nuvem-Coisas. Monitoramento de Desempenho. *Performance Co-Pilot*. Prometheus. Sobrecusto. *Benchmark*.

ABSTRACT

The proliferation of internet-connected devices, estimated at 29.3 billion in 2023, has driven the development of distributed computing paradigms that complement cloud computing, such as edge and fog computing. These paradigms, synthesized by the concept of the cloud-to-things continuum, bring processing closer to end devices and meet requirements that the centralized cloud architecture cannot satisfy. In this scenario, system monitoring is an important functionality for service orchestration, as it collects the updated state of resources and enables management decisions; however, the overhead introduced by monitoring tools can compromise operation on devices with limited computing capacity. This work reviews relevant computing paradigms in the literature, focusing on edge and fog computing. In this context, it also searches for analyses of relevant monitoring and metrics, as well as existing monitoring solutions, defining relevant taxonomies for both.

Furthermore, an experimental evaluation was conducted using a benchmark between two Linux system monitoring tools, Performance Co-Pilot (PCP) and the Prometheus stack, including Node Exporter, evaluating the overhead that each imposes on a machine with constrained resources, aiming to cover edge node cases. The experiments were conducted on the Emulab testbed, using the Yahoo! Cloud Serving Benchmark (YCSB) to generate transactional loads on the MySQL 8.4 database system, in four workload profiles and two buffer pool sizes (128 MB and 1 GB), comparing three configurations: without monitoring (baseline), with pmlogger, and with Node Exporter. The results indicate that both tools impose a limited overhead, with an apparent tendency to worsen in terms of throughput and latency, but of the same order of magnitude as the natural variability between independent executions of the same scenario. As a direction for future work, it is recommended to perform multiple iterations per scenario to estimate variance, increase the granularity of amount of threads per scan, more accurately assess the latency overhead cost, and evaluate NoSQL databases.

Keywords: Edge Computing. Fog Computing. Cloud-to-Things Continuum. Cloud Continuum. Performance Monitoring. *Performance Co-Pilot*. Prometheus. Overhead.

LISTA DE FIGURAS

Figura 1 – Crescimento global de conexões à Internet e dispositivos conectados . .	16
Figura 2 – Conexões M2M por aplicação	17
Figura 3 – Linha do tempo do surgimento dos Paradigmas Computacionais <i>Cloud</i> , IoT, Computação em Borda, Computação em Névoa e Computação em Neblina.	21
Figura 4 – Número de publicações de paradigmas computacionais ao longo do tempo.	22
Figura 5 – Número de publicações acerca de <i>Cloud Continuum</i> ao longo do tempo.	23
Figura 6 – Arquitetura de Paradigmas de Computação.	24
Figura 7 – Ciclo MAPE-K de monitoramento para orquestração e dispositivos. . .	36
Figura 8 – Métricas relevantes para cada paradigma de computação.	38
Figura 9 – Métricas comuns a todos os paradigmas de computação.	39
Figura 10 – Taxonomia de soluções de monitoramento para névoa.	44
Figura 11 – <i>Workload B</i> (95% leitura / 5% atualização) — Latência vs. vazão. . . .	57
Figura 12 – <i>Workload E</i> (95% leitura / 5% inserção) — Latência vs. vazão.	58
Figura 13 – <i>Workload A</i> (50% leitura / 50% atualização) — Latência vs. vazão. . .	58
Figura 14 – <i>Workload F</i> (<i>read-modify-write</i>) — Latência vs. vazão.	59
Figura 15 – Mapa de calor do overhead médio de monitoramento por <i>workload</i> e tamanho de <i>buffer pool</i> . Números negativos indicam redução de vazão em relação ao <i>baseline</i> ; números positivos indicam que o cenário monitorado superou o <i>baseline</i>	60

LISTA DE QUADROS

Quadro 1 – Cargas de trabalho do YCSB utilizadas no experimento.	49
Quadro 2 – Mapeamento qualitativo entre métricas do Node Exporter (Prometheus) e do PCP.	50
Quadro 3 – Métricas do Prometheus sem equivalente no PCP	51
Quadro 4 – Especificações do nó PC3000 (Emulab).	53
Quadro 5 – Especificações do nó D430 (Emulab).	53

LISTA DE TABELAS

Tabela 2 – Comparativo entre os paradigmas de computação em nuvem, borda e névoa.	23
Tabela 3 – Principais diferenças entre computação em nuvem e em borda.	27

LISTA DE ABREVIATURAS E SIGLAS

benchmark	Metodologia de avaliação de desempenho de sistemas computacionais. Vale-se de cargas de trabalho padronizadas e reproduíveis
CNCF	Cloud Native Computing Foundation
Computação em Borda	Paradigma de computação distribuída que realiza o processamento próximo à fonte dos dados, com foco em baixa latência, processamento em tempo real e análise local.
Computação em Neblina	Paradigma de computação distribuída formado pelos próprios dispositivos IoT dotados de capacidade de processamento local colaborativo. Tradução livre da autora para <i>mist computing</i>
Computação em Névoa	Paradigma de computação distribuída de propósito geral com arquitetura multicamadas que provê serviços entre os dispositivos finais e os <i>data centers</i> de nuvem, fornece computação, armazenamento e rede, posicionada em qualquer ponto entre a nuvem e os dispositivos IoT
Contínuo da Nuvem	Paradigma de computação distribuída que engloba os conceitos de computação em nuvem, computação em borda e computação em neblina e outros possíveis paradigmas correlatos
Contínuo Nuvem-Coisas	Semelhante ao Contínuo da Nuvem, mas com foco também nos dispositivos finais
IaaS	<i>Infrastructure as a Service</i>
IoIT	<i>Internet of Intelligent Things</i>
IoT	Internet das Coisas (<i>Internet of Things</i>)
iTaaS	Internet das Coisas como Serviço (<i>Internet of Things as a Service</i>)
ITU	União Internacional de Telecomunicações (<i>International Telecommunication Union</i>)
M2M	Máquina para máquina (<i>Machine to Machine</i>)
MAE	<i>Mean Absolute Error</i>
MAPE	<i>Mean Absolute Percentage Error</i>
MAPE-K	<i>Monitor, Analyze, Plan, Execute, Knowledge</i>
MCC	<i>Mobile Cloud Computing</i>
MEC	<i>Mobile Edge Computing</i>
MIT	Instituto de Tecnologia de Massachusetts (<i>Massachusetts Institute of Technology</i>)
ML	aprendizado de máquina (<i>machine learning</i>)
MSE	<i>Mean Square Error</i>
MTBF	<i>Mean Time Between Failures</i>
MTTR	<i>Mean Time To Repair</i>
NIST	Instituto Nacional de Padrões e Tecnologia (<i>National Institute of Standards and Technology</i>)

offloading	Técnica de delegação de processamento ou armazenamento de um dispositivo local a um recurso remoto (borda, névoa ou nuvem), visando reduzir a carga de trabalho e o consumo de energia locais
overhead	Sobrecusto adicional em recursos computacionais introduzido por um componente de <i>software</i> no sistema em que é executado
PaaS	<i>Platform as a Service</i>
PCP	Performance Co-Pilot
PMDA	<i>Performance Metrics Domain Agent</i>
PRED	<i>Decision Feedback Predictor</i>
Prometheus	Ferramenta de monitoramento de sistemas <i>Linux</i> de código aberto
QoS	Qualidade de Serviço (<i>Quality of Service</i>)
RAN	<i>Radio Access Network</i>
RMSE	<i>Root Mean Square Error</i>
SaaS	<i>Software as a Service</i>
SLA	Acordo de Nível de Serviço (<i>Service Level Agreement</i>)
testbed	Ambiente de testes controlado que disponibiliza recursos de hardware dedicados e topologias de rede configuráveis para que pesquisadores possam construir, depurar e avaliar redes e sistemas distribuídos de forma reproduzível
VM	Máquina Virtual (<i>Virtual Machine</i>)
YCSB	<i>Yahoo! Cloud Serving Benchmark</i>

SUMÁRIO

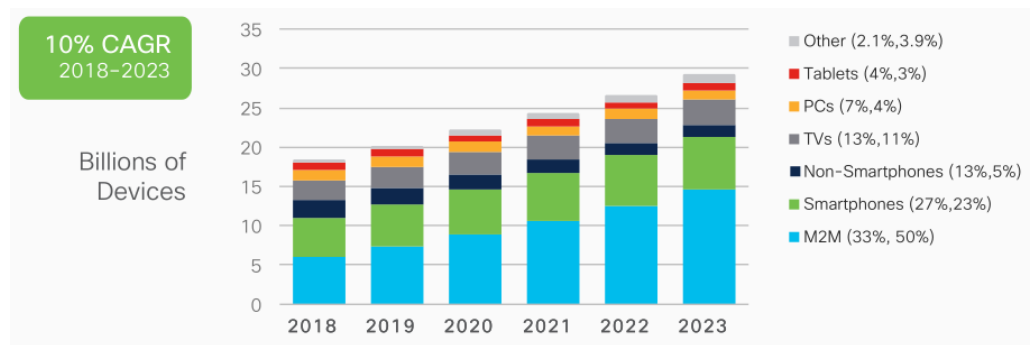
1	INTRODUÇÃO	16
1.1	OBJETIVOS	18
1.1.1	Objetivo Geral	18
1.1.2	Objetivos Específicos	18
1.2	TRABALHOS RELACIONADOS	19
1.3	USO DA INTELIGÊNCIA ARTIFICIAL NESTE TRABALHO	19
1.4	ORGANIZAÇÃO DO TRABALHO	20
2	PARADIGMAS COMPUTACIONAIS	21
2.1	ARQUITETURA DOS PARADIGMAS COMPUTACIONAIS	22
2.2	COMPUTAÇÃO EM NUVEM	25
2.2.1	Limitações	26
2.3	COMPUTAÇÃO EM BORDA	27
2.4	COMPUTAÇÃO EM NÉVOA	28
2.5	INTERNET DAS COISAS	29
2.5.1	Computação em Neblina	30
2.5.2	Desafios que Motivam Novos Paradigmas	31
2.6	CONTINUIDADE DA NUVEM (<i>CLOUD CONTINUUM</i>)	31
2.7	CONEXÕES COM A ENGENHARIA ELETRÔNICA	32
2.7.1	Sistemas Embarcados e IoT Inteligente	32
2.7.2	Redes 6G	33
3	MÉTRICAS E SOLUÇÕES DE MONITORAMENTO PARA SISTEMAS LINUX NO CONTEXTO FOG	35
3.1	O CICLO MAPE-K	36
3.2	SOBRE MÉTRICAS	37
3.2.1	Taxonomia	37
3.2.2	Métricas Relevantes	38
3.2.2.1	Métricas da Fase Monitor	38
3.2.2.2	Métricas da Fase Analyzer	41
3.2.2.3	Métricas da Fase Planner	41
3.2.2.4	Métricas da Fase Executor	42
3.3	SOBRE SOLUÇÕES DE MONITORAMENTO	42
3.3.1	Taxonomia de Soluções de Monitoramento	43
3.3.2	Análise Comparativa das Soluções Existentes	43
3.4	SOLUÇÕES DE MONITORAMENTO RELEVANTES	44
3.4.1	A Stack Prometheus	45
3.4.2	O Performance Co-Pilot (PCP)	45
4	AVALIAÇÃO EXPERIMENTAL	48

4.1	YCSB	48
4.1.0.1	Cargas de Trabalho (<i>Workloads</i>)	48
4.1.1	Seleção de Métricas	49
4.2	<i>SETUP</i> EXPERIMENTAL	51
4.2.1	O <i>Testbed</i> Emulab	51
4.2.2	Nós Utilizados	52
4.2.3	Cenários de Execução	54
4.2.3.1	<i>Buffer pool</i> de 128 MB	54
4.2.3.2	<i>Buffer pool</i> de 1 GB	54
4.3	ANÁLISE DOS RESULTADOS	55
4.4	DIFICULDADES	61
5	CONCLUSÃO	62
	REFERÊNCIAS	65
	APÊNDICE A – MÉTRICAS DO PROMETHEUS/NODE EXPORTER . .	68
	APÊNDICE B – CONFIGURAÇÃO DO PMLOGGER EQUIVA- LENTE AO <i>NODE EXPORTER</i>	79
	APÊNDICE C – MÉTRICAS DO DO PMLOGGER	82

1 INTRODUÇÃO

Devido às grandes evoluções da eletrônica nas últimas décadas, em particular referente à miniaturização progressiva de transistores e em avanços em sistemas digitais, dispositivos com capacidade de conexão à Internet tornaram-se uma ferramenta amplamente utilizada em aplicações modernas. O Cisco Annual Internet Report (2018–2023) (CISCO SYSTEMS, 2020) previu que em 2023 haveria 29,3 bilhões de dispositivos conectados à Internet, em contraste com os 18,4 bilhões de 2018, totalizando 3,6 dispositivos conectados per capita. Estimou também que conexões Máquina para máquina (*Machine to Machine*) (M2M) seriam 14,7 bilhões, conforme Figura 1, totalizando 50% do total de conexões. Destas, metade são de casas conectadas, ao passo que carros conectados são a aplicação com maior Taxa de Crescimento Anual Composta, apesar de ainda representarem uma fatia pequena do total, como ilustra a Figura 2, ainda segundo Cisco Systems (2020). O Relatório de Mobilidade da Ericsson (2026), por sua vez, dedica-se à rede móvel, e ao final de 2025 contabilizou 8,8 bilhões de dispositivos conectados à rede móvel, com 9,5 bilhões previstos para 2031. Ainda neste período, havia 4,5 bilhões de conexões Internet das Coisas (*Internet of Things*) (IoT) na rede móvel, com previsão de cerca de 7,5 bilhões de conexões IoT para 2031.

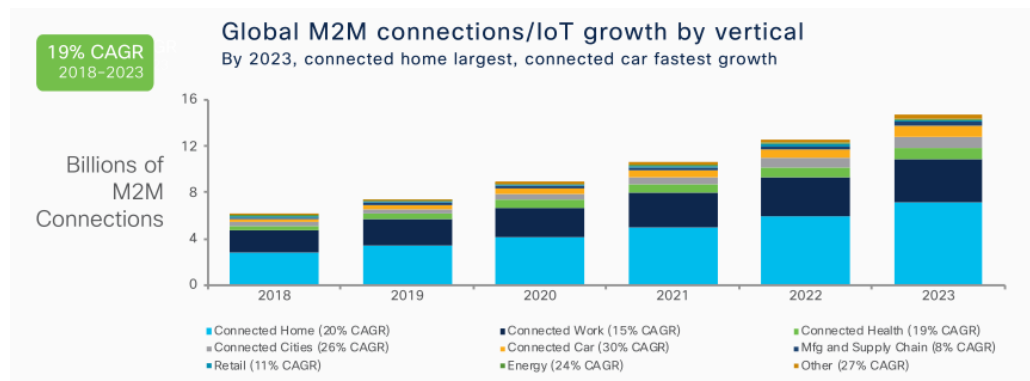
Figura 1 – Crescimento global de conexões à Internet e dispositivos conectados



Fonte: Extraído de Cisco Systems (2020)

A Computação em Nuvem (*Cloud Computing*), segundo Cao *et al.* (2020), é uma alternativa para o volume crescente de conexões. Proposta em 2006 na conferência de ferramentas de busca pelo CEO do Google, sua largura de banda não tem sido suficiente para sistemas de tempo real, além de apresentar problemas relacionados a consumo de energia, segurança dos dados e proteção de privacidade. Pensa-se então em um cenário de computação mais amplo denominado Computação Continuada na Nuvem (*cloud continuum*) (MORESCHINI *et al.*, 2022), composto e complementado por outros paradigmas, sendo eles Computação em Borda (*Edge Computing*), Computação em Névoa (*Fog Computing*) (CAO *et al.*, 2020) e, segundo algumas literaturas, tal qual Aslanpour, Gill e

Figura 2 – Conexões M2M por aplicação



Fonte: Extraído de Cisco Systems (2020)

Toosi (2020), Computação em Neblina (*Mist Computing*, tradução livre da autora). Ainda segundo Cao *et al.* (2020), unificam recursos para fornecer computação, armazenamento e rede próximos ao usuário, seja em distância geográfica ou de rede. As definições de névoa e borda são frequentemente confundidas ou agregadas, mas adota-se que a computação de borda é especializada, tende a executar uma aplicação específica em uma região específica com capacidade limitada de conexões e processamento em tempo real; ao passo que a computação em névoa é de propósito geral e possui arquitetura multicamadas, podendo atuar entre as camadas de borda e nuvem, permitindo maior distribuição de processamento e armazenamento. Já a neblina engloba os dispositivos finais, podendo ser interpretada como um paradigma composto por dispositivos de IoT (ASLANPOUR; GILL; TOOSI, 2020). Esses paradigmas demandam que mais parâmetros além da latência sejam avaliados, incluindo energia, confiabilidade, segurança e custo.

Sistemas operacionais Linux são abundantes em todas essas áreas, seja em dispositivos embarcados como Raspberry Pis ou em máquinas de *datacenters*. Suas ferramentas de monitoramento são capazes de fornecer informações pertinentes ao uso de recursos de hardware e software. Contudo, a própria presença dessas ferramentas já implica na introdução de sobrecusto computacional (*overhead*). Os novos paradigmas de computação possuem recursos mais limitados, e nem sempre a gestão e o armazenamento dos dados de monitoramento poderão ser mantidos localmente na íntegra. O artigo de Costa *et al.* (2022) identifica este ponto como caminho para futuras pesquisas, propondo a utilização de *frameworks* e técnicas de Internet das Coisas como Serviço (*Internet of Things as a Service*) (iTaaS), bem como a redução da dimensionalidade de contadores relevantes para o sistema, a exemplo do que propõe Pedro Freire Popiolek, Karina dos Santos Machado e Odorico Machado Mendizabal (2021).

No contexto apresentado, duas ferramentas de monitoramento amplamente utilizadas em ambientes Linux são o **Performance Co-Pilot (PCP)**, por meio do seu

componente de coleta **pmlogger**, e o Prometheus. O PCP é uma suíte de ferramentas de análise de desempenho desenvolvida originalmente pela Silicon Graphics e mantida atualmente como projeto de código aberto; o **pmlogger** é o componente responsável pela coleta e arquivamento contínuo de métricas de desempenho do sistema operacional. O Prometheus é um sistema de monitoramento e alerta de código aberto, originalmente desenvolvido pela SoundCloud e atualmente um projeto graduado da Cloud Native Computing Foundation (CNCF), amplamente adotado em ambientes de nuvem e contêineres. Apesar de terem propósitos semelhantes, as duas ferramentas possuem arquiteturas, modelos de coleta e impactos sobre o sistema distintos, o que motiva uma comparação do sobrecusto que cada uma impõe ao ambiente monitorado.

O presente trabalho realiza um benchmark comparativo entre o PCP (**pmlogger**) e o Prometheus em sistemas Linux executados em máquinas com recursos constritos, avaliando o sobrecusto computacional introduzido por cada ferramenta. Para isso, levanta e mapeia as métricas disponibilizadas pelas ferramentas, contextualiza-as nos paradigmas de computação distribuída discutidos e realiza experimentos de benchmark para mensurar o impacto de cada solução sobre CPU, memória e rede do sistema monitorado. A escolha por dispositivos com recursos constritos justifica-se por sua representatividade em relação aos sistemas embarcados e de borda, nos quais restrições de CPU, memória e energia são características intrínsecas e tornam o sobrecusto das ferramentas de monitoramento uma variável crítica para a viabilidade operacional do sistema.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo do presente trabalho é avaliar o cenário e possíveis soluções de monitoramento de sistemas Linux em dispositivos relevantes para os novos paradigmas computacionais, mais especificamente a Computação de Borda e Névoa, em dispositivos com recursos constritos, visando aproximar-se destes cenários.

1.1.2 Objetivos Específicos

- a) Sintetizar os principais paradigmas de computação distribuída que surgiram para complementar a computação em nuvem, contextualizando sua relevância e potencial para a área da Engenharia Eletrônica;
- b) Avaliar o panorama de métricas relevantes para os variados paradigmas computacionais, incluindo taxonomia;
- c) Avaliar requisitos de soluções de monitoramento para sistemas Linux relevantes para os paradigmas computacionais supracitados, mapeando ferramentas relevantes;

- d) Definir e executar uma metodologia de avaliação de performance de dispositivos com recursos constrictos para mensurar o sobrecusto introduzido por cada ferramenta;

1.2 TRABALHOS RELACIONADOS

Popiolek e Mendizabal (2012) apresentam uma abordagem de monitoramento e análise de desempenho em ambientes virtualizados. A partir de cargas de trabalho sintéticas, com número de máquinas virtuais ativas variando de uma a seis e perfis distintos de uso de CPU, E/S e memória, os autores identificam as principais causas da degradação de desempenho observada.

Em um trabalho subsequente, Pedro F. Popiolek, Karina S. Machado e Odorico M. Mendizabal (2018) analisam a influência do sobrecusto de monitoramento de desempenho em ambientes virtualizados e, como forma de reduzir a contenção por recursos compartilhados, propõem uma metodologia baseada no processo KDD para reduzir a dimensionalidade do espaço de características de desempenho, selecionando apenas os contadores mais relevantes.

Reichelt, Kühne e Hasselbring (2023) derivam quatro opções de otimização para reduzir o sobrecusto de monitoramento: instrumentação por código-fonte em vez de instrumentação via AspectJ, redução do volume de dados medidos, alteração da fila de processamento e agregação dos dados medidos. Avaliando essas opções com o *benchmark* MooBench sobre o *framework* Kieker, os autores demonstram a redução do sobrecusto de monitoramento.

Costa *et al.* (2022), em revisão sobre monitoramento de computação em névoa, discute trabalhos correlatos relevantes ao tema. Rosa Righi *et al.* (2019) combinam monitoramento de sistemas, gerência de recursos e predição de carga sob o conceito de visão de gerência global, propondo uma taxonomia independente para cada uma dessas três áreas e discutindo suas inter-relações, restrita, contudo, a propostas de nuvem e de ambientes locais. Abderrahim *et al.* (2017) descrevem propriedades que um sistema de monitoramento de névoa deve apresentar e categorizam arquiteturas de monitoramento segundo a topologia e a granularidade de decomposição funcional. Abreha *et al.* (2021) propõem uma taxonomia de soluções de monitoramento de névoa estruturada em três domínios: arquitetura, requisitos e parâmetros de projeto. Os autores analisaram soluções voltadas para a nuvem e concluíram que estas não são adequadas ao contexto de névoa.

1.3 USO DA INTELIGÊNCIA ARTIFICIAL NESTE TRABALHO

A Inteligência Artificial (IA), modelo Claude Sonnet 4.6 com esforço médio, foi utilizada em diferentes momentos da realização e escrita deste trabalho. Em ordem de realização das etapas, detalha-se a seguir as partes com e sem sua intervenção.

A revisão bibliográfica e os fichamentos foram realizados sem seu uso. Para a realização dos experimentos foram criados *scripts* em bash pela autora. Ao longo de adaptações e dificuldades operando as ferramentas escolhidas, a IA foi utilizada para conversão de bash para python, bem como em partes da configuração de métricas coletadas pelas soluções de monitoramento. Também foi assistente para o desenvolvimento dos gráficos apresentados neste trabalho, auxiliando a criar funções recursivas para as variadas execuções e *logs*, e com os elementos gráficos como legendas e organização de cores.

Para o processo de escrita a estrutura do trabalho foi definida pela autora sem o uso de IA. Para a escrita do Capítulo 2 e do Capítulo 3, após a pesquisa bibliográfica e subsequente fichamento, trechos julgados relevantes pela autora foram inseridos nos arquivos de texto dos capítulos, em conjunto com as fontes, compondo uma base para o texto. A seguir, foi solicitado a IA que reescrevesse o texto em português com uma linguagem formal e direta, preservando a ordem das informações apresentadas e sem inventar outras, organizando também as citações e a seção de referências; os artigos relevantes para os capítulos também foram fornecidos à IA na íntegra como contexto. Para os demais capítulos, a IA foi solicitada a organizar a formatação LaTeX e ABNT, ajustando sobretudo detalhes de figuras, tabelas e listas de itens. A conclusão, por fim, é parcialmente escrita pela autora e parcialmente pela IA.

