



Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

Orama⁺⁺: Um Framework Orientado a Redes Neurais Artificiais para Predição de Tempo de Execução e Custos em Plataformas FaaS Multi-Nuvem

Leonardo Rebouças de Carvalho

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Orientadora

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo von Paumgarten

Brasília
2026

Ficha Catalográfica de Teses e Dissertações

Esta página existe apenas para indicar onde a ficha catalográfica gerada para dissertações de mestrado e teses de doutorado defendidas na UnB. A Biblioteca Central é responsável pela ficha, mais informações nos sítios:

<http://www.bce.unb.br>

<http://www.bce.unb.br/elaboracao-de-fichas-catalograficas-de-teses-e-dissertacoes>

Esta página não deve ser incluída na versão final do texto.



Orama⁺⁺: Um Framework Orientado a Redes Neurais Artificiais para Predição de Tempo de Execução e Custos em Plataformas FaaS Multi-Nuvem

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Prof. Dr. Paulo Sergio Lopes de Souza Prof.a Dr.a Lúcia Maria de Assumpção Drummond
ICMC/USP IC/UFF

Prof. Dr. Geraldo Pereira Rocha Filho
UnB/UESB

Prof.a Dr.a Cláudia Nalon
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 12 de maio de 2026

Dedicatória

Para Aurora.

Agradecimentos

Nenhum doutorado é construído em isolamento. Por mais que a escrita, a leitura e as reflexões muitas vezes aconteçam em silêncio, a jornada é sustentada por mãos invisíveis, palavras de incentivo e presenças que nos mantêm de pé quando o caminho parece incerto. Chegar até aqui é um exercício de reconhecer que este trabalho não é apenas meu, mas também de todos aqueles que contribuíram para que ele se tornasse possível.

À minha companheira de vida, Estela Barros Oliveira, expresso meu mais profundo e sincero agradecimento. Sua presença foi essencial em cada etapa desta trajetória, garantindo as condições necessárias para que eu chegasse até aqui. Você cuidou da nossa família de formas que vão além do que as palavras conseguem traduzir, trazendo serenidade aos momentos mais desafiadores e leveza aos dias mais difíceis. Sua presença em minha vida é mais do que apoio: é um verdadeiro talismã, que me acompanha, protege e me fortalece, trazendo sorte, equilíbrio e confiança para seguir em frente. Obrigado por compreender, com tanto amor e generosidade, minha “presença ausente” em tantos momentos, e ainda assim permanecer ao meu lado, firme e acolhedora. Este título também é seu, pois carrega, em cada conquista, a marca do seu amor.

À minha filha, Aurora, que chegou à minha vida enquanto esta jornada já estava em curso, deixo um agradecimento repleto de sentido. Sua presença ressignificou tudo e me deu ainda mais força para continuar. Um dos maiores motivos para persistir foi poder deixar para você um legado que valorize o conhecimento e a importância dos estudos. Que, ao ler estas palavras no futuro, você entenda: o conhecimento transforma caminhos e este trabalho é uma prova de que vale a pena persistir, mesmo quando parece impossível.

Aos meus pais, agradeço profundamente por todo o suporte e pelas condições que me proporcionaram ao longo da vida, permitindo que eu pudesse desenvolver e aproveitar ao máximo o meu potencial, mesmo diante das limitações. Este resultado é também fruto dos valores que vocês me ensinaram e dos esforços que fizeram por mim.

À minha orientadora, Aletéia Araújo, deixo um agradecimento especial. Sua presença ao longo desta jornada foi muito além da orientação acadêmica. Nos momentos mais críticos, encontrei não apenas direcionamento, mas também escuta, acolhimento e confiança. Mesmo quando os caminhos pareciam incertos, você esteve ali, firme, ajudando a

reorganizar as ideias, a recuperar o foco e a seguir em frente. Sua forma de conduzir este processo fez com que eu nunca me sentisse sozinho, e foi essencial para que eu conseguisse atravessar todas as etapas até aqui. Levo comigo não apenas os aprendizados acadêmicos, mas também o exemplo de como orientar, apoiar e acreditar nas pessoas.

Aos meus colegas de doutorado, Breno Costa e João Bachiega Jr., agradeço pela parceria construída ao longo dessa jornada. Formamos um time aguerrido, produtivo e, acima de tudo, verdadeiro. As inúmeras conversas densas, reflexivas e sinceras foram essenciais para manter a calma diante das adversidades e, nesse processo, ganhei grandes amigos para a vida.

Aos professores do PPGI da UnB, registro minha gratidão pela dedicação e pela excelência na transmissão de conhecimento. As disciplinas cursadas no início do doutorado foram fundamentais para a construção da base sólida que sustentou este trabalho. Em especial ao professor Geraldo Pereira Rocha Filho agradeço pelos ensinamentos relacionados à Inteligência Artificial.