Em suma, a concepção do trabalho, a revisão bibliográfica, a estrutura geral do texto e a definição dos objetivos foram realizadas exclusivamente pela autora. Os capítulos de Introdução e Avaliação Experimental foram escritos pela autora e formatados pela IA, já os demais capítulos do desenvolvimento foram estruturados pela autora e majoritariamente escritos pela IA com base em trechos dos artigos científicos relevantes.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os paradigmas de computação distribuída relevantes ao contexto do trabalho, incluindo computação em nuvem, névoa e borda, além de uma revisão sobre IoT e as métricas de desempenho associadas. O Capítulo 3 descreve as métricas relevantes para computação em névoa e as ferramentas de monitoramento estudadas, PCP e Prometheus, bem como o mapeamento das métricas disponibilizadas por cada uma. O Capítulo 4 apresenta a metodologia de avaliação adotada, os experimentos realizados e a análise dos resultados. Por fim, o Capítulo 5 traz as conclusões do trabalho e sugestões para pesquisas futuras.

2 PARADIGMAS COMPUTACIONAIS

Nas últimas duas décadas, o interesse em torno dos paradigmas de computação cresceu de forma expressiva. O paradigma mais notório é a Computação em Nuvem, que consolidou a computação como uma utilidade acessível por rede (DE DONNO; TANGE; DRAGONI, 2019). Contudo, a proliferação de dispositivos conectados, em especial no contexto de IoT, expôs as limitações da abordagem centralizada da nuvem, impulsionando o surgimento de paradigmas complementares, com destaque para a Computação em Borda e a Computação em Névoa. Também será citada neste trabalho a Computação em Neblina, mas que é menos abundante na literatura.

Figura 3 – Linha do tempo do surgimento dos Paradigmas Computacionais *Cloud*, IoT, Computação em Borda, Computação em Névoa e Computação em Neblina.

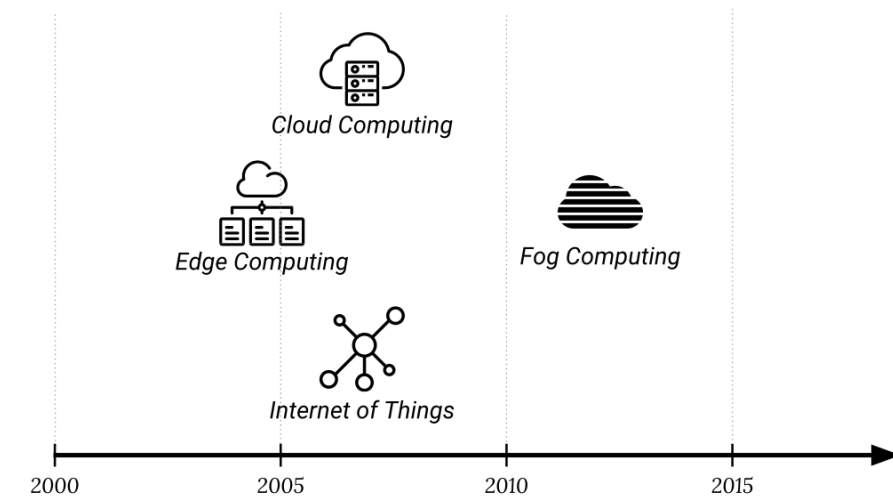


FIGURE 1. First appearance of IoT, Cloud, Edge, and Fog computing in the literature.

Fonte: Extraído de De Donno, Tange e Dragoni (2019)

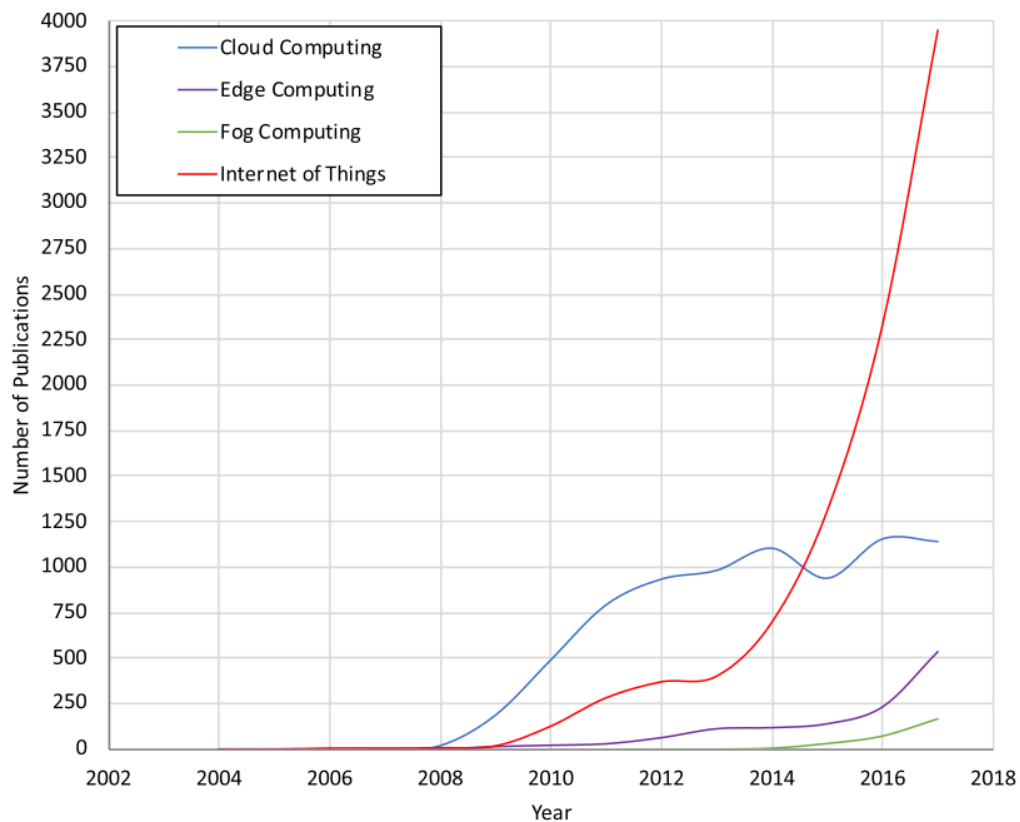
A evolução desses paradigmas não é independente: a IoT e a quantidade de dispositivos globalmente conectados cria pressão sobre a nuvem, que por sua vez não consegue atender a todos os seus requisitos; a computação em borda surge para aproximar o processamento dos dispositivos; e a névoa emerge como a camada integradora entre borda e nuvem, conforme ilustra a Figura 3. O interesse científico reflete essa dinâmica, pois enquanto publicações sobre nuvem estagnaram após 2013, os trabalhos sobre IoT, borda e névoa cresceram continuamente (DE DONNO; TANGE; DRAGONI, 2019), como detalhado na Figura 4.

O surgimento de tantos paradigmas propiciou o surgimento do conceito de Contínuo da Nuvem (Contínuo da Nuvem), que sintetiza essa diversidade de possibilidades computacionais. O interesse acadêmico surgiu em 2016 e permaneceu restrito nos anos seguintes; a partir de 2020 o tema ganhou tração e, em 2021, concentrou a maioria das

publicações sobre o assunto (MORESCHINI *et al.*, 2022), conforme evidencia a Figura 5. Há também literaturas que trabalham com a ideia de Contínuo Nuvem-Coisas (Contínuo Nuvem-Coisas), que engloba os mesmos conceitos, mas com foco ampliado não só na nuvem, também nos dispositivos finais (ASLANPOUR; GILL; TOOSI, 2020; COSTA *et al.*, 2022).

Este capítulo apresenta os fundamentos de cada um desses paradigmas, seus modelos arquiteturais e características, contextualizando-os no escopo do monitoramento de sistemas.

Figura 4 – Número de publicações de paradigmas computacionais ao longo do tempo.

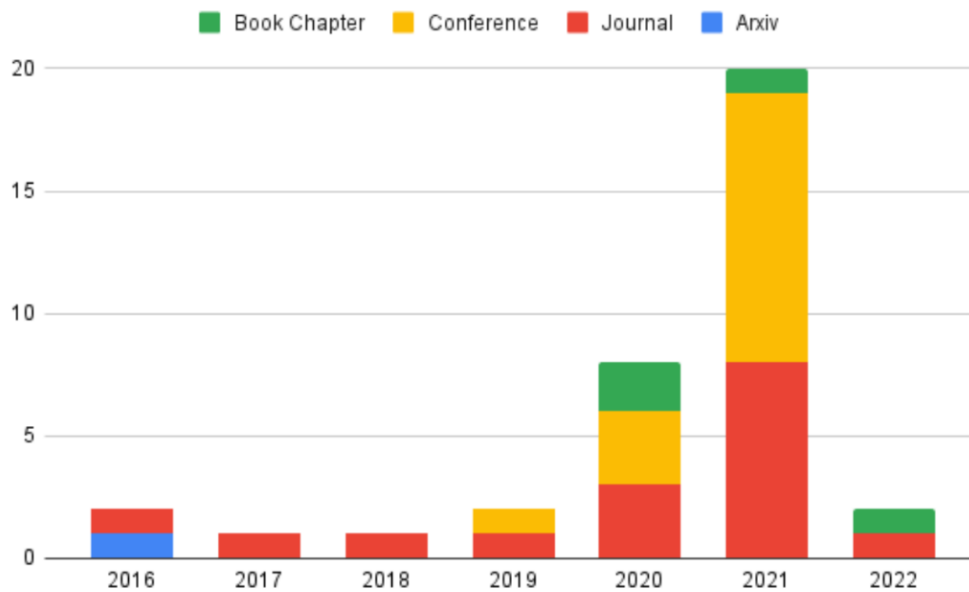


Fonte: Extraído de De Donno, Tange e Dragoni (2019)

2.1 ARQUITETURA DOS PARADIGMAS COMPUTACIONAIS

A relação entre os paradigmas computacionais modernos pode ser compreendida por meio do conceito de *contínuo nuvem-coisas* (*Cloud-to-Things continuum*): um espectro que vai desde os *data centers* centralizados da nuvem até os dispositivos finais da IoT, com múltiplas camadas intermediárias de processamento distribuído (ASLANPOUR; GILL; TOOSI, 2020; COSTA *et al.*, 2022).

Aslanpour, Gill e Toosi (2020) propõem um modelo de referência de quatro camadas para descrever esse contínuo em implantações reais: a camada de IoT, por vezes

Figura 5 – Número de publicações acerca de *Cloud Continuum* ao longo do tempo.

Fonte: Extraído de Moreschini *et al.* (2022)

denominada neblina (*mist*, tradução livre da autora), composta pelos próprios dispositivos finais com capacidade de processamento local; a camada de borda, formada por *gateways*, comutadores e micro *data centers* próximos aos dispositivos; a camada de névoa, com maior capacidade computacional que a borda, atuando como intermediária entre ela e a nuvem; e a camada de nuvem, responsável pelo armazenamento persistente e processamento de longa duração. Costa *et al.* (2022) propõe um modelo semelhante, conforme ilustrado na Figura 6, mas que considera definições mais amplas para cada paradigma, como o *Mobile Cloud Computing* (MCC), que será detalhado em páginas futuras. Cada paradigma ocupa uma posição distinta nesse contínuo, diferindo em capacidade computacional, latência, proximidade ao usuário e modelo de implantação, conforme sintetizado na Tabela 2.

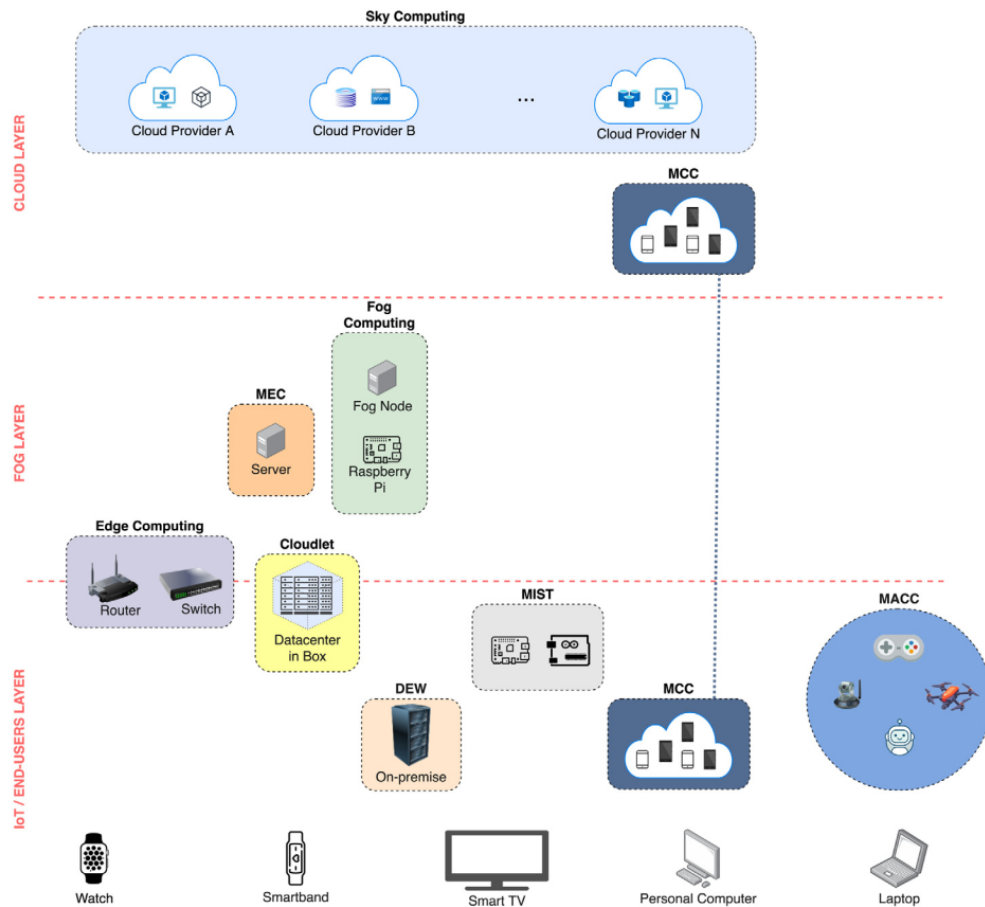
Tabela 2 – Comparativo entre os paradigmas de computação em nuvem, borda e névoa.

Parâmetro	Nuvem	Borda	Névoa
Segurança	Moderada	Alta	Alta
Latência	Alta	Baixa	Moderada
Migração de carga	Alta	Moderada	Moderada
Flexibilidade	Alta	Alta	Alta
Escalabilidade	Alta	Alta	Alta
Largura de banda	Moderada	Moderada	Moderada

Fonte: adaptado de Sharma e Prabha (2021).

O OpenFog Consortium propôs a arquitetura *N-tier*, que refina a camada de névoa em múltiplos níveis hierárquicos (DE DONNO; TANGE; DRAGONI, 2019). Os nós mais

Figura 6 – Arquitetura de Paradigmas de Computação.



Fonte: Extraído de Costa *et al.* (2022)

próximos aos dispositivos coletam e normalizam dados brutos; os níveis intermediários filtram, comprimem e transformam os dados; e os nós mais próximos à nuvem agregam dados e constroem conhecimento. Nós de um mesmo nível podem comunicar-se horizontalmente, formando malhas para prover resiliência, tolerância a falhas e balanceamento de carga. Nesse modelo, a computação em borda corresponde à camada mais próxima dos dispositivos finais, enquanto a névoa ocupa os níveis intermediários, integrando IoT, borda e nuvem em uma arquitetura unificada (DE DONNO; TANGE; DRAGONI, 2019; COSTA *et al.*, 2022).

Aslanpour, Gill e Toosi (2020) descrevem o contínuo nuvem-coisas como uma arquitetura de quatro camadas interdependentes:

- Camada IoT/neblina:* formada pelos próprios dispositivos IoT, tais quais sensores, atuadores, *smartphones*, veículos conectados e demais dispositivos de borda extrema, amplamente distribuídos geograficamente e responsáveis pela coleta de dados do ambiente. Quando dotados de capacidade de processamento local, esses dispositivos podem cooperar entre si como uma rede malha (*mesh*), gerenciada de forma centralizada ou distribuída, e operar de forma completamente

autônoma sem comunicação com camadas superiores;

- b) *Camada de borda*: composta por *gateways* dedicados, roteadores, comutadores e estações base posicionados próximos aos dispositivos IoT. Pode incluir *micro data centers* e *cloudlets*, que aproximam ainda mais o processamento dos dispositivos, reduzem o tráfego transmitido às camadas superiores e permitem o envio apenas dos dados analisados para a névoa ou nuvem;
- c) *Camada de névoa*: formada por nós (*fog nodes*) capazes de computar, transmitir e armazenar dados temporariamente, com maior capacidade computacional que a camada de borda. Segundo o OpenFog Consortium, citado por De Donno, Tange e Dragoni (2019), um nó de névoa é “o elemento de rede físico e lógico que implementa serviços de computação em névoa”. Os nós podem ser posicionados em qualquer ponto entre a nuvem e os dispositivos finais, conectados tanto aos dispositivos IoT quanto à infraestrutura de nuvem; e
- d) *Camada de nuvem*: o *backbone* da infraestrutura, responsável pelo armazenamento persistente e por recursos computacionais ilimitados não disponíveis nas camadas inferiores, como processamento de longa duração e análise aprofundada de dados. Abrange os modelos privado, público, híbrido, de provedor único, multiprovedor e federado (ASLANPOUR; GILL; TOOSI, 2020).

Embora existam na literatura arquiteturas propostas para cada paradigma individualmente, como modelos voltados exclusivamente à computação em borda (CAO *et al.*, 2020) ou à nuvem (SYED *et al.*, 2017), o escopo desta seção concentra-se nas arquiteturas de referência que posicionam conjuntamente névoa, borda e nuvem no contínuo computacional, por serem as mais relevantes para o contexto que motiva este trabalho.

2.2 COMPUTAÇÃO EM NUVEM

O termo *Cloud Computing* foi proposto pela primeira vez em agosto de 2006 pelo então CEO da Google, Eric Schmidt, na conferência *Search Engine Strategies* (SES), ao passo que a Amazon o referenciou como produto comercial no mesmo período (DE DONNO; TANGE; DRAGONI, 2019; CAO *et al.*, 2020). Trabalhos científicos sobre o tema começaram a surgir em 2008 e o interesse cresceu continuamente até 2013 (DE DONNO; TANGE; DRAGONI, 2019). O Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST) define computação em nuvem como um modelo para possibilitar acesso de rede ubíquo, conveniente e sob demanda a um conjunto compartilhado de recursos computacionais configuráveis (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com esforço mínimo de gerenciamento ou interação com o provedor de serviços (DE DONNO; TANGE; DRAGONI, 2019).

O modelo de computação em nuvem é estruturado em torno de cinco características essenciais definidas pelo NIST (DE DONNO; TANGE; DRAGONI, 2019):

- a) *Serviço sob demanda (on-demand self-service)*: capacidades computacionais provisionadas automaticamente sem necessidade de interação humana com o provedor;
- b) *Ampla acesso à rede (broad network access)*: disponibilidade por meio de mecanismos padronizados sobre a rede, acessíveis de diferentes plataformas cliente;
- c) *Agrupamento de recursos (resource pooling)*: recursos computacionais do provedor agrupados para atender múltiplos consumidores com alocação dinâmica conforme demanda, sendo a localização física dos recursos transparente ao consumidor;
- d) *Elasticidade rápida (rapid elasticity)*: capacidades provisionadas e liberadas de forma elástica e, em alguns casos, automaticamente, dando ao consumidor a percepção de recursos ilimitados; e
- e) *Serviço mensurado (measured service)*: uso de recursos monitorado, controlado e reportado de forma transparente, provendo visibilidade tanto ao provedor quanto ao consumidor.

Do ponto de vista dos modelos de serviço, a computação em nuvem oferece três camadas (DE DONNO; TANGE; DRAGONI, 2019; SYED *et al.*, 2017): *Infraestrutura como Serviço (Infrastructure as a Service, IaaS)*, que disponibiliza recursos de hardware virtualizados como servidores, redes e armazenamento; *Plataforma como Serviço (Platform as a Service, PaaS)*, que provê ambientes de desenvolvimento e execução de aplicações; e *Software como Serviço (Software as a Service, SaaS)*, que entrega aplicações prontas ao usuário final. Esses modelos permitem que empresas concentrem seus esforços em seus modelos de negócio, sem necessidade de manter infraestrutura própria (SYED *et al.*, 2017).

Quanto aos modelos de implantação, a nuvem pode ser privada, pública, híbrida, de provedor único, multiprovedor ou federada (ASLANPOUR; GILL; TOOSI, 2020). O custo varia conforme o modelo adotado: pagamento por uso, reserva de crédito antecipado ou reserva de recursos específicos (SYED *et al.*, 2017).

2.2.1 Limitações

Apesar de sua consolidação como paradigma dominante, a computação em nuvem apresenta limitações relevantes para aplicações de IoT e tempo real (DE DONNO; TANGE; DRAGONI, 2019; CAO *et al.*, 2020). Sua arquitetura centralizada é inadequada para requisitos de baixa latência e alta largura de banda, além de apresentar desafios relacionados à conectividade intermitente, distribuição geográfica de dispositivos, consciência de contexto e segurança e privacidade de dados. Essas limitações motivaram o desenvolvimento dos paradigmas apresentados nas seções seguintes.

2.3 COMPUTAÇÃO EM BORDA

A computação em borda emergiu como paradigma na literatura por volta de 2004 e 2005, anteriormente ao crescimento das publicações sobre computação em nuvem (DE DONNO; TANGE; DRAGONI, 2019). Este interesse aumentou a partir da disseminação dos dispositivos IoT, que criam pressão sobre a infraestrutura de rede e os *data centers* centralizados.

Segundo De Donno, Tange e Dragoni (2019) e Aslanpour, Gill e Toosi (2020), a computação em borda pode ser definida como o conjunto de tecnologias que possibilitam a execução de processamento na borda da rede sobre dados descendentes (*downstream*) em nome de serviços de nuvem, e sobre dados ascendentes (*upstream*) em nome de serviços IoT. A ideia central é estender a computação em nuvem à borda da rede, aproximando o processamento das fontes de dados, isto é, dos próprios dispositivos IoT. De forma semelhante, Moreschini *et al.* (2022) e Cao *et al.* (2020) descrevem a borda como um novo paradigma que realiza o processamento próximo à fonte dos dados, com foco em serviços de baixa latência, análise em tempo real e inteligência local.

A Tabela 3 resume as principais diferenças entre os dois paradigmas.

Tabela 3 – Principais diferenças entre computação em nuvem e em borda.

Paradigma	Escopo	Pressão de banda	Tempo real	Modo de processamento
Computação em Nuvem	Global	Alta	Baixo	Centralizado, larga escala
Computação em Borda	Local	Baixa	Alto	Inteligência local, pequena escala

Fonte: adaptado de Cao *et al.* (2020).

As vantagens da computação em borda em relação à nuvem incluem (CAO *et al.*, 2020): (i) *processamento rápido e em tempo real*, pela redução da latência de transmissão; (ii) *segurança aprimorada*, pois dados sensíveis são processados localmente sem necessidade de envio à nuvem; (iii) *redução de custo e consumo de energia*, dado que menos dados precisam ser transmitidos; e (iv) *baixo consumo de largura de banda*, pela filtragem e processamento no local de origem dos dados.

O princípio da computação em borda pode ser implementado de diferentes maneiras (DE DONNO; TANGE; DRAGONI, 2019; LIU *et al.*, 2019). As principais são descritas a seguir.

Mobile Cloud Computing (MCC): baseada na ideia de offloading móvel, delega armazenamento e processamento a entidades externas ao dispositivo móvel, visando reduzir a carga de trabalho local e otimizar consumo de energia. Com a evolução do paradigma de borda, o destino do offloading passou a ser dispositivos na borda da rede, e não mais exclusivamente a nuvem (DE DONNO; TANGE; DRAGONI, 2019).

Cloudlet Computing (CC): proposta pela Universidade Carnegie Mellon (CMU) em 2009, um *cloudlet* é um computador ou *cluster* confiável e rico em recursos, localizado na borda da rede e disponível para dispositivos móveis nas proximidades. Apresenta três características fundamentais (LIU *et al.*, 2019): *Soft State*, ou seja, mantém estado temporário sem persistência de longo prazo; *Rich Resources*, com capacidade computacional suficiente para múltiplos usuários simultâneos; e *Close to Users*, com proximidade física e de rede que reduz latência e *jitter*. O suporte a mobilidade ocorre por meio de três mecanismos: descoberta de *cloudlets* disponíveis, provisionamento de máquinas virtuais e *handoff* entre *cloudlets*.

Mobile Edge Computing (MEC): implementação que posiciona capacidades de nuvem na borda da rede móvel, dentro da Rede de Acesso por Rádio (RAN). Os nós MEC são geralmente localizados junto ao controlador de rede móvel ou a estações rádio base, oferecendo benefícios de consciência de localização/contexto, baixa latência e alta largura de banda (DE DONNO; TANGE; DRAGONI, 2019).

Embora compartilhem o objetivo de aproximar o processamento dos dispositivos finais, essas implementações diferem quanto ao tipo de dispositivos envolvidos, protocolos de comunicação e entidades elegíveis para disponibilizá-las (DE DONNO; TANGE; DRAGONI, 2019).

Liu *et al.* (2019) classificam os sistemas de computação em borda em três categorias segundo a origem do fluxo de dados: *Push From Cloud*, em que provedores de nuvem estendem serviços à borda; *Pull From IoT*, em que aplicações IoT buscam processamento na borda mais próxima; e *Hybrid Cloud-Edge Analytics*, que integra nuvem e borda para análise colaborativa.

2.4 COMPUTAÇÃO EM NÉVOA

A computação em névoa tem origem documentada em 2012, quando a Cisco a definiu pela primeira vez como “uma plataforma altamente virtualizada que fornece serviços de computação, armazenamento e rede entre dispositivos finais e *data centers* de computação em nuvem tradicionais, tipicamente, mas não exclusivamente, localizada na borda da rede” (DE DONNO; TANGE; DRAGONI, 2019). O termo foi cunhado por Flavio Bonomi, da Cisco, em referência à névoa como uma nuvem mais próxima ao chão.

Embora frequentemente tratada como uma implementação da computação em borda, De Donno, Tange e Dragoni (2019) argumentam que a névoa representa a *evolução mais elevada dos princípios da borda* (*the highest evolution of the Edge computing principles*). A névoa não se limita à borda da rede: seus nós podem ser posicionados em qualquer ponto entre os dispositivos finais e a nuvem. Além disso, a névoa não atende apenas ao lado das “coisas”, mas também provê serviços à nuvem, atuando como um novo ator que integra plenamente IoT, borda e nuvem. Neste trabalho, adota-se a definição do OpenFog Consortium, citada por De Donno, Tange e Dragoni (2019):

Uma arquitetura horizontal em nível de sistema que distribui recursos e serviços de computação, armazenamento, controle e rede em qualquer ponto do contínuo entre a Nuvem e as Coisas, acelerando a velocidade de tomada de decisão (DE DONNO; TANGE; DRAGONI, 2019).

Costa *et al.* (2022) distinguem a névoa da borda pelas suas características arquiteturais: enquanto a computação em borda tende a ser especializada, executando uma aplicação específica em uma região com capacidade limitada de conexões e processamento em tempo real, a computação em névoa é de propósito geral, possui arquitetura multicamadas e pode atuar entre as camadas de borda e nuvem, permitindo maior distribuição de processamento e armazenamento. A névoa lida ainda com requisitos de baixa latência, implantações geo-distribuídas, execução em plataformas heterogêneas, interações em tempo real e heterogeneidade de nós (COSTA *et al.*, 2022).

Os principais benefícios da computação em névoa são sintetizados pelo acrônimo SCALE (DE DONNO; TANGE; DRAGONI, 2019): *Security* (Segurança), tratada como alicerce arquitetural e não como adendo, com a névoa atuando como paradigma intrinsecamente seguro ao longo do contínuo nuvem-coisas; *Cognition* (Cognição), com distribuição mais refinada de computação e controle, construindo aplicações que melhor atendem às necessidades dos clientes; *Agility* (Agilidade), pela inovação rápida e escalabilidade acessível por meio de ferramentas abertas e proliferação de dispositivos IoT; *Latency* (Latência), atendendo requisitos de tempo real por meio do processamento próximo ao usuário; e *Efficiency* (Eficiência), pelo agrupamento de capacidades de computação, comunicação, controle e armazenamento entre a nuvem e os dispositivos IoT.

2.5 INTERNET DAS COISAS

O conceito de Internet das Coisas foi criado em 1999 por Kevin Ashton, então diretor executivo do Auto-ID Center do Instituto de Tecnologia de Massachusetts (MIT). Os primeiros trabalhos científicos sobre o tema datam de 2006, e desde então o termo assumiu distintos significados na literatura (DE DONNO; TANGE; DRAGONI, 2019).

Neste trabalho, adota-se a definição estabelecida pela União Internacional de Telecomunicações (ITU): IoT é “uma infraestrutura global para a sociedade da informação, possibilitando serviços avançados por meio da interconexão de coisas físicas e virtuais, com base em tecnologias de informação e comunicação interoperáveis existentes e em evolução” (INTERNATIONAL TELECOMMUNICATION UNION, 2012). Nesse contexto, uma *coisa* é um objeto do mundo físico ou virtual capaz de ser identificado e integrado a redes de comunicação; um *dispositivo* é um equipamento com capacidade obrigatória de comunicação e capacidades opcionais de sensoriamento, atuação, captura, armazenamento e processamento de dados (INTERNATIONAL TELECOMMUNICATION UNION, 2012; DE DONNO; TANGE; DRAGONI, 2019).

Em síntese, a IoT pode ser descrita como uma coleção de dispositivos computacionais interconectados via Internet, voltada a oferecer serviços para todos os tipos de aplicações com cumprimento de requisitos de segurança (DE DONNO; TANGE; DRAGONI, 2019).

O modelo arquitetural de IoT mais amplamente adotado na literatura é composto por três camadas (DE DONNO; TANGE; DRAGONI, 2019; OLIVEIRA *et al.*, 2024):

- a) *Camada de percepção (Perception ou Sensing layer)*: responsável pela aquisição de dados do ambiente físico por meio de sensores e atuadores; é a camada que detecta e coleta informações antes de transmiti-las à camada superior;
- b) *Camada de rede (Network ou Transmission layer)*: encarregada do roteamento e transmissão eficiente de dados entre os dispositivos e os destinos adequados, operando por meio de *gateways*, roteadores e outros dispositivos de rede; e
- c) *Camada de aplicação (Application layer)*: a camada superior, que processa as informações recebidas para implementar os serviços e aplicações específicos do domínio, contendo a lógica de negócio e as interfaces com o usuário.

As características essenciais da IoT são sintetizadas por De Donno, Tange e Dragoni (2019) como: *interconectividade*, pela integração à infraestrutura global de comunicação; *serviços relacionados às coisas (things-related services)*, incluindo proteção de privacidade e consistência semântica entre mundos físico e virtual; *heterogeneidade*, pois os dispositivos podem operar em diferentes redes e plataformas de hardware; *recursos restritos (constrained resources)*, dado que a maioria é caracterizada por restrições energéticas e computacionais; *dinamismo*, uma vez que o estado e o contexto dos dispositivos mudam continuamente; e *escala massiva (huge scale)*, dada a quantidade crescente de dispositivos que precisam se comunicar entre si.

O crescimento do ecossistema IoT é expressivo. Segundo o Cisco Annual Internet Report, em 2023 havia previsão de 29,3 bilhões de dispositivos conectados à Internet, um crescimento de 59% em relação aos 18,4 bilhões de 2018, totalizando 3,6 dispositivos per capita (CISCO SYSTEMS, 2020). As conexões M2M representariam 50% do total, sendo que aplicações em residências inteligentes equivalem a 48% das conexões M2M. Veículos conectados representam a categoria de maior Taxa de Crescimento Anual Composta, de 30% (CISCO SYSTEMS, 2020).

2.5.1 Computação em Neblina

A computação em neblina representa a camada mais extrema do contínuo nuvem-coisas, formada pelos próprios dispositivos IoT quando dotados de capacidade de processamento local colaborativo (ASLANPOUR; GILL; TOOSI, 2020). Enquanto a névoa atua como intermediária entre borda e nuvem, a neblina engloba os dispositivos finais capazes de processar dados localmente e cooperar entre si como uma malha (*mesh*), antes mesmo

de qualquer comunicação com camadas superiores. Dispositivos como o Raspberry Pi são exemplos típicos de nós de neblina (ASLANPOUR; GILL; TOOSI, 2020).

Nesse paradigma, os dispositivos IoT são computacionalmente habilitados para fornecer capacidades além do simples sensoramento e atuação, podendo constituir redes *mesh* gerenciadas de forma centralizada ou distribuída (ASLANPOUR; GILL; TOOSI, 2020). Uma das características mais relevantes da computação em neblina é a capacidade de operar de forma completamente autônoma, sem comunicação com névoa, borda ou nuvem, o que o torna especialmente adequado para cenários de conectividade intermitente ou aplicações de missão crítica com requisitos de latência ultrabaixa.

2.5.2 Desafios que Motivam Novos Paradigmas

A interação entre IoT e a computação em nuvem introduz desafios que a nuvem, por si só, não consegue endereçar (DE DONNO; TANGE; DRAGONI, 2019):

- a) *Baixa latência*: aplicações de controle industrial, saúde e veículos autônomos frequentemente exigem respostas na ordem de milissegundos, requisito fora do alcance da nuvem centralizada;
- b) *Alta largura de banda*: o volume crescente de dados gerado pelos dispositivos, se transmitido integralmente à nuvem, resultaria em congestionamento de rede e elevados custos de transmissão;
- c) *Recursos limitados*: muitos dispositivos IoT não suportam os protocolos complexos necessários para interagir diretamente com a nuvem;
- d) *Conectividade intermitente*: veículos, drones e sensores de campo frequentemente operam com conectividade irregular;
- e) *Distribuição geográfica*: a vastidão geográfica das implantações IoT inviabiliza a centralização dos serviços em poucos *data centers*;
- f) *Consciência de contexto (context awareness)*: muitas aplicações requerem informações locais, como localização do usuário e condições da rede, que não são facilmente obtidas a partir da nuvem; e
- g) *Segurança e privacidade*: a transmissão de dados sensíveis (como dados de saúde ou de vigilância doméstica) para *data centers* remotos introduz riscos de privacidade e exposição a ataques.

Esses desafios motivaram o desenvolvimento dos paradigmas de computação em borda e névoa como complementos à nuvem (DE DONNO; TANGE; DRAGONI, 2019).

2.6 CONTINUIDADE DA NUVEM (CLOUD CONTINUUM)

O conceito de *Cloud Continuum* sintetiza essa arquitetura como uma extensão da nuvem tradicional em direção a múltiplas entidades (como borda, névoa e dispositivos

IoT) que fornecem capacidades de análise, processamento, armazenamento e geração de dados (MORESCHINI *et al.*, 2022). Para chegar a essa definição unificada, Moreschini *et al.* (2022) conduzem um mapeamento sistemático de 36 definições publicadas entre 2016 e 2022, analisando como o conceito evoluiu e onde a nuvem se estende.

As duas primeiras definições do *Cloud Continuum* foram propostas simultaneamente em 2016, mas partem de perspectivas distintas. A primeira, de natureza centrada nos recursos, descreve o contínuo como um espectro de recursos disponíveis desde a borda da rede até o *datacenter* em nuvem. A segunda, de natureza centrada na computação, enfatiza a dimensão de *onde* e *como* o processamento é executado ao longo desse espectro (MORESCHINI *et al.*, 2022). Essa dualidade de origens deu forma aos dois grandes grupos de definições que dominam a literatura subsequente: o primeiro considera o contínuo como a distribuição de recursos em diferentes elementos da rede, incluindo IoT, névoa e borda; o segundo o interpreta como a extensão do poder de processamento a diferentes nós, frequentemente mencionando a possibilidade de execução de inteligência artificial. Adicionalmente, dez estudos adotaram o termo *Computing Continuum* para enfatizar a capacidade computacional em detrimento da simples interconexão de entidades, definindo-o, por exemplo, como “uma infraestrutura digital utilizada conjuntamente por fluxos de trabalho de aplicações complexas” (MORESCHINI *et al.*, 2022).

A partir de 2020, o número de definições cresceu acentuadamente, e em 2021 o conceito consolidou-se como a combinação ou agregação de dispositivos e serviços de névoa, borda e IoT, ou como a extensão da nuvem em direção a esses elementos. Diante dessa diversidade, Moreschini *et al.* (2022) propõem uma definição unificada que incorpora explicitamente os dois aspectos recorrentes: o contínuo como extensão dos recursos e como extensão das capacidades computacionais. A arquitetura resultante organiza-se em três camadas complementares: *escalabilidade da nuvem*, com recursos virtualmente ilimitados e amplas possibilidades de integração; *flexibilidade da névoa*, que permite a realocação dinâmica de funcionalidades; e *dependabilidade da borda*, caracterizada por comunicação de baixa latência, determinismo e preservação local de dados (MORESCHINI *et al.*, 2022). Nesse enquadramento, os dispositivos Linux com recursos restritos estudados neste trabalho situam-se na camada de borda do contínuo, onde os requisitos de dependabilidade tornam o sobrecusto das ferramentas de monitoramento uma variável crítica.

2.7 CONEXÕES COM A ENGENHARIA ELETRÔNICA

2.7.1 Sistemas Embarcados e IoT Inteligente

Os dispositivos que compõem as camadas de *neblina* e borda são, em sua maioria, sistemas embarcados: plataformas de hardware com capacidades computacionais e de comunicação integradas, frequentemente sujeitas a severas restrições de energia, memória e processamento. A evolução desses sistemas ao longo das décadas, dos primeiros microcon-

troladores dos anos 1970 aos modernos *single-board computers* (SBCs) com processadores de 64 bits e suporte a aceleração de *deep learning*, criou as condições para o surgimento de novos paradigmas de aplicação (OLIVEIRA *et al.*, 2024).

Oliveira *et al.* (2024) propõem o conceito de Internet das Coisas Inteligentes (IoIT) como a convergência de sistemas embarcados, computação em borda e aprendizado de máquina. Nesse paradigma, dispositivos embarcados são equipados com modelos de inteligência artificial comprimidos por meio de uma abordagem denominada *TinyML*, que permite a inferência local em dispositivos com severas restrições de recursos. Isso reduz a dependência de conectividade com a nuvem, viabilizando aplicações em cidades inteligentes, saúde, agricultura de precisão e automação industrial, entre outras.

Sistemas embarcados executando Linux, como os baseados em Raspberry Pi, são o foco das ferramentas de monitoramento estudadas neste trabalho. Nesses ambientes, a capacidade de monitorar eficientemente o uso de CPU, memória, rede e outros recursos é essencial para a orquestração e manutenção dos serviços providos.

2.7.2 Redes 6G

Cada geração de tecnologia de comunicação móvel foi projetada para atender às necessidades de usuários finais e operadores de rede (GIORDANI *et al.*, 2020). No entanto, as sociedades contemporâneas tornam-se cada vez mais orientadas a dados, dependentes de dados e automatizadas. A automação radical dos processos de manufatura industrial impulsionará a produtividade de forma sem precedentes (GIORDANI *et al.*, 2020).

Sistemas autônomos já estão presentes em rodovias, oceanos e espaço aéreo. Milhões de sensores serão incorporados a cidades, residências e ambientes de produção, e novos sistemas operados por inteligência artificial residindo em ambientes locais de nuvem e névoa viabilizarão uma ampla gama de aplicações inéditas (GIORDANI *et al.*, 2020).

As redes de comunicação constituirão o sistema nervoso desses novos paradigmas inteligentes. As demandas, no entanto, serão elevadas: as redes precisarão transmitir volumes muito maiores de dados a velocidades muito superiores. Ampliando uma tendência já iniciada no 4G e 5G, as conexões de sexta geração (6G) irão além da comunicação personalizada em direção à plena realização do paradigma da Internet das Coisas, interconectando não apenas pessoas, mas também recursos computacionais, veículos, dispositivos, equipamentos vestíveis (*wearables*), sensores e até agentes robóticos (GIORDANI *et al.*, 2020).

O rápido desenvolvimento de processos centrados em dados e automatizados, que exigem taxas de transmissão na ordem de terabits por segundo, latências de centenas de microssegundos e 10^7 conexões por km^2 , pode superar até mesmo as capacidades dos sistemas 5G emergentes (GIORDANI *et al.*, 2020). Além disso, a proliferação de serviços de saúde digital (*eHealth*) desafiará a capacidade de atender a seus rígidos requisitos de qualidade de serviço (QoS): disponibilidade de conexão contínua (confiabilidade de

99,99999%), latência submilissegundo e suporte à mobilidade. O tráfego móvel deverá superar 125 bilhões de dispositivos conectados até 2030, com densidades de 10^7 dispositivos por km^2 em áreas densas, superior aos 10^6 do 5G (GIORDANI *et al.*, 2020).

O 6G conectará dispositivos pessoais, sensores para o paradigma de cidades inteligentes e veículos autônomos. A conexão de veículos autônomos exige níveis inéditos de confiabilidade e baixa latência, acima de 99,99999% e abaixo de 1 ms, respectivamente, mesmo em cenários de alta mobilidade, como velocidades de até 1.000 km/h (GIORDANI *et al.*, 2020). Além disso, veículos aéreos não tripulados, como drones, representam enorme potencial para construção civil, primeiros socorristas e expansão da conectividade. Arquiteturas 6G heterogêneas futuras proverão cobertura tridimensional, complementando infraestruturas terrestres com plataformas não terrestres como drones, balões e satélites (GIORDANI *et al.*, 2020).

No plano de rede, o 6G levará a desagregação ao extremo ao virtualizar componentes de controle de acesso ao meio (MAC) e de camada física (PHY), que atualmente exigem implementações em hardware dedicado, realizando plataformas distribuídas de baixo custo compostas essencialmente por antenas e processamento mínimo (GIORDANI *et al.*, 2020).

O volume massivo de dados gerado pelos dispositivos futuros sobrecarregará as tecnologias de comunicação, que poderão não garantir a qualidade de serviço requerida. Nesse contexto, estratégias de aprendizado de máquina (ML) são fundamentais para avaliar o grau de correlação entre observações, extrair características de vetores de entrada e operar a rede de forma autônoma por meio de abordagens não supervisionadas e de aprendizado por reforço (GIORDANI *et al.*, 2020). As compensações entre latência, consumo de energia, sobrecusto de sistema e custo serão estudadas tanto para soluções embarcadas quanto para soluções assistidas por borda e nuvem. Redes orientadas por aprendizado ainda estão em seus estágios iniciais, mas constituirão um componente fundamental dos complexos sistemas 6G, que preconizam inteligência artificial distribuída para implementar uma arquitetura de rede verdadeiramente centrada no usuário (GIORDANI *et al.*, 2020).

3 MÉTRICAS E SOLUÇÕES DE MONITORAMENTO PARA SISTEMAS LINUX NO CONTEXTO FOG

Conforme explicitado anteriormente, a computação em nuvem é comercializada de acordo com três modelos de serviço distintos: *Infraestrutura como Serviço* (IaaS), *Plataforma como Serviço* (PaaS), e *Software como Serviço* (SaaS), e deve cumprir com Acordos de Nível de Serviço (SLAs). Eles são típicos do modelo de negócios dos serviços de nuvem, são assinados no momento da contratação e descrevem métricas como tempo de atividade, tempo de entrega, tempo de resposta e tempo de resolução, e à partir delas estabelecem critérios mínimos de fornecimento dos serviços. Esta garantia está diretamente ligada à orquestração de serviços em ambientes de computação distribuída. Costa *et al.* (2022) definem orquestração como o conjunto de funcionalidades que coordena a infraestrutura dinâmica e gerencia os serviços para garantir os Acordos de Nível de Serviço (SLAs).

Em ambientes de névoa e borda, a orquestração enfrenta desafios adicionais em relação à nuvem: os recursos são inerentemente voláteis, os nós são heterogêneos e geograficamente distribuídos, as cargas de trabalho são dinâmicas e os dispositivos frequentemente operam com recursos constritos (COSTA *et al.*, 2022). A orquestração abrange funcionalidades de controle de admissão, gerenciamento de serviços, gerenciamento de recursos, otimização, gerenciamento de comunicação e segurança, todas dependentes de informações coletadas pelo monitoramento para que possam agir em tempo hábil (COSTA *et al.*, 2022).

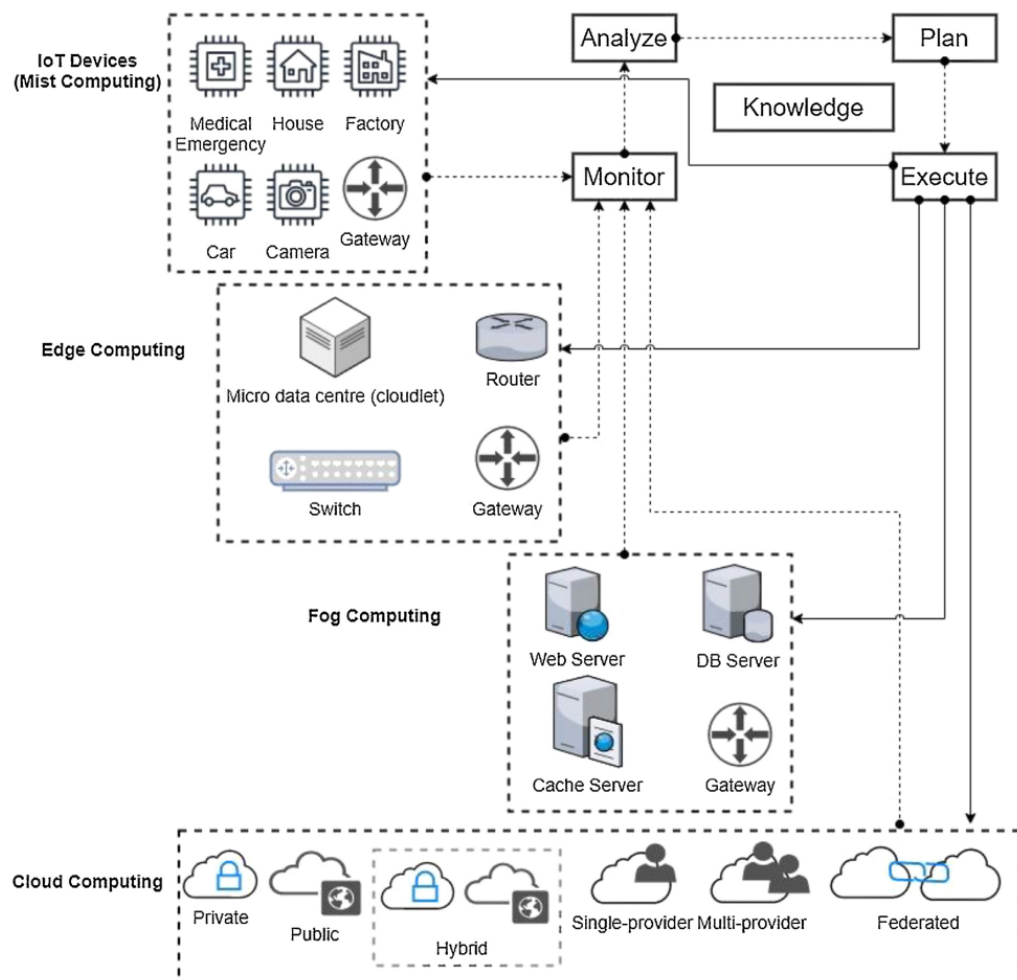
Nesse contexto, o monitoramento é a funcionalidade de maior importância para a orquestração: é por meio dele que o estado atualizado dos recursos e serviços é coletado e enviado ao orquestrador, habilitando as ações de gerenciamento e execução que compõem o ciclo de vida do serviço (COSTA *et al.*, 2022). Costa *et al.* (2022) distinguem dois aspectos complementares do monitoramento: verificar *se* um nó ou serviço está operando corretamente (*black-box monitoring*), e explicar *por que* ele parou de funcionar adequadamente (*white-box monitoring*). O monitoramento opera por meio de três domínios de instrumentação: *métricas* (medidas numéricas pontuais, como utilização de CPU e tempo de resposta), *logs* (coleções de strings não estruturadas que fornecem contexto adicional sobre eventos) e *rastreamentos* (*traces*, registros do fluxo completo de uma requisição ao longo dos serviços). A literatura define ainda os *golden signals* do monitoramento como o conjunto mínimo de telemetria suficiente para caracterizar o estado de um serviço em produção: latência, tráfego, erros e saturação (COSTA *et al.*, 2022).