À Cinara Fredo, agradeço por acreditar em mim de forma tão firme, mesmo quando eu próprio hesitava. Seu incentivo constante, a forma como comemorou cada conquista e a maneira como me fortaleceu nos momentos de maior fragilidade foram fundamentais para que eu seguisse em frente. Sua presença foi decisiva em muitos momentos desta caminhada, e levo comigo essa força que você sempre fez questão de me lembrar que eu tinha.

À Izabel Susuki, minha gratidão por sua presença constante e pelo cuidado genuíno ao longo de toda essa jornada. Seu interesse sincero, suas mensagens e sua atenção nos momentos certos fizeram toda a diferença. Saber que havia alguém acompanhando, se preocupando e torcendo por mim trouxe conforto e equilíbrio em muitos momentos difíceis.

Aos meus colegas de trabalho, Flávio Myiashiro, Diego Morandini, Oscar Marques, Kelvin Neves e Humbertho Mattar, meu sincero agradecimento pelo incentivo constante e por celebrarem comigo cada pequena vitória ao longo do caminho. Obrigado também por ouvirem, com paciência, meus momentos de frustração. O apoio de vocês fez toda a diferença.

Por fim, esta conquista reflete muito mais do que um esforço individual, sendo resultado de uma rede de apoio construída ao longo de toda a jornada. Muitas outras pessoas, de diferentes formas, contribuíram para que este momento fosse possível, seja com palavras, gestos, oportunidades ou simplesmente presença. A todas elas, mesmo que não estejam explicitamente mencionadas, deixo minha sincera gratidão.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

A computação *serverless* e, em particular, o modelo *Function-as-a-Service* (FaaS), têm ganhado popularidade devido à sua escalabilidade, flexibilidade e ao gerenciamento simplificado de infraestrutura. Entretanto, a avaliação de desempenho de aplicações em diferentes provedores de nuvem ainda apresenta desafios significativos, decorrentes da heterogeneidade das plataformas, das variações na alocação de recursos e da ausência de ferramentas padronizadas para experimentação sistemática. Este trabalho propõe o Orama⁺⁺, um *framework* orientado a redes neurais artificiais para predição do tempo de execução e dos custos de aplicações *FaaS* em ambientes multi-nuvem. Além disso, o Orama⁺⁺ possibilita a execução estruturada de *benchmarks* em plataformas *serverless*, abrangendo provedores tais como *Amazon Web Services* (AWS), *Google Cloud Platform* (GCP), Microsoft Azure e Alibaba Cloud. O sistema automatiza o provisionamento de cenários experimentais, suportando desde funções simples até aplicações que interagem com serviços de armazenamento e bancos de dados. O *framework* também orquestra execuções repetitivas para garantir reprodutibilidade experimental e aplica análises estatísticas baseadas em projeto *fatorial* 2^k e *testes t* pareados, permitindo avaliar o impacto de diferentes fatores no desempenho das aplicações. Com base nos dados coletados, modelos de redes neurais artificiais são treinados para estimar o tempo de execução e os custos de aplicações FaaS em diferentes configurações de provedores e tamanhos de entrada. Dessa forma, este trabalho contribui para apoiar o processo de tomada de decisão na escolha do uso de serviços FaaS, fornecendo um ambiente estruturado para experimentação, análise comparativa e predição de tempo de execução e custo em plataformas de nuvem pública.

Palavras-chave: Computação em Nuvem, *Serverless*, Funções como um Serviço, FaaS, *Framework* Orama⁺⁺, Predição de tempo de execução e custo, Redes Neurais Artificiais

Abstract

Serverless computing, particularly the FaaS model, has gained significant popularity due to its scalability, flexibility, and simplified infrastructure management. However, evaluating the performance of applications across different cloud providers remains challenging due to platform heterogeneity, variations in resource allocation, and the lack of standardized tools for systematic experimentation. This work proposes Orama⁺⁺, a framework based on artificial neural networks for predicting the execution time and cost of *FaaS* applications in multi-cloud environments. Furthermore, Orama⁺⁺ facilitates the structured execution of benchmarks on serverless platforms, encompassing providers such as AWS, GCP, Microsoft Azure, and Alibaba Cloud. The system automates the provisioning of experimental scenarios, supporting workloads ranging from simple functions to applications that interact with storage services and databases. The framework also orchestrates repeated executions to ensure experimental reproducibility and applies statistical analyses based on a 2^k factorial design and paired *t*-tests to assess the impact of different factors on application performance. Based on the collected data, artificial neural network models are trained to estimate the execution time and cost of serverless applications under different provider configurations and input sizes. Thus, this work contributes to supporting decision-making in the selection of *FaaS* environments by providing a structured platform for experimentation, comparative analysis, and performance prediction across public cloud platforms.