A presença da Internet das Coisas exige uma consideração ainda mais cuidadosa da orquestração devido à sensibilidade à latência de muitas aplicações (ou serviços) de IoT e às limitações específicas da IoT. Neste sentido, a computação de borda e a computação em névoa aproximam os recursos computacionais dos usuários e dispositivos de IoT (ASLANPOUR; GILL; TOOSI, 2020).

3.1 O CICLO MAPE-K

Aslanpour, Gill e Toosi (2020) e Costa *et al.* (2022) organizam o processo de orquestração com base no ciclo *Monitor, Analyze, Plan, Execute, Knowledge* (MAPE-K), proposto pela IBM como modelo de orquestração autônoma. Nesse ciclo: o componente *Monitor* observa o estado do sistema, desde o nível de aplicação (requisições dos usuários, violações de SLA) até o nível de hardware; o *Analyzer* enriquece os dados coletados, eventualmente produzindo previsões; o *Planner* decide sobre a orquestração em si, adapta e aloca recursos; o *Executor* implementa as ações decididas; e o repositório de *Knowledge* mantém todo o histórico de informações monitoradas para subsidiar decisões futuras. A Figura 7 ilustra o ciclo MAPE-K.

Figura 7 – Ciclo MAPE-K de monitoramento para orquestração e dispositivos.



Fonte: Extraído de Costa *et al.* (2022)

3.2 SOBRE MÉTRICAS

A avaliação de desempenho em ambientes de computação distribuída requer a seleção cuidadosa de métricas adequadas a cada camada da infraestrutura (ASLANPOUR; GILL; TOOSI, 2020). Conforme argumentam Aslanpour, Gill e Toosi (2020), a seleção incorreta de métricas pode induzir avaliações equivocadas: por exemplo, um escalonador de recursos ciente de latência deve ser avaliado por métricas de tempo de resposta, e não apenas por custo. Além disso, os objetivos de avaliação tornaram-se mais diversos com o avanço dos paradigmas de névoa e borda, exigindo métricas além da latência, como energia, confiabilidade, segurança e custo (ASLANPOUR; GILL; TOOSI, 2020).

3.2.1 Taxonomia

Aslanpour, Gill e Toosi (2020) propõem uma taxonomia que distingue métricas comuns a todas as camadas do contínuo nuvem-coisas e métricas específicas de cada camada.

As **métricas comuns** incluem: *Throughput* (vazão), Congestionamento de Rede, Violação de SLA, Tempo para Adaptação/Escalabilidade, Razão de Competição, Taxa de Acerto de *Cache*, Custo/Lucro, Consumo de Energia, Temperatura e Segurança e Privacidade.

Entre as **métricas específicas por camada**, destacam-se (ASLANPOUR; GILL; TOOSI, 2020):

- a) *IoT/Mist*: latência ultrabaixa, tempo de vida do dispositivo IoT, confiabilidade crítica e consumo de energia dos dispositivos;
- b) *Borda*: latência muito baixa, temperatura de *gateways* e roteadores, confiabilidade moderada e disponibilidade;
- c) *Névoa*: latência muito baixa, temperatura dos servidores de névoa, segurança, confiabilidade moderada; e
- d) *Nuvem*: latência baixa (escala de segundos), temperatura dos *data centers*, métricas específicas por modelo de implantação (privacidade em nuvens privadas; tempo de adaptação em nuvens multiprovedor e federadas).

De especial relevância para este trabalho é a métrica denominada **Sobrecusto/Leveza da Técnica** (*Technique's Overhead/Lightness*), classificada na fase *Executor* do ciclo MAPE-K (ASLANPOUR; GILL; TOOSI, 2020). Ela quantifica o impacto que a própria solução de orquestração ou monitoramento impõe ao ambiente monitorado. Em ambientes com recursos constritos, como os de borda e névoa, essa métrica é especialmente crítica: o sobrecusto introduzido pela ferramenta de monitoramento pode comprometer o desempenho das aplicações que se pretende monitorar (ASLANPOUR; GILL; TOOSI, 2020; COSTA *et al.*, 2022). A redução da dimensionalidade dos contadores

de monitoramento é identificada por Costa *et al.* (2022) como um dos principais caminhos para pesquisas futuras nesse domínio, e constitui a motivação central do benchmark proposto neste trabalho.

3.2.2 Métricas Relevantes

As métricas a seguir são as identificadas por Aslanpour, Gill e Toosi (2020) como relevantes para o contínuo nuvem-coisas, e são apresentadas na Figura 8. À partir da síntese feita no trabalho, Aslanpour, Gill e Toosi (2020) também identifica na Figura 9 as métricas comuns a todas as camadas do contínuo nuvem-coisas.

Figura 8 – Métricas relevantes para cada paradigma de computação.

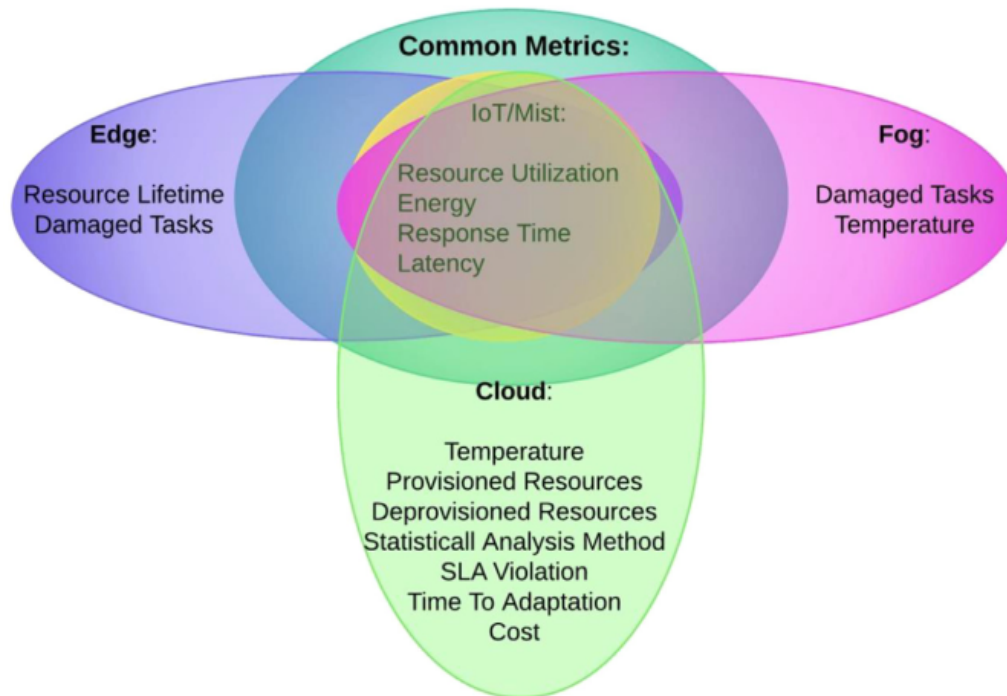


Fonte: Extraído de Aslanpour, Gill e Toosi (2020)

3.2.2.1 Métricas da Fase Monitor

A fase Monitor é responsável por observar o desempenho da infraestrutura, desde o nível de aplicação (requisições de usuários, violações de SLA) até o nível de hardware (utilização de CPU, temperatura). São as mais diretamente relacionadas a ferramentas

Figura 9 – Métricas comuns a todos os paradigmas de computação.



Fonte: Extraído de Aslanpour, Gill e Toosi (2020)

de monitoramento de sistema. As principais métricas identificadas são (ASLANPOUR; GILL; TOOSI, 2020):

- Utilização de recursos** (*Resource Utilization*): percentual de recursos consumido pela carga de trabalho recebida. Define-se como a razão entre o tempo real gasto pelo recurso executando carga e o total de tempo de atividade. Recursos podem ser CPU, memória, armazenamento ou largura de banda;
- Carga de recursos** (*Resource Load*): percentual de carga sobre um *host*, Máquina Virtual (*Virtual Machine*) (VM) ou contêiner, calculado pelo número de tarefas aguardando na fila de CPU somado às que estão em execução. Difere da utilização por incluir tarefas aguardando;
- Vazão** (*Throughput*): razão entre o número de tarefas que chegam ao sistema e o número processado em um período. Oferece visão geral do desempenho, mas não garante baixo tempo de resposta nem baixo custo;
- Tempo de vida do recurso** (*Resource Lifetime*): tempo em que um recurso (VM, contêiner) permanece ativo. Tempos de vida maiores, com menor custo de locação, é o objetivo principal de técnicas de escalonamento, pois reduzem a espera do usuário por novos recursos;
- Máximo de recursos em execução** (*Maximum Running Resource*): número máximo de recursos provisionados simultaneamente. Indica a resistência do

sistema a picos de demanda e *flash crowds*;

- f) **Tempo de resposta** (*Response Time*): intervalo entre o envio de uma requisição e a conclusão da tarefa. É uma das métricas mais avaliadas na literatura e é especialmente relevante para aplicações com requisitos de tempo real;
- g) **Latência/Atraso** (*Delay Time*): diferença entre o tempo real de conclusão de uma tarefa e o tempo desejado de conclusão. Latência e atraso são usados de forma intercambiável e são métricas centrais para aplicações de névoa e borda;
- h) **Congestionamento de rede** (*Network Congestion/Traffic Control*): carga produzida pelos usuários em um período, medida pelo número de requisições chegando ao sistema. Reflete o quanto as VMs ou nós estão ocupados atendendo tráfego;
- i) **Tarefas danificadas** (*Number of Damaged Tasks*): número de requisições que não obtiveram resposta ou não obtiveram resposta dentro do prazo. Inclui tarefas canceladas por liberação de *host* e tarefas falhas por encerramento abrupto ou estouro de *timeout*;
- j) **Violação de SLA** (*SLA Violation*): número ou percentual de tarefas que experimentaram atraso superior ao acordado contratualmente. Violações podem implicar penalidades ao provedor;
- k) **Consumo de energia** (*Energy Consumption*): quantidade de energia consumida por um recurso para executar a carga de trabalho. Com a proliferação de dispositivos IoT com restrições energéticas, esta métrica tornou-se central para técnicas de otimização no contínuo nuvem-coisas;
- l) **Temperatura** (*Temperature*): grau de calor gerado pela infraestrutura durante a execução de tarefas. Relevante principalmente para *data centers* em nuvem; em dispositivos de borda e IoT, a restrição energética é geralmente a preocupação predominante;
- m) **Confiabilidade** (*Reliability*): capacidade do mecanismo de operar corretamente sob determinadas condições, mesmo com falhas ocasionais. Calculada com base no *Mean Time Between Failures* (MTBF) e *Mean Time To Repair* (MTTR); e
- n) **Disponibilidade e Confiança** (*Availability and Trustworthiness*): disponibilidade é a capacidade do sistema de estar presente a qualquer momento em que um usuário emite uma requisição; confiança mede o grau em que a técnica lida com dispositivos não confiáveis, situação especialmente relevante em computação em neblina, onde nós IoT podem abandonar a rede sem aviso.

3.2.2.2 Métricas da Fase Analyzer

A fase Analyzer processa os parâmetros monitorados para produzir valores mais precisos e previsões sobre o estado futuro do sistema. Métodos de aprendizado de máquina são amplamente empregados nessa fase (ASLANPOUR; GILL; TOOSI, 2020):

- a) **Medidas de análise estatística** (*Statistical Analysis Measurements*): métodos para análise de séries temporais dos parâmetros monitorados, com objetivo de extrair estatísticas significativas e prever comportamentos futuros. Os critérios mais utilizados são *Mean Square Error* (MSE), *Root Mean Square Error* (RMSE), *Mean Absolute Percentage Error* (MAPE), *Mean Absolute Error* (MAE) e *Decision Feedback Predictor* (PRED). Com os avanços em aprendizado profundo e aprendizado federado na borda, esses métodos convencionais precisam ser reavaliados no contexto de computação em borda.

3.2.2.3 Métricas da Fase Planner

A fase Planner anuncia decisões de orquestração, como alocação de VMs e migração de tarefas. As métricas associadas avaliam a qualidade e o impacto dessas decisões (ASLANPOUR; GILL; TOOSI, 2020):

- a) **Número de decisões de orquestração** (*Number of Orchestration Decisions*): total de decisões tomadas pelo orquestrador, como escalonamento ou offloading. Um número elevado pode indicar maior consciência sobre mudanças no sistema, mas também implica maior sobrecusto;
- b) **Decisões contraditórias** (*Contradictory Decisions*): número de vezes que decisões já tomadas são revertidas por imprecisão. Valor baixo indica eficiência do orquestrador frente a flutuações de carga;
- c) **Tempo de adaptação e escalabilidade** (*Time to Adaptation/Scalability*): tempo necessário para estabilizar o sistema após a aplicação de uma decisão. Escalabilidade é um caso especial que mede a capacidade de adicionar ou remover recursos em resposta à demanda, tanto em ambientes de nuvem quanto em IoT;
- d) **Razão de competição** (*Competition Ratio*): razão na qual os recursos competem para atender a uma requisição. Relevante para avaliação de técnicas de escalonamento de tarefas em computação em borda; e
- e) **Taxa de acerto de cache** (*Cache Hit Ratio*): sucesso do mecanismo de cache em reduzir a transmissão de dados pela rede. Mede a proporção de requisições atendidas localmente sem necessidade de busca em camadas superiores, reduzindo latência e consumo de banda.

3.2.2.4 Métricas da Fase Executor

A fase Executor implementa as ações decididas pelo Planner. As métricas associadas avaliam a fidelidade e o impacto da execução efetiva (ASLANPOUR; GILL; TOOSI, 2020):

- a) **Ações de orquestração** (*Orchestration Actions*): frequência com que as decisões do Planner são efetivamente executadas. Mede também a capacidade de mitigar ações desnecessárias decorrentes de flutuações de carga, rejeitando decisões inviáveis do Planner;
- b) **Recursos provisionados e desprovisionados** (*Provisioned/Deprovisioned Resources*): número de recursos (VMs, contêineres) provisionados ou removidos durante a execução. Métrica central em técnicas de escalonamento automático (*auto-scaling*);
- c) **Ações contraditórias** (*Contradictory Actions*): número de vezes que ações já executadas são revertidas por imprecisão. Distingue-se das decisões contraditórias por ocorrer na fase de execução;
- d) **Mitigação de oscilações** (*Oscillation Mitigation*): resistência do sistema a flutuações em decisões e ações. Oscilações ocorrem quando ações de orquestração são executadas de forma muito rápida, antes que seu impacto possa ser avaliado. Mecanismos de *cooldown* são tipicamente empregados para mitigá-las;
- e) **Custo e lucro** (*Cost/Profit*): custo monetário ou de complexidade imposto ao usuário ou provedor como resultado do desempenho da técnica de orquestração. Em névoa e borda, a mensuração de custo ainda requer investigação adicional, pois os modelos de precificação da nuvem não se transferem diretamente para esses ambientes;
- f) **Sobrecusto/Leveza da técnica** (*Technique's Overhead/Lightness*): sobrecusto que a própria solução de orquestração ou monitoramento impõe ao sistema. Em termos de leveza, mede o tempo necessário para a técnica estar operacional. Em ambientes com recursos constritos, como os de borda e IoT/neblina, orquestradores que demandam múltiplos núcleos de CPU ou grande quantidade de memória não são viáveis para implantação; e
- g) **Segurança e privacidade** (*Security and Privacy*): grau em que a solução de orquestração mantém dados e serviços protegidos de atividades maliciosas, por meio de criptografia e controles de acesso.

3.3 SOBRE SOLUÇÕES DE MONITORAMENTO

Diversos trabalhos analisaram soluções de monitoramento concebidas para a nuvem e constataram que nenhuma delas é adequada para uso em ambientes de névoa (COSTA *et al.*, 2022). As soluções de monitoramento para névoa devem atender a requisitos novos,

ausentes nas soluções já consolidadas para nuvem, demandando, portanto, abordagens específicas (COSTA *et al.*, 2022).

Costa *et al.* (2022) consolidam, com base na literatura, um conjunto de propriedades que uma solução de monitoramento para névoa deve apresentar: escalabilidade, para lidar com o aumento do número de nós; resiliência ao surgimento e à remoção de nós, dada a mobilidade característica do ambiente; resiliência a alterações e falhas de rede; modularidade, permitindo adaptação ou parametrização para cenários distintos de uso (e.g. *Industrial IoT* versus gerenciamento de tráfego veicular); e localidade, mantendo o monitoramento o mais próximo possível dos recursos e serviços observados.

3.3.1 Taxonomia de Soluções de Monitoramento

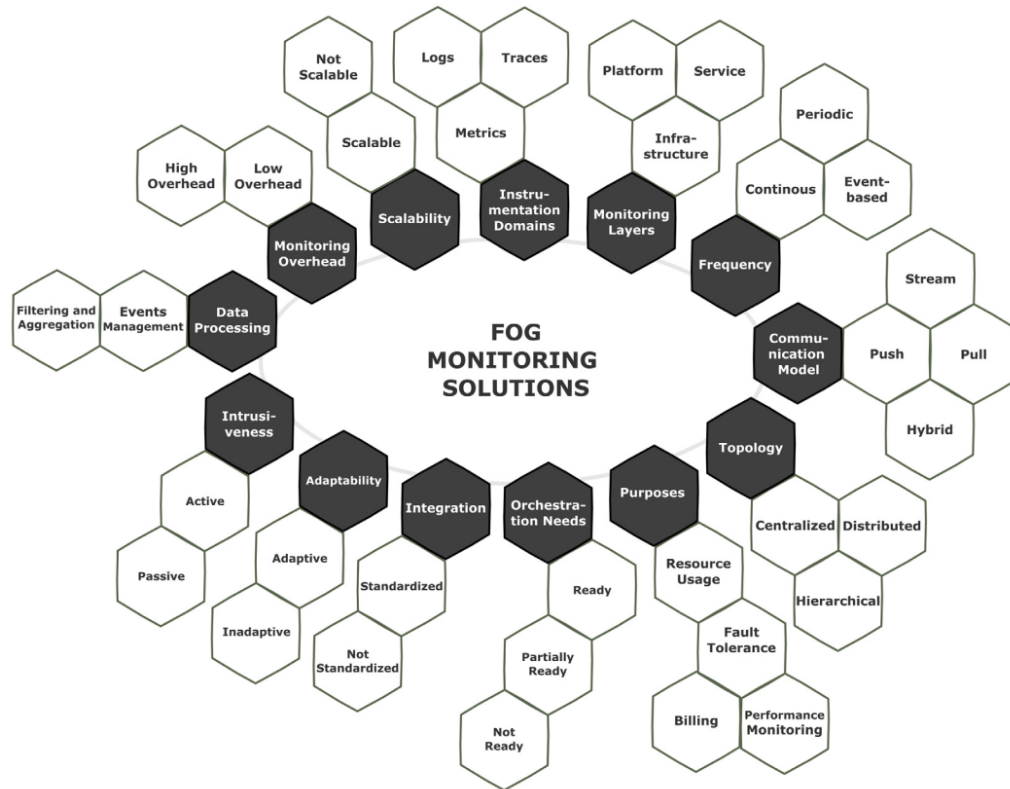
Para sistematizar a análise das soluções existentes, Costa *et al.* (2022) propõem uma taxonomia composta por treze domínios, derivada de revisão sistemática da literatura, apresentada na Figura 10. Cada domínio representa um eixo de classificação que descreve uma característica relevante de uma solução de monitoramento para névoa: *propósitos* (monitoramento de uso de recursos, monitoramento de desempenho, tolerância a falhas, faturamento); *topologia* (centralizada, hierárquica ou distribuída); *modelo de comunicação* (*pull*, *push*, híbrido ou *stream*); *frequência de monitoramento* (contínua, periódica ou orientada a eventos); *camadas monitoradas* (infraestrutura, plataforma e serviço); *domínios de instrumentação* (métricas, *logs* e *traces*); *processamento de dados* (filtragem, agregação, gerenciamento de eventos); *intrusividade* (ativa ou passiva); *escalabilidade*; *sobrecusto de monitoramento*; *adaptabilidade*; *integração* a formatos padronizados; e *atendimento às necessidades de orquestração*. Esta taxonomia é a referência adotada neste trabalho para descrever e comparar as ferramentas de monitoramento avaliadas.

3.3.2 Análise Comparativa das Soluções Existentes

Costa *et al.* (2022) aplicam a taxonomia descrita à análise de dez soluções de monitoramento propostas na literatura entre 2017 e 2022, a saber: PyMon, FMonE, *stack* Prometheus, *Osmotic Monitoring*, MCC/*Cloudlet*-based, Switch, SCB, Rule-Based, TE-Mon e FogMon. A maioria adota topologia centralizada, propósito de monitoramento de desempenho, instrumentação por métricas, modelo de comunicação *push* e frequência periódica. Apenas uma das soluções analisadas coleta *logs* e nenhuma coleta *traces*, evidenciando uma lacuna no tratamento dos demais domínios de instrumentação (COSTA *et al.*, 2022).

Entre as soluções avaliadas, a *stack* Prometheus se destaca como a única classificada simultaneamente como escalável, com baixo sobrecusto, adaptável, padronizada e capaz de atender plenamente às necessidades de orquestração (COSTA *et al.*, 2022). Por essa razão, ela foi selecionada como uma das ferramentas comparadas no benchmark proposto neste trabalho. A seção a seguir descreve as características da *stack* Prometheus à luz da

Figura 10 – Taxonomia de soluções de monitoramento para névoa.



Fonte: Extraído de Costa *et al.* (2022)

taxonomia adotada e situa, em complemento, o *Performance Co-Pilot* (PCP), ferramenta não analisada por Costa *et al.* (2022) mas amplamente disponível em distribuições Linux e que constitui o segundo objeto de estudo desta pesquisa.

3.4 SOLUÇÕES DE MONITORAMENTO RELEVANTES

A literatura identifica desafios específicos que as soluções de monitoramento para névoa precisam endereçar (COSTA *et al.*, 2022). O *churn* dos nós, decorrente da volatilidade dos recursos, exige detecção de aparições e desaparecimentos. A heterogeneidade não se restringe aos dispositivos: o ambiente de execução pode ser diverso e coexistir no mesmo nó (e.g. máquinas virtuais e contêineres servindo serviços independentes). O gerenciamento dos dados de instrumentação é particularmente crítico, pois sua coleta, processamento e transmissão podem sobrecarregar a rede em ambientes de larga escala. Soluções com topologia hierárquica ou distribuída precisam ainda lidar com a coordenação descentralizada, sob risco de dessincronização das ações de gerenciamento. Por fim, a tolerância a falhas, o gerenciamento de mobilidade, a interoperabilidade e a evitação de *vendor lock-in* completam o conjunto de desafios não totalmente atendidos pelas soluções herdadas da nuvem.

3.4.1 A Stack Prometheus

A *stack* Prometheus (COSTA *et al.*, 2022) é um sistema de monitoramento construído pela integração do servidor Prometheus a componentes complementares de código aberto, como exportadores de métricas (e.g. *Node Exporter*, para sistemas baseados em Unix, e *cAdvisor*, para uso de recursos por contêineres) e ferramentas de visualização (*dashboards*). Possui propósito de monitoramento de desempenho, adota topologia centralizada e coleta métricas em frequência periódica.

O modelo de comunicação padrão é *pull*, no qual o servidor Prometheus consulta periodicamente os agentes (exportadores) instalados nos nós monitorados; o modelo *push* também é suportado por meio de *gateway* já implementado. O formato de troca de métricas definido pelo Prometheus serviu de base para o padrão OpenMetrics, conferindo à *stack* caráter padronizado. Costa *et al.* (2022) classificam-na como adaptável devido ao acoplamento frouxo entre seus componentes de *software*, o que permite modificações para adequação a novos cenários sem grande complexidade. O servidor é compatível com diversos exportadores, podendo coletar métricas de bancos de dados, servidores *web* e serviços diversos.

Embora não tenha sido realizada avaliação formal sobre o sobrecusto que a *stack* Prometheus injeta no sistema, sua flexibilidade e modularidade permitem configurá-la para operar em cenários de baixo sobrecusto (COSTA *et al.*, 2022), o que a torna candidata aplicável a ambientes com recursos constrictos. A *stack* Prometheus monitora as camadas de infraestrutura e plataforma, implementa agregação de dados e atende plenamente às necessidades de orquestração de serviços de névoa.

3.4.2 O Performance Co-Pilot (PCP)

O *Performance Co-Pilot* (PCP) é um arcabouço de código aberto para monitoramento de desempenho em sistemas Unix e Linux, originalmente desenvolvido pela Silicon Graphics nos anos 1990 e atualmente mantido pela Red Hat como projeto comunitário. O PCP integra as principais distribuições Linux (Red Hat Enterprise Linux, Fedora, Debian, Ubuntu) e provê uma infraestrutura unificada para coleta, armazenamento e análise de métricas de desempenho com baixo acoplamento entre coletores e consumidores. Diferentemente da *stack* Prometheus, o PCP não foi objeto de análise no levantamento sistemático de Costa *et al.* (2022) sobre soluções específicas para névoa; entretanto, suas características arquiteturais o tornam um candidato adequado para o tipo de comparação proposto neste trabalho.

Arquiteturalmente, o PCP organiza-se em torno de três componentes principais. O daemon central, o *Performance Metrics Collector Daemon* (*pmcd*), executa em cada nó monitorado e atua como ponto de agregação local: recebe consultas de clientes e as encaminha aos coletores especializados. Esses coletores, denominados *Performance Metrics*

Domain Agents (PMDAs), são módulos extensíveis responsáveis por expor métricas de domínios específicos — e.g. `pmda-linux` (*kernel*, processos, memória), `pmda-proc` (estatísticas por processo), `pmda-mysql`, `pmda-nginx`, `pmda-postgresql`. A adição de um novo domínio de monitoramento se dá pelo desenvolvimento de um PMDA, sem que seja necessário modificar o `pmcd` ou as ferramentas de cliente. O terceiro componente é o conjunto de ferramentas de cliente, que consultam o `pmcd` via protocolo PCP para extrair, exibir, registrar ou exportar métricas.

Entre as ferramentas de cliente, duas são particularmente relevantes para este trabalho:

- a) **pmlogger**: serviço responsável pela gravação periódica de métricas em arquivos de *archive* no disco local, no formato binário próprio do PCP. O conjunto de métricas registrado e a frequência de amostragem são definidos por um arquivo de configuração (`pmlogger.config`), o que permite restringir a coleta apenas às séries de interesse e reduzir o sobrecusto de E/S e armazenamento. Os *archives* produzidos podem ser consultados *a posteriori* pelas demais ferramentas do PCP (`pmval`, `pminfo`, `pmlogextract`, `pmchart`), o que torna o `pmlogger` análogo, em propósito, ao papel desempenhado pelo servidor Prometheus combinado a seu banco de séries temporais. Neste trabalho, o `pmlogger` é o agente cuja sobrecarga é mensurada nos cenários experimentais com PCP; e
- b) **pmstat**: ferramenta de exibição de estatísticas em tempo real, no espírito dos utilitários clássicos `vmstat` e `iostat`, mas alimentada pelo `pmcd` e capaz de operar tanto sobre métricas ao vivo quanto sobre *archives* previamente gravados pelo `pmlogger`. Apresenta, em uma única linha por intervalo de amostragem, indicadores agregados de carga de CPU, memória, E/S e rede. No contexto deste trabalho, o `pmstat` foi utilizado tanto durante a fase de calibração dos experimentos — para verificar visualmente o estado do nó sob teste — quanto na inspeção *post mortem* dos *archives* produzidos pelo `pmlogger`.

Aplicando a taxonomia de Costa *et al.* (2022) ao PCP, observa-se um perfil semelhante em vários aspectos ao da *stack* Prometheus: topologia centralizada (um `pmcd` por nó, com agregação opcional via `pmproxy` ou federação por *archive merging*); modelo de comunicação *pull* entre clientes e `pmcd`, e entre `pmcd` e PMDAs; frequência periódica configurável; instrumentação predominantemente por métricas, sem suporte nativo a *logs* e *traces*; camadas de infraestrutura e plataforma cobertas pelos PMDAs disponíveis; e arquitetura modular, derivada da separação clara entre PMDAs e demais componentes. O PCP define ainda um formato padronizado de *archive* amplamente documentado, o que confere à solução característica análoga à de padronização observada no Prometheus via OpenMetrics.

Duas distinções relevantes em relação à *stack* Prometheus merecem registro. Primeiro, o PCP mantém os dados gravados localmente em cada nó pelo `pmlogger`, em vez

de empurrá-los ou disponibilizá-los para consulta remota por um servidor central, como faz o Prometheus. Esse modelo favorece a localidade, requisito identificado por Costa *et al.* (2022) como propriedade desejável em soluções de névoa, mas exige uma camada adicional (e.g. `pmproxy` ou cópia explícita de *archives*) para consolidação centralizada. Segundo, o PCP é distribuído e mantido como parte do sistema operacional na maioria das distribuições Linux, o que reduz a fricção de implantação em parques heterogêneos e o torna imediatamente disponível em ambientes com recursos constritos, o cenário-alvo deste trabalho.

Pelas razões expostas, o PCP foi selecionado como segunda ferramenta a ser comparada com a *stack* Prometheus no benchmark apresentado no Capítulo 4. A configuração específica adotada para o `pmlogger` tem escopo de métricas equivalente ao do *Node Exporter* do Prometheus e a metodologia de medição do sobrecusto induzido por cada ferramenta são detalhadas no capítulo seguinte.

4 AVALIAÇÃO EXPERIMENTAL

As avaliações experimentais propostas neste trabalho visam responder ao seguinte questionamento: Considerando que máquinas com recursos restritos emulam um cenário de Computação de Borda ou Névoa, ao executar nelas soluções de monitoramento adequadas para Computação em Nuvem (PCP e Prometheus), qual a sobrecarga adicionada ao sistema durante uma carga de trabalho?

Os *benchmarks* padronizados são necessários para a avaliação do sobrecusto introduzido por uma ferramenta de monitoramento sobre uma carga de trabalho. A literatura distingue dois tipos principais: *microbenchmarks*, que exercitam componentes isolados do sistema (e.g. chamadas de sistema, primitivas de disco ou rede) e fornecem medidas de granularidade fina; e *macrobenchmarks*, que reproduzem cenários completos de aplicação, gerando carga sobre múltiplos componentes simultaneamente e produzindo métricas representativas do comportamento percebido por um usuário final (COOPER *et al.*, 2010). Como o objetivo deste trabalho é avaliar o impacto das ferramentas de monitoramento em condições próximas às de operação real, foram realizadas avaliações experimentais de *macrobenchmarks* utilizando o *Yahoo! Cloud Serving Benchmark* (YCSB) em máquinas disponibilizadas gratuitamente pelo Emulab. O Emulab é uma testbed de rede gerenciada pelo grupo Flux da Universidade de Utah que proporciona recursos para que cientistas da computação e pesquisadores possam construir, debugar e avaliar redes e sistemas distribuídos (GROUP, 2026).

4.1 YCSB

O YCSB (COOPER *et al.*, 2010) é um arcabouço extensível para avaliação de sistemas de armazenamento e serviço de dados. Define um gerador de carga e um conjunto padronizado de cargas de trabalho (*workloads*) que combinam, em proporções variáveis, operações de leitura, atualização, inserção e varredura. Neste trabalho, foram utilizadas as cargas *workload A* (50% leitura / 50% atualização), *workload B* (95% leitura / 5% atualização), *workload E* (varreduras curtas com inserções) e *workload F* (*read-modify-write* com 50% de leitura e 50% de leituras seguidas de atualização), por cobrirem perfis representativos de aplicações transacionais e analíticas leves. Como sistema sob teste foi adotado o MySQL 8.4 (LTS), configurado com o mecanismo de armazenamento *InnoDB* e dois tamanhos distintos de *buffer pool* (Seção 4.3).

4.1.0.1 Cargas de Trabalho (*Workloads*)

O Quadro 1 apresenta as características das cargas de trabalho utilizadas neste experimento, conforme definido pelo YCSB (COOPER *et al.*, 2010, 2026). A coluna de operações indica a proporção de cada tipo de operação, e a última coluna apresenta uma

aplicação típica que exemplifica o perfil de acesso correspondente.

Quadro 1 – Cargas de trabalho do YCSB utilizadas no experimento.

Carga	Operações	Exemplo de aplicação
<i>Workload A</i>	50% leitura, 50% atualização	Registro de sessão de usuário
<i>Workload B</i>	95% leitura, 5% atualização	Marcação de fotos; leituras dominantes
<i>Workload E</i>	95% varredura, 5% inserção	Conversas em tópicos; cada varredura cobre as postagens de um tópico
<i>Workload F</i>	50% leitura, 50% <i>read-modify-write</i> [†]	Banco de dados de usuários com leitura e modificação de registros

Nota: [†] *Workload F* executa operações de *read-modify-write*: cada “atualização” consiste em uma leitura seguida de uma modificação do registro, diferentemente de atualizações simples.

Fonte: Extraído de Cooper *et al.* (2010) e Cooper *et al.* (2026).

4.1.1 Seleção de Métricas

A comparação coerente entre as ferramentas escolhidas para este trabalho, PCP e Prometheus, exige um mapeamento qualitativo das métricas coletadas, de modo a garantir que as configurações comparadas observem aproximadamente o mesmo subsistema do *host*. O *Node Exporter* do Prometheus expõe, em sua configuração padrão, um conjunto de coletores baseados nas interfaces `/proc` e `/sys` do Linux. O PCP, por sua vez, organiza suas métricas em *Performance Metric Domains* (PMDs), com nomenclatura hierárquica (e.g. `kernel.cpu.util.*`, `mem.util.*`, `disk.dev.*`). Isto significa que nem todas as informações possuem contrapartida direta entre as duas ferramentas, e que se faz necessário selecionar o conjunto de métricas para coerência na sobrecarga.

A comparação quantitativa do número total de métricas evidencia uma diferença substancial entre as ferramentas em suas configurações de fábrica. Em uma configuração equivalente em escopo de coleta, o PCP produz um conjunto de métricas comparável ao do Prometheus. Já em sua configuração padrão completa, o PCP coleta um número significativamente maior de séries, incluindo PMDAs específicos para subsistemas que não possuem contraparte direta no *Node Exporter*. Para as máquinas deste experimento, foi verificada a disponibilidade de 3390 métricas através do comando `pminfo` citado na Seção 3.4.2. Tentou-se configurar o PCP para coletar todas as métricas num intervalo de 1 segundo, porém o agente retornou erro na inicialização. O *Node Exporter*, por sua vez, coleta aproximadamente 1000 métricas por *node*. O valor exato pode ser verificado com o comando `curl -s localhost:9100/metrics | wc -l`, e sua configuração também pode ser estendida ajustando o arquivo `/etc/default/prometheus-node-exporter`.

Para obter uma comparação adequada entre as duas ferramentas, ambas foram configuradas para coletar um conjunto de métricas comparável em escopo de coleta. As

métricas selecionadas são comuns a ambas as ferramentas e estão relacionadas ao subsistema de CPU, memória, disco e rede. Vale notar que o Prometheus possui métricas de *powersupplyclass*, pertinentes para o contexto de IoT, que não foram consideradas na comparação.

```
cat <<'EOF' | sudo tee /etc/default/prometheus-node-exporter
ARGS="--collector.disable-defaults \
--collector.cpu \
--collector.diskstats \
--collector.filesystem \
--collector.loadavg \
--collector.meminfo \
--collector.netdev \
--collector.stat \
--collector.vmstat \
--collector.diskstats.ignored-devices='^(ram|loop|fd|sr)[0-9]+$' \
--collector.filesystem.fs-types-exclude=
'^(autofs|binfmt_misc|bpf|cgroup2?|configfs|debugfs|devpts|
devtmpfs|fusectl|hugetlbfs|iso9660|mqueue|nsfs|overlay|proc|
procfs|pstore|rpc_pipefs|securityfs|selinuxfs|squashfs|sysfs|
tmpfs|tracefs|nfs|nfs4|nfsd)$' "
EOF
```

O Quadro 2 apresenta o mapeamento aproximado entre os principais grupos de métricas das duas ferramentas, derivado da análise dos coletores padrão e da documentação dos respectivos PMDAs.

Quadro 2 – Mapeamento qualitativo entre métricas do Node Exporter (Prometheus) e do PCP.

Node Exporter	Equivalente em PCP
node_cpu_seconds_total	kernel.cpu.util.*, kernel.percpu.cpu.*
node_load*	kernel.all.load
node_context_switches_total	kernel.all.pswitch
node_memory_*	mem.util.*
node_vmstat_pg*	mem.vmstat.pgpgin/pgpgout/pgfault/pgmajfault
node_disk_*	disk.all.*, disk.dev.*
node_network_*	network.interface.*
node_filesystem_*	filesys.*

Fonte: Elaborado pela autora.

O PCP foi configurado com o conjunto reduzido do arquivo `pmlogger.conf` apresentado no Anexo B, que resultou em 131 métricas apresentadas no Anexo C, obtidas

especificamente com o comando:

```
HOST=pmstat.giulia-307663.scalablesmr.emulab.net
pminfo -a /var/log/pcp/pmlogger/$HOST/20260608.00.10 -f \
> /tmp/pmstat-metrics
```

O *Node Exporter* do Prometheus, por sua vez, foi inicializado com a configuração abaixo, que resulta em aproximadamente 390 métricas apresentadas no Anexo A.

Algumas métricas expostas pelo *Node Exporter* foram desabilitadas por não possuírem equivalente em PCP. Essa exclusão garante que o conjunto de séries efetivamente coletadas pelas duas ferramentas seja, dentro das limitações de cada projeto, semanticamente equivalente. Contudo, mesmo assim a quantidade de métricas coletadas pelo Prometheus é maior do que a do PCP. As métricas sem equivalência foram sintetizadas no Quadro 3, bem como sua quantidade. Escolheu manter-se estas métricas para preservar as capacidades da ferramenta, mas seria possível alterar o arquivo de configuração e excluir métricas de acordo com o regex necessário.

Quadro 3 – Métricas do Prometheus sem equivalente no PCP

Família de métricas	Quantidade
go_*	34
process_*	7
promhttp_*	6
node_scrape_collector_*	16
node_exporter_build_info	1
node_cpu_core_throttles_total, node_cpu_guest_seconds_total	2
node_disk_discard_*, node_disk_flush_requests_*, node_disk_info, node_disk_io_now	8
node_filesystem_files*, node_filesystem_device_error, node_filesystem_readonly	4
node_network_receive_compressed, fifo, frame, multicast_total	4
node_network_transmit_compressed, fifo, carrier, colls_total	4
Total	86

Fonte: Elaborado pela autora.

4.2 SETUP EXPERIMENTAL

4.2.1 O Testbed Emulab

Os experimentos foram realizados sobre o Emulab, um testbed de rede desenvolvido e mantido pela Universidade de Utah, em operação contínua desde o início dos anos 2000 e amplamente adotado pela comunidade acadêmica de sistemas distribuídos e redes. O Emulab disponibiliza um conjunto heterogêneo de nós físicos que podem ser alocados de forma dedicada por pesquisadores; durante o período de alocação, o usuário tem acesso

exclusivo aos recursos da máquina, incluindo memória, CPU e interfaces de rede, o que evita contaminação das medidas por carga concorrente de outros experimentos. (GROUP, 2026)

A reserva de recursos é descrita por meio de uma especificação de *topology* (em geral, um *script ns* ou um arquivo *Profile* no Emulab moderno), na qual o experimentador declara o número e o tipo dos nós desejados, as ligações lógicas entre eles e, opcionalmente, restrições de largura de banda, latência e perdas em cada *link*. O testbed provisiona automaticamente a topologia solicitada, instala uma imagem de sistema operacional escolhida pelo usuário e disponibiliza acesso por SSH a cada nó. Esta capacidade de configurar topologias controladas e reproduzíveis é particularmente relevante para *benchmarks* de monitoramento, pois garante condições de rede idênticas entre execuções comparadas.

A escolha do Emulab para este trabalho deve-se a três fatores principais: (i) a disponibilidade de nós físicos dedicados, requisito necessário para mensurar de forma fidedigna o sobrecusto de CPU e memória das ferramentas de monitoramento; (ii) a heterogeneidade de gerações de *hardware* disponíveis, que permite caracterizar máquinas com recursos relativamente restritos (como a classe PC3000) representativas de cenários de borda e névoa; e (iii) o acesso aberto à comunidade acadêmica.

4.2.2 Nós Utilizados

Foram utilizadas duas classes de nós, alocadas no *site* Emulab:

- a) **PC3000**: classe de nó adotada para hospedar o sistema sob teste. Trata-se de uma geração de *hardware* mais antiga do parque do Emulab, equipada com um processador Intel Xeon de soquete único, núcleo único e dois *threads* de *hardware* (*hyper-threading*), 2 GB de memória RAM e armazenamento total de aproximadamente 208 GB. Esta combinação de baixa contagem de núcleos e memória limitada torna o PC3000 representativo do perfil de máquina com recursos restritos visado por este trabalho. As especificações detalhadas são apresentadas no Quadro 4. Em cada experimento, este nó executa o servidor MySQL e, conforme o cenário, o agente de monitoramento **pmlogger** (PCP) e/ou o *Node Exporter* do Prometheus. É sobre este nó que o sobrecusto da ferramenta de monitoramento é mensurado;
- b) **D430**: classe de nó significativamente mais recente, baseada em processadores Intel Xeon E5-2630 v3 (microarquitetura Haswell-EP, lançada em 2014) em configuração de soquete duplo. Cada processador possui 8 núcleos físicos com 2 *threads* por núcleo, totalizando 32 *threads* de *hardware*; soma-se a isso 64 GB de memória RAM e cerca de 2 TB de armazenamento. Esta classe foi adotada como gerador de carga, executando o cliente YCSB e disparando as operações contra o servidor MySQL. A escolha de uma máquina substancialmente mais robusta para o gerador de carga garante que o cliente YCSB não se torne o

gargalo do experimento, isolando o efeito sob investigação no nó PC3000. As especificações detalhadas são apresentadas no Quadro 5.

Quadro 4 – Especificações do nó PC3000 (Emulab).

Arquitetura	x86_64 (64 <i>bits</i>)
CPU	Intel Xeon a 3,0 GHz; 1 soquete, 1 núcleo, 2 <i>threads</i> (<i>hyper-threading</i>)
Virtualização	Não suportada em <i>hardware</i>
Memória	2 GB de RAM
Disco	≈ 208 GB no total (≈ 68 GB para volume de sistema, ≈ 140 GB para uso experimental)
Rede	Interfaces Ethernet configuradas a 2 Gbps no <i>link</i> experimental

Fonte: Portal Emulab (<<https://www.emulab.net/portal/show-nodetype.php?type=pc3000>>).

Quadro 5 – Especificações do nó D430 (Emulab).

Arquitetura	x86_64 (64 <i>bits</i>)
CPU	2× Intel Xeon E5-2630 v3 (Haswell-EP) a 2,4 GHz; 8 núcleos por soquete e 2 <i>threads</i> por núcleo, totalizando 32 <i>threads</i> de <i>hardware</i>
Virtualização	Suportada em <i>hardware</i> (Intel VT-x)
Memória	64 GB de RAM
Disco	≈ 2 TB no total (≈ 119 GB para volume de sistema, ≈ 1,9 TB para uso experimental)
Rede	Interfaces Ethernet configuradas a 2 Gbps no <i>link</i> experimental

Fonte: Portal Emulab (<<https://www.emulab.net/portal/show-nodetype.php?type=d430>>).

A diferença de gerações entre as duas classes é deliberada e bastante expressiva: o D430 oferece cerca de 32× mais *threads* de *hardware* e 32× mais memória RAM do que o PC3000, embora opere com uma frequência nominal de relógio inferior (2,4 GHz contra 3,0 GHz). O gerador de carga deve ser suficientemente provisionado para sustentar a vazão máxima exigida pela carga YCSB sem se tornar o componente limitante, enquanto o sistema sob teste deve representar um ambiente de recursos constritos, no qual o sobrecusto de monitoramento se manifesta de forma mais evidente. Em ambas as classes foi utilizada a imagem padrão Ubuntu 22.04 LTS (UBUNTU22-64-STD), assegurando uniformidade da pilha de *software* de base entre os nós.

A topologia do experimento foi declarada com largura de banda de 2 Gbps no *link* entre o gerador de carga (D430) e o sistema sob teste (PC3000). Em termos de vazão efetivo, o teto teórico de 2 Gbps corresponde a aproximadamente 250 MB/s de dados transmitidos, capacidade o suficiente para o tráfego gerado pelo cliente YCSB sobre o

protocolo MySQL. As cargas YCSB adotadas operam sobre registros de 1 kB, com pacotes de requisição e resposta tipicamente abaixo desse tamanho; mesmo nas *workloads* mais agressivas em vazão, como a *workload B*, dominada por leituras, garantindo que a rede não constitua gargalo do experimento.

4.2.3 Cenários de Execução

Para isolar o efeito do monitoramento, cada combinação de *workload* e tamanho de *buffer pool* foi executada em três configurações: (i) *baseline*, com o MySQL em execução isolada; (ii) com o agente *pmlogger* do PCP em execução concorrente; e (iii) com o *Node Exporter* do Prometheus em execução concorrente. Os dados do MySQL foram mantidos em arquivos sobre o sistema de arquivos da imagem do nó (Seção 4.4).

Os experimentos foram executados em dois cenários, distinguidos pelo tamanho do *InnoDB buffer pool*. O *buffer pool* é o componente do MySQL responsável pelo *cache* de páginas de dados e índices em memória. Sua dimensão determina a fração da carga de trabalho que pode ser servida *sem* acesso a disco, o que reduz a latência para a obtenção dos dados e, portanto, modula fortemente o perfil de uso de recursos observado. Os valores escolhidos foram 128 MB e 1 GB. O primeiro é o valor padrão do *buffer pool* no MySQL, o que torna este cenário representativo de implantações sem ajuste fino de configuração; já o segundo segue a recomendação documentada de alocar entre 50% e 75% da memória RAM disponível em servidores dedicados para o *buffer pool*.

4.2.3.1 *Buffer pool* de 128 MB

Com *buffer pool* de 128 MB, o sistema opera em regime limitado por E/S de disco, dado que o conjunto de trabalho da carga YCSB excede largamente o *cache* disponível. Nesse regime, a vazão (vazão) do MySQL é dominada pela latência de acesso ao disco, e o sobrecusto de CPU introduzido pelos agentes de monitoramento tende a ser absorvido pelos períodos de espera por E/S.

4.2.3.2 *Buffer pool* de 1 GB

A literatura comercial e a documentação oficial do MySQL recomendam dimensionar o *buffer pool* entre 50% e 75% da memória física disponível em servidores dedicados. O cenário de 1 GB segue a recomendação para a configuração de memória do nó PC3000 utilizado, mantendo ainda margem para o sistema operacional e para os agentes de monitoramento. Neste regime, a fração da carga atendida em memória aumenta significativamente, e o sistema passa a ser limitado por CPU e por contenção de *locks* internos do MySQL, condição na qual o sobrecusto de CPU dos agentes de monitoramento se torna mais facilmente perceptível na vazão total.

4.3 ANÁLISE DOS RESULTADOS

Inicialmente, com os resultados do YCSB, foram elaborados gráficos de Latência em função da Vazão, apresentados nas Figuras 13 a 14. Eles apresentam os valores de acordo com a operação executada no banco de dados, conforme consta na legenda, distinguindo a latência para leitura, escrita, atualização e leitura seguida de atualização. Antes de prosseguir com a análise, vale notar a similaridade entre alguns *Workloads*, mais especificamente o *workload A* e o *workload F*, que apresentam uma distribuição de operações de leitura e atualização de 50% cada, sendo que o primeiro realiza atualizações regulares e o segundo realiza leituras seguidas de atualizações; bem como entre o *workload B* e o *workload E*, que apresentam uma distribuição de operações de 95% de leitura e 5% de inserção, sendo que o primeiro realiza atualizações e o segundo realiza inserções. As operações de inserção, atualização e leitura seguida de atualização são distintas entre si mas semelhantes, e dada a proporção de operações realizadas, julga-se valioso comparar essas métricas durante as observações dos resultados.

Também é importante notar que a quantidade de *threads* escolhida para todas as varreduras foi heterogênea para o mesmo *workload* em distintos tamanhos de *buffer pool*, o que introduz uma variável de confusão na análise. Isto ocorreu em função de dificuldades continuadas na execução dos testes, e é um ponto a ser melhorado em trabalhos futuros.

Para as operações de leitura, cinco das maiores vazões foram da máquina sem monitoramento, ao passo que as outras três foram da máquina com o PCP, nenhuma foi do Prometheus. Há aumento expressivo da latência com o *buffer pool* de 128 MB nos *Workloads A* e *F* se comparado com o *buffer pool* de 1 GB conforme cresce a quantidade de *threads*, subindo de algo em torno de 10 ms para algo em torno de 90 ms. Isto, contudo, nada diz do sobrecusto das ferramentas de monitoramento, mas sim do regime de operação do banco de dados, pois não há variação na latência em função do monitoramento dentro de uma execução específica do YCSB.

Além disso vale destacar que o *workload E* é o único que mostra aumento indefinido de latência nos cenários de saturação para as operações de leitura. A saturação é caracterizada pelo atingimento do teto de vazão e subsequente crescimento abrupto de latência conforme a quantidade de *threads* aumenta. Nos outros *Workloads* a latência permanece bem comportada para as leituras mesmo que a vazão atinja um limite.

Para melhor compreensão dos resultados das operações de escrita, é importante notar que geram *locks* (travas) no banco de dados. Elas podem ser de tipos diferentes, como exclusivo ou compartilhado, e podem ser de linhas ou de tabelas. São responsáveis por garantir a consistência dos dados, evitando que operações concorrentes os modifiquem ao mesmo tempo, bem como que leituras incorram em informações desatualizadas. Abaixo, discorre-se brevemente sobre cada tipo disponível para o MySQL.

O InnoDB implementa cinco tipos de *locks* (ORACLE CORPORATION, 2026):

- a) **Record lock**: trava sobre um registro de índice específico;
- b) **Gap lock**: trava sobre o intervalo entre registros de índice, ou sobre o espaço antes do primeiro ou após o último registro. Integra o equilíbrio entre desempenho e concorrência, sendo empregado em determinados níveis de isolamento de transação, mas desnecessário quando se usa um índice exclusivo para localizar uma única linha;
- c) **Next-key lock**: combinação de um *record lock* sobre o registro de índice com um *gap lock* sobre o intervalo anterior a ele;
- d) **Intention lock**: trava em nível de tabela que sinaliza qual tipo de trava (compartilhada ou exclusiva) uma transação precisará adquirir futuramente sobre uma linha, habilitando o suporte a travamento de múltipla granularidade;
- e) **Insert intention lock**: subtipo de *gap lock* definido por operações INSERT antes da inserção da linha. Permite que múltiplas transações que inserem no mesmo intervalo de índice não precisem aguardar umas às outras, desde que não insiram na mesma posição.

Estes *locks* são utilizados pelas distintas operações da seguinte forma (ORACLE CORPORATION, 2024):

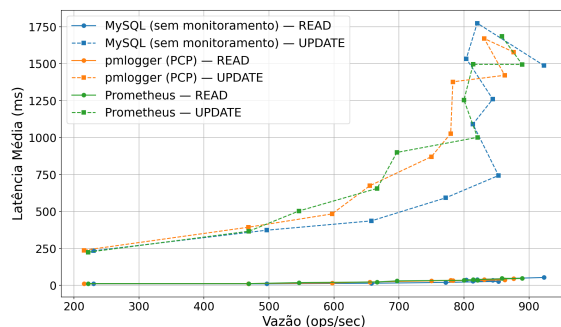
- a) **SELECT ... FROM**: realiza leitura consistente de um *snapshot* do banco de dados sem impor travas, exceto no nível de isolamento **SERIALIZABLE**, em que a busca adquire *next-key locks* compartilhados nos registros encontrados, sendo necessário apenas um *record lock* quando se usa um índice exclusivo para localizar uma única linha;
- b) **INSERT**: define um *record lock* exclusivo na linha inserida, não sendo um *next-key lock* e, portanto, não impedindo que outras sessões insiram no intervalo anterior à linha. Antes da inserção é definido um *insert intention gap lock*, que permite a múltiplas transações inserir no mesmo intervalo de índice sem aguardar umas às outras, desde que não insiram na mesma posição;
- c) **INSERT INTO T SELECT ... FROM S WHERE ...**: define um *record lock* exclusivo sem *gap lock* em cada linha inserida em T. Na tabela de origem S, o InnoDB realiza leitura consistente se o nível de isolamento for **READ COMMITTED**, ou adquire *next-key locks* compartilhados nos demais níveis;
- d) **UPDATE ... WHERE ...**: define um *next-key lock* exclusivo em cada registro encontrado na busca, impedindo que outras transações modifiquem a linha ou insiram registros no intervalo imediatamente anterior a ela. Quando se usa um índice exclusivo para localizar uma única linha, apenas um *record lock* de índice é necessário;

- e) `SELECT ... FOR UPDATE` e `SELECT ... FOR SHARE`: adquirem travas nas linhas varridas e as liberam para as que não satisfazem a condição de inclusão. Para leituras com trava (`FOR UPDATE` ou `FOR SHARE`), `UPDATE` e `DELETE`, o tipo de trava pode variar: com índice exclusivo e condição única, o InnoDB trava apenas o registro encontrado, sem o intervalo anterior; com condição de faixa ou índice não exclusivo, o InnoDB trava o intervalo varrido com *gap locks* ou *next-key locks*, bloqueando inserções de outras sessões na faixa coberta.

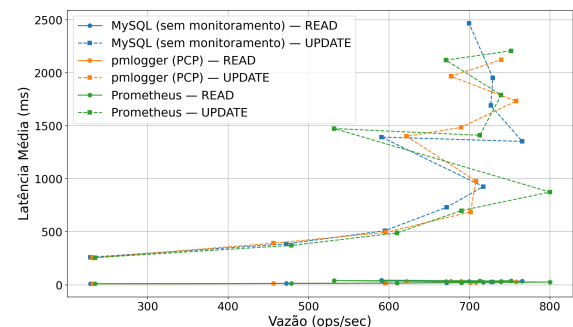
Para as outras operações do banco de dados, de forma geral a carga de trabalho *Workload B*, com 5% de operações de atualização, foi a que apresentou melhor desempenho em termos de vazão, com valores acima dos demais *workloads*, tanto no *buffer pool* de 1 GB quanto no de 128 MB, atingindo picos na região de saturação de 900 ops/s no primeiro e 950 ops/s no segundo, mais do que 50% maior do que a segunda maior vazão, cerca de 400 ops/s no *Workload E*. Que estes dois cenários apresentem maiores vazões é esperado, visto que são compostos por 95% de operações de leitura, operação que se vale de *locks* compartilhados do banco de dados, propícios para concorrência. A vazão elevada do *Workload B*, contudo, reflete em uma latência média mais alta do que as demais para as operações de atualização, em torno de 500 ms antes da saturação.

Já o *Workload E*, com 5% de operações de inserção e as demais de leitura, foi o que apresentou os menores valores de latência, com média de 70ms tanto no *Buffer pool* de 1 GB quanto no de 128 MB antes da saturação. A vazão, por sua vez, foi a segunda maior entre todos os *workloads*, saturando em torno de 400 ops/s, como citado anteriormente. As curvas de latência vs. vazão para os *Workloads B*, *E*, *A* e *F* são apresentadas, respectivamente, nas Figuras 11, 12, 13 e 14.

Figura 11 – *Workload B* (95% leitura / 5% atualização) — Latência vs. vazão.



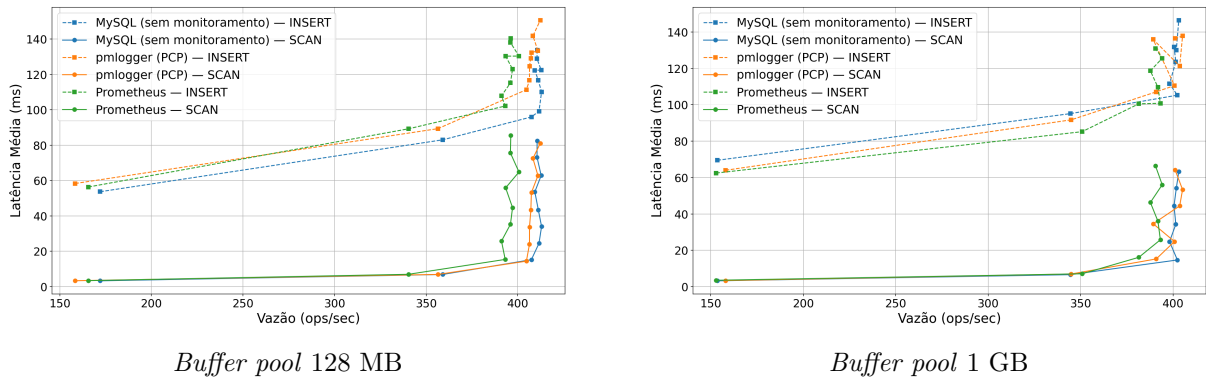
Buffer pool 128 MB



Buffer pool 1 GB

Fonte: Elaborado pela autora.

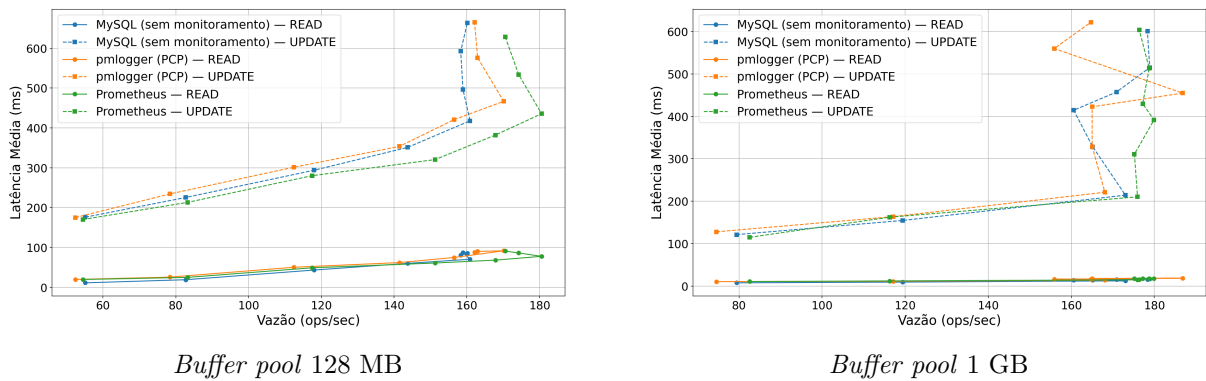
Ou seja, a natureza distinta dos *locks* das operações de inserção e atualização dos *Workloads B* e *E* gera uma discrepância de desempenho entre os dois cenários, apesar de ambos possuírem 95% de operações de leitura. O primeiro consegue atingir vazões maiores para todas as operações com aumento da latência para operações de atualização, enquanto

Figura 12 – *Workload E* (95% leitura / 5% inserção) — Latência vs. vazão.

Fonte: Elaborado pela autora.

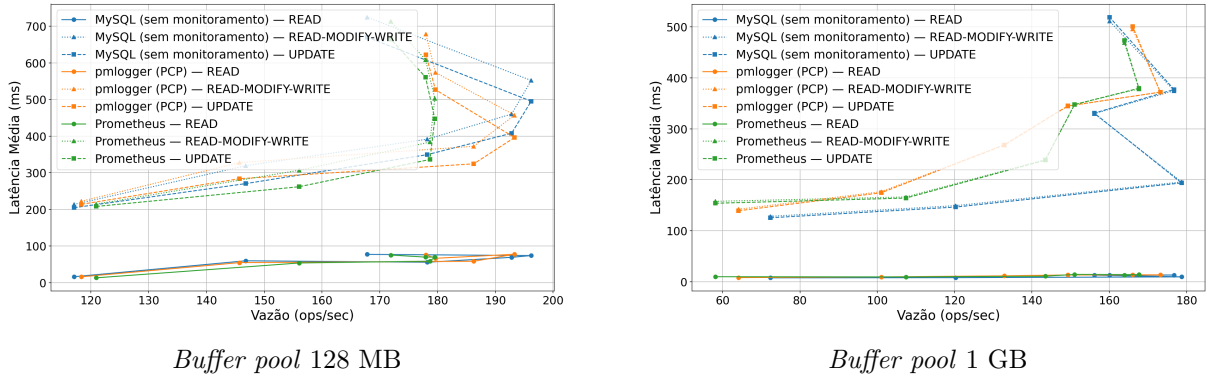
o segundo é mais limitado na vazão mas possui latências menores. Em termos de *buffer pool*, ambos apresentaram latências ligeiramente menores e vazão ligeiramente maior para o *buffer pool* de 128 MB.

Já os *Workloads A* e *F* apresentaram desempenho similar em termos de vazão e latência para as operações de escrita. Em todos os cenários a vazão para operações de escrita saturou em aproximadamente 180 ops/s. Já as latências giram em torno de 300 ms antes da saturação, com valores inferiores para o *buffer pool* de 1 GB em ambos os casos. A complexidade das travas explica porque estes dois cenários tiveram as menores vazões, e sua similaridade explica porque os resultados foram consistentes entre si. Em termos do *buffer pool*, tanto as operações de leitura quanto as de atualização apresentaram menores latências no de 1 GB, mas vazões consistentes entre ambos. Essas diferenças são mais significativas do que as observadas nos demais *workloads*, com latências expressivamente diferentes.

Figura 13 – *Workload A* (50% leitura / 50% atualização) — Latência vs. vazão.

Fonte: Elaborado pela autora.

Para responder o questionamento central do trabalho acerca da sobrecarga imposta pelas ferramentas de monitoramento, deve-se comparar agora o desempenho dos cenários monitorados com o *baseline* tanto em termos de latência quanto em termos de vazão.

Figura 14 – *Workload F* (read-modify-write) — Latência vs. vazão.

Fonte: Elaborado pela autora.

Inspecionando visualmente para o primeiro, alguns cenários têm uma tendência clara de menores latências para a máquina sem monitoramento, em específico o *Workload E* exibe este comportamento de forma pronunciada, mas também o *Workload F* e o *buffer pool* de 128 MB do *Workload B*. Em contrapartida, a máquina com o Prometheus têm as menores latências no *Workload A*. A máquina do *pmlogger* teve desempenho intermediário em todos os cenários. Em termos de vazão a descrição é semelhante; fora da saturação, enquanto a máquina ainda performa dentro do esperado, a máquina sem monitoramento costuma ter uma maior vazão para uma mesma latência. Contudo, a diferença entre vazões/latências não é significativa. Com isto entende-se que a própria variabilidade dos resultados para várias execuções do mesmo cenário pode ser maior do que a diferença de desempenho entre os cenários monitorados e o *baseline*, como fica claro para o *Workload A*, na Figura 13.

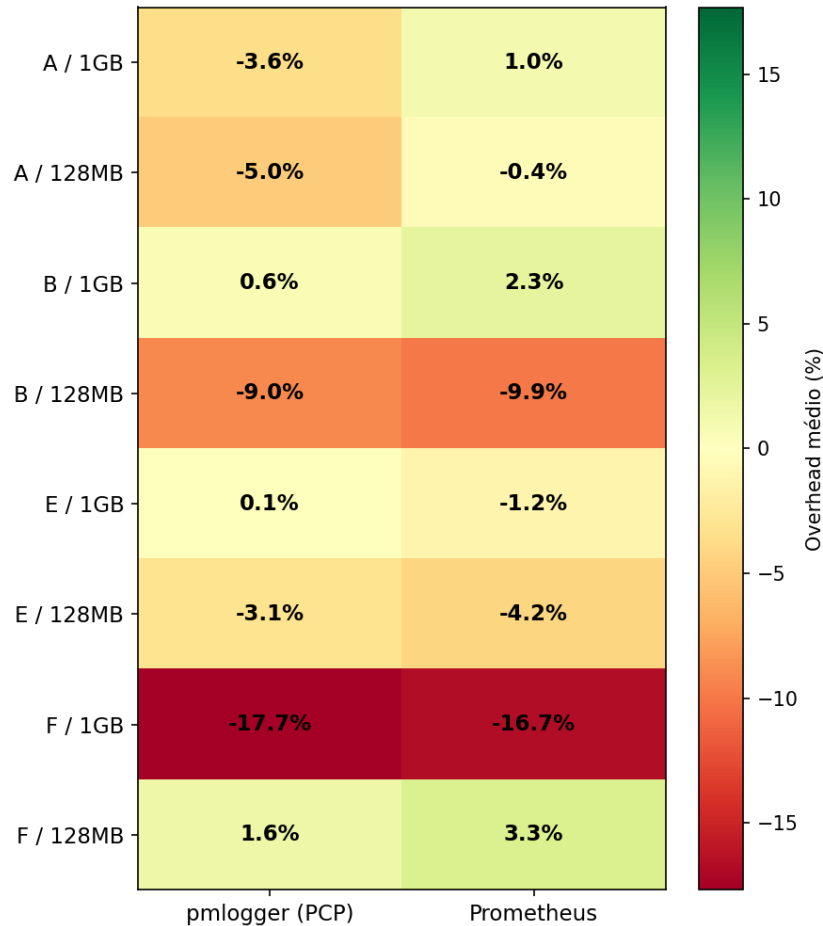
Na Figura 15 é apresentado um mapa de calor com a sobrecarga média de monitoramento por *workload* e tamanho de *buffer pool*. A vazão é medida em operações por segundo (ops/s) e *baseline* refere-se ao cenário sem monitoramento. Somente os três primeiros pontos de cada medição foram utilizados, pois os demais estão na região de saturação, na qual o aumento de *threads* causa limite no desempenho. Os valores são calculados pela seguinte fórmula:

$$\text{overhead}(\text{threads}) = \left(\frac{\text{vazão}_{\text{monitorada}}(\text{threads})}{\text{vazão}_{\text{baseline}}(\text{threads})} - 1 \right) \times 100$$

Esta expressão não foi encontrada na literatura e é adotada empiricamente neste trabalho por trazer uma inspeção visual intuitiva e interessante acerca do comportamento das ferramentas, sobretudo comparando-as entre si, ao invés de inspecionar ponto a ponto as Figuras 13 a 14. Células em vermelho (números negativos) indicam redução de vazão em relação ao *baseline*; células em verde (números positivos) indicam que o cenário monitorado superou o *baseline*.

Há redução da sobrecarga para alguns cenários, mas, de forma geral, a tendência é de

Figura 15 – Mapa de calor do overhead médio de monitoramento por *workload* e tamanho de *buffer pool*. Números negativos indicam redução de vazão em relação ao *baseline*; números positivos indicam que o cenário monitorado superou o *baseline*.



Fonte: Elaborado pela autora.

piora na performance. Este cenário vai de encontro com a análise anterior, na qual especula-se que a própria variabilidade dos resultados no mesmo cenário pode ser maior do que a sobrecarga imposta pelas ferramentas de monitoramento, mas que há tendência de piora na performance. Por isso, é difícil estabelecer qual das soluções de monitoramento tem menor sobrecarga em termos de vazão, ainda que um olhar estrito para a Figura 15 sugira que o Prometheus é a ferramenta com menor sobrecarga. Dois pontos que poderiam fornecer dados mais fidedignos seriam o aumento da quantidade de *threads* visando aumentar a granularidade das medições, e a execução de múltiplas execuções do YCSB para cada carga de trabalho e subsequente cálculo da média de todas estas execuções, eliminando desvios pontuais que afetam a média.

É necessário ressaltar que este mapa de calor utiliza-se somente dos primeiros três pontos nas Figuras 13 a 14, pois os demais estão na região de saturação, na qual o aumento de *threads* causa limite no desempenho. O limite dos desempenhos também é

uma informação relevante, com aparente tendência de pior performance na latência para o Prometheus, e que foi detalhada neste mesmo capítulo em parágrafos anteriores.

4.4 DIFICULDADES

Durante a execução do estudo, foram enfrentadas dificuldades técnicas que merecem registro por terem influenciado o desenho final dos experimentos:

- a) **Erros de inserção no YCSB:** Houve colisões de chave primária no MySQL devido à reutilização do espaço de chaves entre execuções consecutivas da fase de carga (*load*), bem como durante a execução *multithread* do *workload E*, que envolve operações de inserção no banco de dados. A mitigação exigiu o ajuste do parâmetro `insertstart` para iniciar cada execução em um *offset* distinto. Ainda assim, persistiram colisões em parte das execuções, o que motivou a alteração adicional do parâmetro `insertorder` de `hashed` para `ordered`, que geram chaves bastante distintas entre si, garantindo que não ocorressem colisões nas inserções da etapa de execução com dados existentes na etapa de carga.
- b) **Provisionamento de armazenamento no Emulab:** a criação e o *attach* de discos adicionais aos nós alocados via console do Emulab apresentaram documentação esparsa, dificultando a configuração de volumes dedicados ao *datadir* do MySQL. Como contorno, optou-se por manter os arquivos de dados do MySQL no sistema de arquivos da imagem do nó, ciente de que essa escolha limita o tamanho máximo do *dataset* mas não afeta a comparabilidade entre os cenários, pois é mantida idêntica em todos eles.
- c) **Bloqueio de acesso ao Emulab:** Para a execução e *download* dos resultados foram utilizados scripts e ferramentas de Inteligência Artificial para monitoramento contínuo e automatizado dos experimentos. Contudo, após algumas realizações o acesso ao Emulab foi bloqueado, exigindo locomoção até o laboratório para execução e *download* dos resultados, bem como a eliminação da IA como assistente. Muitos testes levaram muitas horas para a sua conclusão, e a execução deste trabalho se deu fora de horário comercial, fazendo com que a IA fosse uma assistente valiosa para a obtenção dos resultados após a conclusão dos experimentos.

5 CONCLUSÃO

O crescimento expressivo do número de dispositivos conectados à Internet (CISCO SYSTEMS, 2020) e a subsequente emergência de aplicações com requisitos rigorosos motivaram o surgimento de paradigmas computacionais complementares à nuvem, em especial a computação de borda e em névoa, dentro do contínuo nuvem-coisas. Nesses ambientes, o monitoramento é peça fundamental para a orquestração de recursos e garantia de Acordos de Nível de Serviço. O sobrecurso computacional introduzido por ferramentas de monitoramento deve ser baixo para a viabilidade operacional.

Este trabalho sintetizou os principais paradigmas de computação e buscou revisões sistemáticas sobre o estado da arte em monitoramento de sistemas operacionais para este contexto, abordando tanto métricas quanto soluções disponíveis. A seguir, definiu duas ferramentas de monitoramento utilizadas em ambientes Linux, o PCP (`pmlogger`) e o Prometheus para avaliar comparativamente, tendo como metodologia a utilização do *Yahoo! Cloud Serving Benchmark* (YCSB), que possui cargas de trabalho representativas de aplicações reais. O experimento foi realizado em máquinas disponíveis no *testbed* Emulab, uma de recursos mais abundantes executou a geração de carga do YCSB, e as outras de recursos constritos (emulando um nó de borda ou névoa) hospedaram um servidor de banco de dados MySQL, alvo das operações do YCSB.

O trabalho apresentou os paradigmas que compõem o contínuo nuvem-coisas, evidenciando que a proliferação de dispositivos conectados e seus requisitos de baixa latência, conectividade intermitente e recursos constritos motivaram o surgimento de camadas intermediárias de processamento distribuído, em particular a computação em borda e em névoa (DE DONNO; TANGE; DRAGONI, 2019; CAO *et al.*, 2020). O conceito de *Cloud Continuum* (MORESCHINI *et al.*, 2022) sintetiza essa arquitetura em três camadas complementares. As tendências de expansão desse ecossistema, como as densidades de 10^7 dispositivos por km^2 e a latência submilissegundo projetadas para redes 6G (GIORDANI *et al.*, 2020), ampliam a escala dos desafios identificados e reforçam o contexto que motiva esta pesquisa.

Nesse enquadramento, os dispositivos Linux com recursos constritos avaliados neste trabalho situam-se na camada de borda do contínuo, onde o sobrecurso introduzido por ferramentas de monitoramento é uma variável crítica para a viabilidade da operação dos serviços providos (MORESCHINI *et al.*, 2022; ASLANPOUR; GILL; TOOSI, 2020). A revisão dos paradigmas também evidenciou que o monitoramento de infraestruturas distribuídas que abrangem nuvem, névoa, borda e dispositivos IoT constitui um desafio em aberto na literatura (COSTA *et al.*, 2022), e que a redução do sobrecurso de monitoramento sem perda de observabilidade é apontada como direção de pesquisa futura (POPIOLEK, Pedro Freire; MACHADO, K. d. S.; MENDIZABAL, Odorico Machado, 2021). O presente trabalho contribui empiricamente para essa direção ao comparar, em condições contro-

ladas, o impacto de duas ferramentas de monitoramento sobre uma máquina com perfil representativo da camada de borda.

A seguir, situou-se o monitoramento como funcionalidade essencial para a orquestração, bem como o ciclo MAPE-K, através do qual o estado atualizado dos recursos e serviços é coletado e enviado ao orquestrador (COSTA *et al.*, 2022). A taxonomia de métricas proposta por Aslanpour, Gill e Toosi (2020) evidencia que, além das métricas comuns a todas as camadas do contínuo nuvem-coisas, como vazão, tolerância a falhas e consumo de energia, cada camada impõe requisitos específicos, sendo a camada de borda caracterizada por exigências de latência muito baixa, disponibilidade e confiabilidade. Entre todas as métricas identificadas, o sobrecusto da técnica (*Technique's Overhead/Lightness*), classificado na fase *Executor* do ciclo MAPE-K, ocupa posição central neste trabalho. Em ambientes com recursos restritos, o impacto que a própria ferramenta de monitoramento impõe ao sistema pode inviabilizar sua implantação (ASLANPOUR; GILL; TOOSI, 2020; COSTA *et al.*, 2022). A revisão de dez soluções de monitoramento conduzida por Costa *et al.* (2022) identificou a *stack* Prometheus como a única classificada simultaneamente como escalável, de baixo sobrecusto, adaptável, padronizada e capaz de atender plenamente às necessidades de orquestração de névoa, justificando sua seleção como objeto de estudo. O *Performance Co-Pilot* (PCP), não avaliado por Costa *et al.* (2022), foi incluído neste trabalho por integrar nativamente as principais distribuições Linux, tornando-o imediatamente disponível para comparação.

Os experimentos realizados utilizaram o *Yahoo! Cloud Serving Benchmark* (YCSB) (COOPER *et al.*, 2010) para gerar cargas transacionais, comparando três configurações: sem monitoramento (*baseline*), com **pmlogger** e com o *Node Exporter* do Prometheus. Em termos de sobrecusto de vazão, a inspeção do mapa de calor (Figura 15) indica leve vantagem do Prometheus sobre o **pmlogger** na maioria das combinações de *workload* e tamanho de *buffer pool*. Já em termos de latência, o **pmlogger** apresentou desempenho intermediário e mais estável do que o Prometheus, que exibiu tendência de latências mais elevadas nos cenários de saturação. Contudo, a variabilidade inerente aos experimentos mostrou-se da mesma ordem de grandeza que as diferenças entre os cenários monitorados e o *baseline*, incluindo casos em que a máquina com Prometheus superou a máquina sem monitoramento, como evidenciado no *Workload A* (Figura 13). Isto impede conclusões definitivas sobre qual ferramenta impõe menor sobrecusto.

Trabalhos futuros podem dedicar-se a repetir os experimentos e aumentarem a granularidade das medições, a fim de confirmar ou refutar as tendências observadas. Outros trabalhos também podem desenvolver um mapa de calor semelhante à Figura 15 para avaliar a latência, permitindo uma melhor contextualização com as demandas levantadas para a rede 6G. Vale notar que o serviço do Prometheus não iniciou com sucesso para um intervalo de monitoramento de 100ms, intervalo este que é centenas de vezes maior do que as demandas de latência da rede 6G. Outra possibilidade seria a avaliação de bases de

dados distintas do MySQL, sobretudo bases NoSQL, que podem vir a ser mais adequadas para máquinas operando na borda da rede, e outras ferramentas de benchmark também podem ser consideradas.

REFERÊNCIAS

- ABDERRAHIM, Manel *et al.* A holistic monitoring service for fog/edge infrastructures: a foresight study. *In: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. [S.l.]: IEEE, 2017. P. 337–344.
- ABREHA, Haftay Gebreslasie *et al.* Monitoring in fog computing: state-of-the-art and research challenges. **International Journal of Ad Hoc and Ubiquitous Computing**, v. 36, p. 114–130, 2021.
- ASLANPOUR, Mohammad S.; GILL, Sukhpal Singh; TOOSI, Adel N. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. **Internet of Things**, v. 12, p. 100273, 2020. ISSN 2542-6605. DOI: <10.1016/j.iot.2020.100273>.
- CAO, Keyan *et al.* An Overview on Edge Computing Research. **IEEE Access**, Institute of Electrical e Electronics Engineers (IEEE), v. 8, p. 85714–85728, 2020. ISSN 2169-3536. DOI: <10.1109/access.2020.2991734>. Disponível em: <<http://dx.doi.org/10.1109/ACCESS.2020.2991734>>.
- CISCO SYSTEMS. **Cisco Annual Internet Report (2018–2023) White Paper**. [S.l.], 2020. Disponível em: <<https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>>. Acesso em: 21 abr. 2025.
- COOPER, Brian F. *et al.* **YCSB Core Workloads**. [S.l.: s.n.], 2026. Repositório GitHub do Yahoo! Cloud Serving Benchmark. Disponível em: <<https://github.com/brianfrankcooper/YCSB/blob/master/workloads/>>. Acesso em: 2 jun. 2026.
- COOPER, Brian F. *et al.* Benchmarking Cloud Serving Systems with YCSB. *In: PROCEEDINGS of the 1st ACM Symposium on Cloud Computing (SoCC'10)*. Indianapolis, Indiana, USA: ACM, 2010. P. 143–154. DOI: <10.1145/1807128.1807152>.
- COSTA, Breno *et al.* Monitoring fog computing: A review, taxonomy and open challenges. **Computer Networks**, v. 215, p. 109189, 2022. ISSN 1389-1286. DOI: <10.1016/j.comnet.2022.109189>.
- DE DONNO, Michele; TANGE, Koen; DRAGONI, Nicola. Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. **IEEE Access**, v. 7, p. 150936–150948, 2019. DOI: <10.1109/ACCESS.2019.2947652>.
- ERICSSON. **Ericsson Mobility Report**. [S.l.], jun. 2026. Disponível em: <<https://www.ericsson.com/49ce36/assets/local/reports-papers/mobility-report/documents/2026/ericsson-mobility-report-june-2026.pdf>>. Acesso em: 7 jul. 2026.

GIORDANI, Marco *et al.* Toward 6G Networks: Use Cases and Technologies. **IEEE Communications Magazine**, Institute of Electrical e Electronics Engineers (IEEE), v. 58, n. 3, p. 55–61, 2020. ISSN 1558-1896. DOI: <10.1109/MCOM.001.1900411>.

GROUP, Flux Research. **Emulab**. [S.l.: s.n.], 2026. Acessado: 2026-05-25. Disponível em: <<https://www.emulab.net/>>. Acesso em: 25 mai. 2026.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation Y.2060: Overview of the Internet of Things**. [S.l.], 2012. Disponível em: <<https://www.itu.int/rec/T-REC-Y.2060-201206-I>>. Acesso em: 25 abr. 2025.

LIU, Fang *et al.* A Survey on Edge Computing Systems and Tools. **Proceedings of the IEEE**, v. 107, n. 8, p. 1537–1562, 2019. ISSN 0018-9219. DOI: <10.1109/JPROC.2019.2920341>.

MORESCHINI, Sergio *et al.* Cloud Continuum: The Definition. **IEEE Access**, Institute of Electrical e Electronics Engineers (IEEE), v. 10, p. 131876–131886, 2022. ISSN 2169-3536. DOI: <10.1109/ACCESS.2022.3229185>.

OLIVEIRA, Franklin *et al.* Internet of Intelligent Things: A convergence of embedded systems, edge computing and machine learning. **Internet of Things**, v. 26, p. 101153, 2024. ISSN 2542-6605. DOI: <10.1016/j.iot.2024.101153>.

ORACLE CORPORATION. **InnoDB Locking**. [S.l.: s.n.], 2026. MySQL 9.7 Reference Manual. Disponível em: <<https://dev.mysql.com/doc/refman/9.7/en/innodb-locking.html>>. Acesso em: 1 jun. 2026.

_____. **Locks Set by Different SQL Statements in InnoDB**. [S.l.: s.n.], 2024. MySQL 9.1 Reference Manual. Disponível em: <<https://dev.mysql.com/doc/refman/9.1/en/innodb-locks-set.html>>. Acesso em: 1 jun. 2026.

POPIOLEK, Pedro F.; MACHADO, Karina S.; MENDIZABAL, Odorico M. Reducing Monitoring Overhead in Virtualized Environments Through Feature Selection. *In*: ANAIS do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC). [S.l.: s.n.], 2018. DOI: <10.5753/sbrc.2018.2403>.

POPIOLEK, Pedro Freire; MACHADO, Karina dos Santos; MENDIZABAL, Odorico Machado. Low overhead performance monitoring for shared infrastructures. **Expert Systems with Applications**, v. 171, p. 114558, 2021. ISSN 0957-4174. DOI: <10.1016/j.eswa.2020.114558>.

POPIOLEK, Pedro Freire; MENDIZABAL, Odorico Machado. Monitoring and analysis of performance impact in virtualized environments. **Journal of Applied Computing Research**, v. 2, n. 2, p. 75–82, 2012. ISSN 2236-3009. DOI: <10.4013/jacr.2012.22.03>.

REICHELT, David Georg; KÜHNE, Stefan; HASSELBRING, Wilhelm. Towards Solving the Challenge of Minimal Overhead Monitoring. *In: COMPANION of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion)*. [S.l.: s.n.], 2023. P. 381–388. DOI: <10.1145/3578245.3584851>.

ROSA RIGHI, Rodrigo da *et al.* A survey on global management view: toward combining system monitoring, resource management, and load prediction. **Journal of Grid Computing**, v. 17, p. 473–502, 2019.

SHARMA, Neha; PRABHA, Chander. Computing Paradigms: An Overview. *In: 2021 Asian Conference on Innovation in Technology (ASIANCON)*. Pune, India: IEEE, 2021. P. 1–6. DOI: <10.1109/ASIANCON51346.2021.9545007>.

SYED, Hassan Jamil *et al.* Cloud monitoring: A review, taxonomy, and open research issues. **Journal of Network and Computer Applications**, v. 98, p. 11–26, 2017. ISSN 1084-8045. DOI: <<https://doi.org/10.1016/j.jnca.2017.08.021>>. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1084804517302783>>.

APÊNDICE A – MÉTRICAS DO PROMETHEUS/NODE EXPORTER

O arquivo abaixo contém as métricas disponíveis do Prometheus e *Node Exporter*, com métricas selecionadas para o experimento. As métricas estão organizadas por subsistema e mapeadas conforme o Quadro 2.

```

go_gc_duration_seconds{quantile="0"} 7.8231e-05
go_gc_duration_seconds{quantile="0.25"} 0.000126129
go_gc_duration_seconds{quantile="0.5"} 0.000162925
go_gc_duration_seconds{quantile="0.75"} 0.000217367
go_gc_duration_seconds{quantile="1"} 0.006841356
go_gc_duration_seconds_sum 1.906761386
go_gc_duration_seconds_count 3528
go_goroutines 9
go_info{version="go1.18.1"} 1
go_memstats_alloc_bytes 2.874032e+06
go_memstats_alloc_bytes_total 7.125202952e+09
go_memstats_buck_hash_sys_bytes 1.590773e+06
go_memstats_frees_total 4.730684e+07
go_memstats_gc_cpu_fraction 0.0014359991879339684
go_memstats_gc_sys_bytes 5.10248e+06
go_memstats_heap_alloc_bytes 2.874032e+06
go_memstats_heap_idle_bytes 8.06912e+06
go_memstats_heap_inuse_bytes 3.956736e+06
go_memstats_heap_objects 15749
go_memstats_heap_released_bytes 6.905856e+06
go_memstats_heap_sys_bytes 1.2025856e+07
go_memstats_last_gc_time_seconds 1.7808629752090907e+09
go_memstats_lookups_total 0
go_memstats_mallocs_total 4.7322589e+07
go_memstats_mcache_inuse_bytes 2400
go_memstats_mcache_sys_bytes 15600
go_memstats_mspan_inuse_bytes 64600
go_memstats_mspan_sys_bytes 81600
go_memstats_next_gc_bytes 4.194304e+06
go_memstats_other_sys_bytes 632531
go_memstats_stack_inuse_bytes 557056
go_memstats_stack_sys_bytes 557056
go_memstats_sys_bytes 2.0005896e+07
go_threads 9

```

```
node_boot_time_seconds 1.780854852e+09
node_context_switches_total 1.1732849e+07
node_cpu_core_throttles_total{core="0",package="0"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_guest_seconds_total{cpu="1",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="1",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 6532.28
node_cpu_seconds_total{cpu="0",mode="iowait"} 954.09
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 0.33
node_cpu_seconds_total{cpu="0",mode="softirq"} 1.35
node_cpu_seconds_total{cpu="0",mode="steal"} 0
node_cpu_seconds_total{cpu="0",mode="system"} 171.11
node_cpu_seconds_total{cpu="0",mode="user"} 429.13
node_cpu_seconds_total{cpu="1",mode="idle"} 6594.85
node_cpu_seconds_total{cpu="1",mode="iowait"} 889.77
node_cpu_seconds_total{cpu="1",mode="irq"} 0
node_cpu_seconds_total{cpu="1",mode="nice"} 0.26
node_cpu_seconds_total{cpu="1",mode="softirq"} 51.78
node_cpu_seconds_total{cpu="1",mode="steal"} 0
node_cpu_seconds_total{cpu="1",mode="system"} 159.57
node_cpu_seconds_total{cpu="1",mode="user"} 381.5
node_disk_discard_time_seconds_total{device="sda"} 0
node_disk_discard_time_seconds_total{device="sda1"} 0
node_disk_discard_time_seconds_total{device="sda2"} 0
node_disk_discard_time_seconds_total{device="sda3"} 0
node_disk_discard_time_seconds_total{device="sda99"} 0
node_disk_discard_time_seconds_total{device="sdb"} 0
node_disk_discarded_sectors_total{device="sda"} 0
node_disk_discarded_sectors_total{device="sda1"} 0
node_disk_discarded_sectors_total{device="sda2"} 0
node_disk_discarded_sectors_total{device="sda3"} 0
node_disk_discarded_sectors_total{device="sda99"} 0
node_disk_discarded_sectors_total{device="sdb"} 0
node_disk_discards_completed_total{device="sda"} 0
node_disk_discards_completed_total{device="sda1"} 0
node_disk_discards_completed_total{device="sda2"} 0
node_disk_discards_completed_total{device="sda3"} 0
```

```
node_disk_discards_completed_total{device="sda99"} 0
node_disk_discards_completed_total{device="sdb"} 0
node_disk_discards_merged_total{device="sda"} 0
node_disk_discards_merged_total{device="sda1"} 0
node_disk_discards_merged_total{device="sda2"} 0
node_disk_discards_merged_total{device="sda3"} 0
node_disk_discards_merged_total{device="sda99"} 0
node_disk_discards_merged_total{device="sdb"} 0
node_disk_flush_requests_time_seconds_total{device="sda"} 753.503
node_disk_flush_requests_time_seconds_total{device="sda1"} 0
node_disk_flush_requests_time_seconds_total{device="sda2"} 0
node_disk_flush_requests_time_seconds_total{device="sda3"} 0
node_disk_flush_requests_time_seconds_total{device="sda99"} 0
node_disk_flush_requests_time_seconds_total{device="sdb"} 0.021
node_disk_flush_requests_total{device="sda"} 40816
node_disk_flush_requests_total{device="sda1"} 0
node_disk_flush_requests_total{device="sda2"} 0
node_disk_flush_requests_total{device="sda3"} 0
node_disk_flush_requests_total{device="sda99"} 0
node_disk_flush_requests_total{device="sdb"} 4
node_disk_info{device="sda",major="8",minor="0"} 1
node_disk_info{device="sda1",major="8",minor="1"} 1
node_disk_info{device="sda2",major="8",minor="2"} 1
node_disk_info{device="sda3",major="8",minor="3"} 1
node_disk_info{device="sda99",major="259",minor="0"} 1
node_disk_info{device="sdb",major="8",minor="16"} 1
node_disk_io_now{device="sda"} 0
node_disk_io_now{device="sda1"} 0
node_disk_io_now{device="sda2"} 0
node_disk_io_now{device="sda3"} 0
node_disk_io_now{device="sda99"} 0
node_disk_io_now{device="sdb"} 0
node_disk_io_time_seconds_total{device="sda"} 1630.088
node_disk_io_time_seconds_total{device="sda1"} 1.028
node_disk_io_time_seconds_total{device="sda2"} 1.052
node_disk_io_time_seconds_total{device="sda3"} 1620.936
node_disk_io_time_seconds_total{device="sda99"} 57.008
node_disk_io_time_seconds_total{device="sdb"} 6.104
node_disk_io_time_weighted_seconds_total{device="sda"} 10179.406
```

```
node_disk_io_time_weighted_seconds_total{device="sda1"} 1.295
node_disk_io_time_weighted_seconds_total{device="sda2"} 1.044
node_disk_io_time_weighted_seconds_total{device="sda3"} 9087.464
node_disk_io_time_weighted_seconds_total{device="sda99"} 321.551
node_disk_io_time_weighted_seconds_total{device="sdb"} 6.535
node_disk_read_bytes_total{device="sda"} 4.105780736e+09
node_disk_read_bytes_total{device="sda1"} 4.161536e+06
node_disk_read_bytes_total{device="sda2"} 217088
node_disk_read_bytes_total{device="sda3"} 4.080669696e+09
node_disk_read_bytes_total{device="sda99"} 1.60384e+07
node_disk_read_bytes_total{device="sdb"} 3.82976e+06
node_disk_read_time_seconds_total{device="sda"} 5689.821
node_disk_read_time_seconds_total{device="sda1"} 1.289
node_disk_read_time_seconds_total{device="sda2"} 0.93900000000000001
node_disk_read_time_seconds_total{device="sda3"} 5652.715
node_disk_read_time_seconds_total{device="sda99"} 20.443
node_disk_read_time_seconds_total{device="sdb"} 6.492
node_disk_reads_completed_total{device="sda"} 199423
node_disk_reads_completed_total{device="sda1"} 108
node_disk_reads_completed_total{device="sda2"} 53
node_disk_reads_completed_total{device="sda3"} 197643
node_disk_reads_completed_total{device="sda99"} 1087
node_disk_reads_completed_total{device="sdb"} 465
node_disk_reads_merged_total{device="sda"} 6022
node_disk_reads_merged_total{device="sda1"} 0
node_disk_reads_merged_total{device="sda2"} 0
node_disk_reads_merged_total{device="sda3"} 4503
node_disk_reads_merged_total{device="sda99"} 1519
node_disk_reads_merged_total{device="sdb"} 0
node_disk_write_time_seconds_total{device="sda"} 3736.082
node_disk_write_time_seconds_total{device="sda1"} 0.005
node_disk_write_time_seconds_total{device="sda2"} 0.105
node_disk_write_time_seconds_total{device="sda3"} 3434.7490000000003
node_disk_write_time_seconds_total{device="sda99"} 301.107
node_disk_write_time_seconds_total{device="sdb"} 0.021
node_disk_writes_completed_total{device="sda"} 191446
node_disk_writes_completed_total{device="sda1"} 3
node_disk_writes_completed_total{device="sda2"} 4
node_disk_writes_completed_total{device="sda3"} 189546
```

```
node_disk_writes_completed_total{device="sda99"} 1885
node_disk_writes_completed_total{device="sdb"} 4
node_disk_writes_merged_total{device="sda"} 269559
node_disk_writes_merged_total{device="sda1"} 0
node_disk_writes_merged_total{device="sda2"} 0
node_disk_writes_merged_total{device="sda3"} 204743
node_disk_writes_merged_total{device="sda99"} 64809
node_disk_writes_merged_total{device="sdb"} 0
node_disk_written_bytes_total{device="sda"} 3.783438848e+09
node_disk_written_bytes_total{device="sda1"} 512
node_disk_written_bytes_total{device="sda2"} 0
node_disk_written_bytes_total{device="sda3"} 3.51025152e+09
node_disk_written_bytes_total{device="sda99"} 2.73145856e+08
node_disk_written_bytes_total{device="sdb"} 0
node_exporter_build_info{branch="debian/sid",goversion="go1.18.1",
  revision="1.3.1-1ubuntu0.22.04.3",version="1.3.1"} 1
node_filesystem_avail_bytes{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 2.57544192e+08
node_filesystem_avail_bytes{device="/dev/sda3",fstype="ext3",
  mountpoint="/"} 5.716631552e+10
node_filesystem_device_error{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 0
node_filesystem_device_error{device="/dev/sda3",
  fstype="ext3",mountpoint="/"} 0
node_filesystem_files{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 0
node_filesystem_files{device="/dev/sda3",fstype="ext3",
  mountpoint="/"} 4.194304e+06
node_filesystem_files_free{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 0
node_filesystem_files_free{device="/dev/sda3",fstype="ext3",
  mountpoint="/"} 4.08851e+06
node_filesystem_free_bytes{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 2.57544192e+08
node_filesystem_free_bytes{device="/dev/sda3",fstype="ext3",
  mountpoint="/"} 6.060228608e+10
node_filesystem_readonly{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 0
node_filesystem_readonly{device="/dev/sda3",fstype="ext3",
```

```
mountpoint="/" 0
node_filesystem_size_bytes{device="/dev/sda1",fstype="vfat",
  mountpoint="/boot/efi"} 2.68148736e+08
node_filesystem_size_bytes{device="/dev/sda3",fstype="ext3",
  mountpoint="/" 6.7050090496e+10
node_forks_total 12889
node_intr_total 6.859523e+06
node_load1 0.09
node_load15 7.27
node_load5 3.67
node_memory_Active_anon_bytes 7.1995392e+07
node_memory_Active_bytes 7.56973568e+08
node_memory_Active_file_bytes 6.84978176e+08
node_memory_AnonHugePages_bytes 0
node_memory_AnonPages_bytes 4.34573312e+08
node_memory_Bounce_bytes 0
node_memory_Buffers_bytes 4.6538752e+07
node_memory_Cached_bytes 1.319256064e+09
node_memory_CommitLimit_bytes 9.619488768e+09
node_memory_Committed_AS_bytes 1.276420096e+09
node_memory_DirectMap2M_bytes 1.996488704e+09
node_memory_DirectMap4k_bytes 1.507328e+08
node_memory_Dirty_bytes 20480
node_memory_FileHugePages_bytes 0
node_memory_FilePmdMapped_bytes 0
node_memory_HardwareCorrupted_bytes 0
node_memory_HugePages_Free 0
node_memory_HugePages_Rsvd 0
node_memory_HugePages_Surp 0
node_memory_HugePages_Total 0
node_memory_Hugepagesize_bytes 2.097152e+06
node_memory_Hugetlb_bytes 0
node_memory_Inactive_anon_bytes 3.6593664e+08
node_memory_Inactive_bytes 1.0461184e+09
node_memory_Inactive_file_bytes 6.8018176e+08
node_memory_KReclaimable_bytes 6.1526016e+07
node_memory_KernelStack_bytes 3.11296e+06
node_memory_Mapped_bytes 5.6283136e+07
node_memory_MemAvailable_bytes 1.364840448e+09
```

```
node_memory_MemFree_bytes 1.03436288e+08
node_memory_MemTotal_bytes 2.058330112e+09
node_memory_Mlocked_bytes 0
node_memory_NFS_Unstable_bytes 0
node_memory_PageTables_bytes 4.235264e+06
node_memory_Percpu_bytes 2.834432e+06
node_memory_SReclaimable_bytes 6.1526016e+07
node_memory_SUnreclaim_bytes 3.0253056e+07
node_memory_ShmemHugePages_bytes 0
node_memory_ShmemPmdMapped_bytes 0
node_memory_Shmem_bytes 634880
node_memory_Slab_bytes 9.1779072e+07
node_memory_SwapCached_bytes 6.422528e+06
node_memory_SwapFree_bytes 8.320864256e+09
node_memory_SwapTotal_bytes 8.590323712e+09
node_memory_Unevictable_bytes 0
node_memory_VmallocChunk_bytes 0
node_memory_VmallocTotal_bytes 3.5184372087808e+13
node_memory_VmallocUsed_bytes 1.7825792e+07
node_memory_WritebackTmp_bytes 0
node_memory_Writeback_bytes 0
node_network_receive_bytes_total{device="enp10s3f0"} 1.21223256e+08
node_network_receive_bytes_total{device="enp10s3f1"} 0
node_network_receive_bytes_total{device="enp6s7"} 5.1291575e+07
node_network_receive_bytes_total{device="enp7s8"} 0
node_network_receive_bytes_total{device="enp9s4f0"} 0
node_network_receive_bytes_total{device="enp9s4f1"} 0
node_network_receive_compressed_total{device="enp10s3f0"} 0
node_network_receive_compressed_total{device="enp10s3f1"} 0
node_network_receive_compressed_total{device="enp6s7"} 0
node_network_receive_compressed_total{device="enp7s8"} 0
node_network_receive_compressed_total{device="enp9s4f0"} 0
node_network_receive_compressed_total{device="enp9s4f1"} 0
node_network_receive_drop_total{device="enp10s3f0"} 0
node_network_receive_drop_total{device="enp10s3f1"} 0
node_network_receive_drop_total{device="enp6s7"} 4034
node_network_receive_drop_total{device="enp7s8"} 0
node_network_receive_drop_total{device="enp9s4f0"} 0
node_network_receive_drop_total{device="enp9s4f1"} 0
```

```
node_network_receive_errs_total{device="enp10s3f0"} 0
node_network_receive_errs_total{device="enp10s3f1"} 0
node_network_receive_errs_total{device="enp6s7"} 0
node_network_receive_errs_total{device="enp7s8"} 0
node_network_receive_errs_total{device="enp9s4f0"} 0
node_network_receive_errs_total{device="enp9s4f1"} 0
node_network_receive_fifo_total{device="enp10s3f0"} 0
node_network_receive_fifo_total{device="enp10s3f1"} 0
node_network_receive_fifo_total{device="enp6s7"} 0
node_network_receive_fifo_total{device="enp7s8"} 0
node_network_receive_fifo_total{device="enp9s4f0"} 0
node_network_receive_fifo_total{device="enp9s4f1"} 0
node_network_receive_frame_total{device="enp10s3f0"} 0
node_network_receive_frame_total{device="enp10s3f1"} 0
node_network_receive_frame_total{device="enp6s7"} 0
node_network_receive_frame_total{device="enp7s8"} 0
node_network_receive_frame_total{device="enp9s4f0"} 0
node_network_receive_frame_total{device="enp9s4f1"} 0
node_network_receive_multicast_total{device="enp10s3f0"} 0
node_network_receive_multicast_total{device="enp10s3f1"} 0
node_network_receive_multicast_total{device="enp6s7"} 4907
node_network_receive_multicast_total{device="enp7s8"} 0
node_network_receive_multicast_total{device="enp9s4f0"} 0
node_network_receive_multicast_total{device="enp9s4f1"} 0
node_network_receive_packets_total{device="enp10s3f0"} 894999
node_network_receive_packets_total{device="enp10s3f1"} 0
node_network_receive_packets_total{device="enp6s7"} 404993
node_network_receive_packets_total{device="enp7s8"} 0
node_network_receive_packets_total{device="enp9s4f0"} 0
node_network_receive_packets_total{device="enp9s4f1"} 0
node_network_transmit_bytes_total{device="enp10s3f0"} 7.91150986e+08
node_network_transmit_bytes_total{device="enp10s3f1"} 0
node_network_transmit_bytes_total{device="enp6s7"} 928815
node_network_transmit_bytes_total{device="enp7s8"} 0
node_network_transmit_bytes_total{device="enp9s4f0"} 0
node_network_transmit_bytes_total{device="enp9s4f1"} 0
node_network_transmit_carrier_total{device="enp10s3f0"} 0
node_network_transmit_carrier_total{device="enp10s3f1"} 0
node_network_transmit_carrier_total{device="enp6s7"} 0
```

```
node_network_transmit_carrier_total{device="enp7s8"} 0
node_network_transmit_carrier_total{device="enp9s4f0"} 0
node_network_transmit_carrier_total{device="enp9s4f1"} 0
node_network_transmit_colls_total{device="enp10s3f0"} 0
node_network_transmit_colls_total{device="enp10s3f1"} 0
node_network_transmit_colls_total{device="enp6s7"} 0
node_network_transmit_colls_total{device="enp7s8"} 0
node_network_transmit_colls_total{device="enp9s4f0"} 0
node_network_transmit_colls_total{device="enp9s4f1"} 0
node_network_transmit_compressed_total{device="enp10s3f0"} 0
node_network_transmit_compressed_total{device="enp10s3f1"} 0
node_network_transmit_compressed_total{device="enp6s7"} 0
node_network_transmit_compressed_total{device="enp7s8"} 0
node_network_transmit_compressed_total{device="enp9s4f0"} 0
node_network_transmit_compressed_total{device="enp9s4f1"} 0
node_network_transmit_drop_total{device="enp10s3f0"} 0
node_network_transmit_drop_total{device="enp10s3f1"} 0
node_network_transmit_drop_total{device="enp6s7"} 0
node_network_transmit_drop_total{device="enp7s8"} 0
node_network_transmit_drop_total{device="enp9s4f0"} 0
node_network_transmit_drop_total{device="enp9s4f1"} 0
node_network_transmit_errs_total{device="enp10s3f0"} 0
node_network_transmit_errs_total{device="enp10s3f1"} 0
node_network_transmit_errs_total{device="enp6s7"} 0
node_network_transmit_errs_total{device="enp7s8"} 0
node_network_transmit_errs_total{device="enp9s4f0"} 0
node_network_transmit_errs_total{device="enp9s4f1"} 0
node_network_transmit_fifo_total{device="enp10s3f0"} 0
node_network_transmit_fifo_total{device="enp10s3f1"} 0
node_network_transmit_fifo_total{device="enp6s7"} 0
node_network_transmit_fifo_total{device="enp7s8"} 0
node_network_transmit_fifo_total{device="enp9s4f0"} 0
node_network_transmit_fifo_total{device="enp9s4f1"} 0
node_network_transmit_packets_total{device="enp10s3f0"} 1.083986e+06
node_network_transmit_packets_total{device="enp10s3f1"} 0
node_network_transmit_packets_total{device="enp6s7"} 10256
node_network_transmit_packets_total{device="enp7s8"} 0
node_network_transmit_packets_total{device="enp9s4f0"} 0
node_network_transmit_packets_total{device="enp9s4f1"} 0
```

```
node_procs_blocked 0
node_procs_running 3
node_scrape_collector_duration_seconds{collector="cpu"} 0.001231755
node_scrape_collector_duration_seconds{collector="diskstats"} 0.009383697
node_scrape_collector_duration_seconds{collector="filesystem"} 0.001067998
node_scrape_collector_duration_seconds{collector="loadavg"} 0.000198331
node_scrape_collector_duration_seconds{collector="meminfo"} 0.001032294
node_scrape_collector_duration_seconds{collector="netdev"} 0.001680836
node_scrape_collector_duration_seconds{collector="stat"} 0.000550029
node_scrape_collector_duration_seconds{collector="vmstat"} 0.000727232
node_scrape_collector_success{collector="cpu"} 1
node_scrape_collector_success{collector="diskstats"} 1
node_scrape_collector_success{collector="filesystem"} 1
node_scrape_collector_success{collector="loadavg"} 1
node_scrape_collector_success{collector="meminfo"} 1
node_scrape_collector_success{collector="netdev"} 1
node_scrape_collector_success{collector="stat"} 1
node_scrape_collector_success{collector="vmstat"} 1
node_vmstat_oom_kill 0
node_vmstat_pgfault 3.025441e+06
node_vmstat_pgmajfault 3397
node_vmstat_pgpgin 4.013291e+06
node_vmstat_ppggout 3.694764e+06
node_vmstat_pswpin 2467
node_vmstat_pswpout 66685
process_cpu_seconds_total 169.57
process_max_fds 1024
process_open_fds 11
process_resident_memory_bytes 1.4651392e+07
process_start_time_seconds 1.78085744186e+09
process_virtual_memory_bytes 1.264771072e+09
process_virtual_memory_max_bytes 1.8446744073709552e+19
promhttp_metric_handler_errors_total{cause="encoding"} 0
promhttp_metric_handler_errors_total{cause="gathering"} 0
promhttp_metric_handler_requests_in_flight 1
promhttp_metric_handler_requests_total{code="200"} 5529
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

Fonte: Elaborado pela autora.

APÊNDICE B – CONFIGURAÇÃO DO `PMLOGGER` EQUIVALENTE AO *NODE EXPORTER*

O arquivo abaixo contém a configuração do `pmlogger` utilizada nos experimentos para obter um conjunto de métricas semanticamente equivalente ao coletado pelo *Node Exporter* do Prometheus em sua configuração restrita. As métricas estão organizadas por subsistema e mapeadas conforme o Quadro 2.

```
log mandatory on 1 second {
    # CPU utilization - node_cpu_seconds_total (per-mode aggregate + per-CPU)
    kernel.cpu.util.user
    kernel.cpu.util.nice
    kernel.cpu.util.sys
    kernel.cpu.util.idle
    kernel.cpu.util.wait
    kernel.cpu.util.steal
    kernel.cpu.util.intr
    kernel.percpu.cpu.user
    kernel.percpu.cpu.nice
    kernel.percpu.cpu.sys
    kernel.percpu.cpu.idle
    kernel.percpu.cpu.wait
    kernel.percpu.cpu.steal
    kernel.percpu.cpu.intr

    # System load and scheduling - node_load*, node_context_switches_total,
    # node_procs_*
    kernel.all.load
    kernel.all.pswitch
    kernel.all.intr
    kernel.all.running
    kernel.all.blocked

    # Memory - node_memory_* (matches /proc/meminfo fields exported by
    # node_exporter meminfo collector)
    mem.util.used
    mem.util.free
    mem.util.available
    mem.util.shared
    mem.util.bufmem
```

```
mem.util.cached
mem.util.dirty
mem.util.writeback
mem.util.active
mem.util.inactive
mem.util.swapTotal
mem.util.swapFree
mem.util.swapCached
mem.util.slab
mem.util.mapped
mem.util.committed_AS
mem.util.pageTables
mem.util.anonpages

# VM / page activity - node_vmstat_* (matches /proc/vmstat fields)
mem.vmstat.pgpgin
mem.vmstat.pgpgout
mem.vmstat.pgfault
mem.vmstat.pgmajfault
mem.vmstat.pgfree
mem.vmstat.pswpin
mem.vmstat.pswpout
swap.pagesin
swap.pagesout

# Disk I/O per-device only - node_disk_* (node_exporter diskstats
# collector, no aggregates)
disk.dev.read
disk.dev.write
disk.dev.read_bytes
disk.dev.write_bytes
disk.dev.read_merge
disk.dev.write_merge
disk.dev.avactive
disk.dev.aveq
disk.dev.read_rawactive
disk.dev.write_rawactive

# Network (per-interface) - node_network_*
```

```
network.interface.in.bytes
network.interface.out.bytes
network.interface.in.packets
network.interface.out.packets
network.interface.in.errors
network.interface.out.errors
network.interface.in.drops
network.interface.out.drops

# Filesystem - node_filesystem_*
filesystem.used
filesystem.free
filesystem.capacity
filesystem.avail
}
```

Fonte: Elaborado pela autora.

APÊNDICE C – MÉTRICAS DO DO PMLOGGER

O arquivo abaixo contém métricas do pmlogger obtidas com `pminfo -a /var/log/pcp/pmlogger/pmstat.giulia-307663.scalablesmr.emulab.net/20260608.00.10 -f > /tmp/pmstat-metrics` e mapeadas conforme o Quadro 2.

```
swap.pagesin  
  value 540
```

```
swap.pagesout  
  value 61199
```

```
disk.dev.read  
  inst [1 or "sdb"] value 569  
  inst [2 or "sda"] value 185153
```

```
disk.dev.write  
  inst [1 or "sdb"] value 4  
  inst [2 or "sda"] value 238257
```

```
disk.dev.read_bytes  
  inst [1 or "sdb"] value 3740  
  inst [2 or "sda"] value 3707914
```

```
disk.dev.write_bytes  
  inst [1 or "sdb"] value 0  
  inst [2 or "sda"] value 3481376
```

```
disk.dev.avactive  
  inst [1 or "sdb"] value 3976  
  inst [2 or "sda"] value 1973736
```

```
disk.dev.aveq  
  inst [1 or "sdb"] value 4376  
  inst [2 or "sda"] value 10271784
```

```
disk.dev.read_merge  
  inst [1 or "sdb"] value 0  
  inst [2 or "sda"] value 5232
```

```
disk.dev.write_merge
    inst [1 or "sdb"] value 0
    inst [2 or "sda"] value 236143

disk.dev.read_rawactive
    inst [1 or "sdb"] value 4327
    inst [2 or "sda"] value 5301885

disk.dev.write_rawactive
    inst [1 or "sdb"] value 24
    inst [2 or "sda"] value 4056744
disk.dev.await: pmLookupDesc: Derived metric definition failed

disk.dev.r_await
No value(s) available!

disk.dev.w_await
No value(s) available!
disk.dev.d_await: pmLookupDesc: Derived metric definition failed
disk.dev.f_await: pmLookupDesc: Derived metric definition failed

disk.dev.avg_qlen
No value(s) available!

disk.dev.avg_rqsz
No value(s) available!

disk.dev.r_avg_rqsz
No value(s) available!

disk.dev.w_avg_rqsz
No value(s) available!
disk.dev.d_avg_rqsz: pmLookupDesc: Derived metric definition failed

disk.dev.util
No value(s) available!
disk.dm.await: pmLookupDesc: Derived metric definition failed
disk.dm.r_await: pmLookupDesc: Derived metric definition failed
disk.dm.w_await: pmLookupDesc: Derived metric definition failed
```

disk.dm.d_await: pmLookupDesc: Derived metric definition failed
disk.dm.f_await: pmLookupDesc: Derived metric definition failed
disk.dm.avg_qlen: pmLookupDesc: Derived metric definition failed
disk.dm.avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.dm.r_avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.dm.w_avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.dm.d_avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.dm.util: pmLookupDesc: Derived metric definition failed
disk.md.await: pmLookupDesc: Derived metric definition failed
disk.md.r_await: pmLookupDesc: Derived metric definition failed
disk.md.w_await: pmLookupDesc: Derived metric definition failed
disk.md.d_await: pmLookupDesc: Derived metric definition failed
disk.md.f_await: pmLookupDesc: Derived metric definition failed
disk.md.avg_qlen: pmLookupDesc: Derived metric definition failed
disk.md.avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.md.r_avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.md.w_avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.md.d_avg_rqsiz: pmLookupDesc: Derived metric definition failed
disk.md.util: pmLookupDesc: Derived metric definition failed

network.interface.in.bytes

inst [0 or "lo"] value 2504882
inst [3 or "enp6s7"] value 151388281
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 121219005
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0

network.interface.in.packets

inst [0 or "lo"] value 48190
inst [3 or "enp6s7"] value 2411216
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 895481
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0

network.interface.in.errors

```
inst [0 or "lo"] value 0
inst [3 or "enp6s7"] value 0
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 0
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0
```

`network.interface.in.drops`

```
inst [0 or "lo"] value 0
inst [3 or "enp6s7"] value 22010
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 0
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0
```

`network.interface.out.bytes`

```
inst [0 or "lo"] value 2504882
inst [3 or "enp6s7"] value 770060
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 791777584
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0
```

`network.interface.out.packets`

```
inst [0 or "lo"] value 48190
inst [3 or "enp6s7"] value 6792
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 1079573
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0
```

`network.interface.out.errors`

```
inst [0 or "lo"] value 0
inst [3 or "enp6s7"] value 0
inst [4 or "enp7s8"] value 0
```

```
inst [5 or "enp9s4f0"] value 0
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0
```

network.interface.out.drops

```
inst [0 or "lo"] value 0
inst [3 or "enp6s7"] value 0
inst [4 or "enp7s8"] value 0
inst [5 or "enp9s4f0"] value 0
inst [6 or "enp9s4f1"] value 0
inst [7 or "enp10s3f0"] value 0
inst [8 or "enp10s3f1"] value 0
```

filesystem.capacity

```
inst [0 or "/dev/sda3"] value 65478604
inst [1 or "/dev/sda1"] value 261864
```

filesystem.used

```
inst [0 or "/dev/sda3"] value 6051684
inst [1 or "/dev/sda1"] value 10356
```

filesystem.free

```
inst [0 or "/dev/sda3"] value 59426920
inst [1 or "/dev/sda1"] value 251508
```

filesystem.avail

```
inst [0 or "/dev/sda3"] value 56071480
inst [1 or "/dev/sda1"] value 251508
```

mem.util.used

```
value 1933804
```

mem.util.free

```
value 76284
```

mem.util.shared

No value(s) available!

mem.util.bufmem
value 113092

mem.util.cached
value 1267400

mem.util.swapCached
value 2052

mem.util.active
value 744312

mem.util.inactive
value 1037548

mem.util.swapTotal
value 8388988

mem.util.swapFree
value 8145504

mem.util.dirty
value 224

mem.util.writeback
value 0

mem.util.mapped
value 36424

mem.util.slab
value 86008

mem.util.committed_AS
value 919436

mem.util.pageTables
value 3992

```
mem.util.anonpages
    value 399856

mem.util.available
    value 1353684

mem.vmstat.pgpgin
    value 3711654

mem.vmstat.pgpgout
    value 3481376

mem.vmstat.pswpin
    value 540

mem.vmstat.pswpout
    value 61199

mem.vmstat.pgfree
    value 6539129

mem.vmstat.pgfault
    value 3636997

mem.vmstat.pgmajfault
    value 2848

kernel.percpu.cpu.user
    inst [0 or "cpu0"] value 399380
    inst [1 or "cpu1"] value 357920

kernel.percpu.cpu.nice
    inst [0 or "cpu0"] value 1980
    inst [1 or "cpu1"] value 780

kernel.percpu.cpu.sys
    inst [0 or "cpu0"] value 242080
    inst [1 or "cpu1"] value 228440
```

kernel.percpu.cpu.idle

inst [0 or "cpu0"] value 42253530

inst [1 or "cpu1"] value 42348680

kernel.percpu.cpu.wait.total

inst [0 or "cpu0"] value 1172730

inst [1 or "cpu1"] value 1027880

kernel.percpu.cpu.intr

inst [0 or "cpu0"] value 270

inst [1 or "cpu1"] value 51840

kernel.percpu.cpu.steal

inst [0 or "cpu0"] value 0

inst [1 or "cpu1"] value 0

kernel.all.intr

value 14863042

kernel.all.pswitch

value 30052739

kernel.all.running

value 2

kernel.all.blocked

value 0

kernel.all.load

inst [1 or "1 minute"] value 0.75

inst [5 or "5 minute"] value 0.20999999

inst [15 or "15 minute"] value 0.05999999

kernel.cpu.util.user

No value(s) available!

kernel.cpu.util.nice

No value(s) available!

kernel.cpu.util.sys

No value(s) available!

kernel.cpu.util.idle

No value(s) available!

kernel.cpu.util.intr

No value(s) available!

kernel.cpu.util.wait

No value(s) available!

kernel.cpu.util.steal

No value(s) available!

pmcd.seqnum

value 12

pmcd.pid

value 1037

pmcd.pmlogger.archive

inst [1804 or "1804"] value "/var/log/pcp/pmlogger/pmstat.giulia-307663.scalabl

pmcd.pmlogger.port

inst [1804 or "1804"] value 4330

pmcd.pmlogger.host

inst [1804 or "1804"] value "pc207.emulab.net"

event.flags

No value(s) available!

event.missed

No value(s) available!

mssql.buffer_manager.buffer_cache_hit_ratio: pmLookupDesc: Derived metric definition

mssql.databases.log_cache_hit_ratio: pmLookupDesc: Derived metric definition failed

mssql.latches.average_latch_wait_time: pmLookupDesc: Derived metric definition fail

mssql.locks.average_wait_time: pmLookupDesc: Derived metric definition failed

mssql.plan_cache.cache_hit_ratio: pmLookupDesc: Derived metric definition failed
proc.psinfo.age: pmLookupDesc: Derived metric definition failed
proc.io.total_bytes: pmLookupDesc: Derived metric definition failed
proc.hog.cpu: pmLookupDesc: Derived metric definition failed
proc.hog.mem: pmLookupDesc: Derived metric definition failed
proc.hog.disk: pmLookupDesc: Derived metric definition failed
proc.hog.net: pmLookupDesc: Derived metric definition failed

Fonte: Elaborado pela autora.