Keywords: Cloud Computing, Serverless, Function-as-a-Service, FaaS, Orama⁺⁺ *Framework*, Execution time and cost prediction, Artificial Neural Networks

Sumário

1	Introdução	1
1.1	Problema de Pesquisa	4
1.2	Objetivos	5
1.3	Metodologia Geral desta Pesquisa	6
1.4	Contribuições deste Trabalho	7
1.4.1	Aprimoramentos Tecnológicos	8
1.4.2	Contribuições Científicas	9
1.4.3	Publicações Científicas	10
1.5	Organização deste Documento	10
2	Computação em Nuvem e Avaliação Experimental de FaaS	11
2.1	Computação em Nuvem	11
2.2	Modelos de Serviço em Nuvem	12
2.3	<i>Serverless</i> e <i>Function-as-a-Service</i> (FaaS)	14
2.4	<i>Benchmarks</i> e Análise Estatística	16
2.4.1	Avaliação de Ambientes Distribuídos	17
2.4.2	Métodos Estatísticos para Avaliar Desempenho	17
2.4.2.1	Projeto <i>Fatorial</i> 2^k	19
2.4.2.2	Teste <i>t</i> Pareado	19
2.5	Avaliação de Provedores em Ambientes FaaS	20
2.5.1	Calculadora Simples	22
2.5.2	<i>Application Programming Interface</i> (API) com Armazenamento em <i>Object Storage</i>	24
2.5.3	API com Armazenamento em <i>Database-as-a-Service</i> (DBaaS)	25
2.5.4	Alinhamento de Sequências Genéticas	26
2.6	Considerações Finais	28
3	Redes Neurais e Técnicas de Predição de Desempenho	29
3.1	Contextualização	29

3.2	Fundamentos de Inteligência Artificial e Aprendizado de Máquina	30
3.3	Redes Neurais para Predição	31
3.3.1	Avaliação de Modelos Preditivos	33
3.3.1.1	Função de Perda (<i>Loss</i>)	34
3.3.1.2	Análise Gráfica	34
3.3.2	Otimização de Modelos Preditivos	37
3.3.3	Composição do Treinamento	39
3.4	Tratamento dos Dados e Pré-processamento	39
3.4.1	Tratamento de Valores Ausentes	40
3.4.2	Categorização	41
3.4.3	Normalização	41
3.4.4	Tratamento de <i>Outliers</i>	42
3.4.5	Análise de Correlação	42
3.4.6	Redução de Escala	43
3.4.7	Remoção de <i>Spikes</i>	43
3.5	Considerações Finais	44
4	<i>Framework Orama⁺⁺</i>	45
4.1	Definição	45
4.2	Arquitetura	47
4.2.1	<i>Frontend</i>	48
4.2.2	<i>Backend</i>	48
4.2.3	<i>Database</i>	49
4.2.4	<i>Cloud Orchestrator</i>	50
4.2.5	<i>Benchmark</i>	51
4.2.6	<i>Remote Benchmark</i>	52
4.2.7	Apache Kafka	53
4.2.8	<i>Halsteader</i>	54
4.2.9	<i>Predictor</i>	56
4.3	Cálculo de Custo em Provedores de Nuvem	58
4.4	Fluxo de Trabalho	60
4.5	Casos de Uso Pré-configurados	64
4.5.1	Calculadora	64
4.5.2	API com Armazenamento em Objetos	67
4.5.3	API com Armazenamento em DBaaS	69
4.5.4	Alinhamento de Sequência Genética	72
4.5.5	Impacto no Esforço Experimental	74
4.5.6	<i>Overhead</i> Experimental do <i>Framework</i>	76

4.6	Análise Estatística	76
4.6.1	Projeto <i>Fatorial</i> 2^k	77
4.6.2	Teste t Pareado	77
4.7	Modelo de Redes Neurais	78
4.7.1	Construção do <i>Dataset</i>	78
4.7.2	Elaboração do Modelo	80
4.7.2.1	Pré-processamento	80
4.7.2.2	Modelagem	82
4.7.2.3	Otimização	84
4.7.2.4	Treinamento	85
4.8	Tecnologias Empregadas	85
4.8.1	JavaScript	86
4.8.2	Python	86
4.8.3	PostgreSQL	87
4.8.4	Terraform	87
4.8.5	Apache Kafka	87
4.8.6	Apache JMeter	88
4.8.7	<i>Docker</i>	88
4.9	Implantação do <i>Framework</i> Orama ⁺⁺	88
4.10	Evoluções do <i>Framework</i> Orama ⁺⁺	89
4.11	Limitações do <i>Framework</i> Orama ⁺⁺	90
4.11.1	Dependência de APIs e Especificidades de Provedores	90
4.11.2	Dificuldades na Incorporação de Novos Provedores	91
4.11.3	Acoplamento entre Componentes	91
4.11.4	Limitações na Cobertura de Cenários	92
4.11.5	Dependência de Configuração e Credenciais	92
4.12	Considerações Finais	93
5	Trabalhos Relacionados	94
5.1	Levantamento Bibliográfico	94
5.2	<i>Benchmarking</i> de Plataformas FaaS	96
5.3	Modelagem e Predição de Desempenho, Custo e Comportamento de Aplicações	99
5.4	Análise Comparativa	103
5.5	Considerações Finais	105
6	Avaliação do <i>Framework</i> Orama⁺⁺	106
6.1	Experimento 1	106

6.1.1	Análise dos Resultados do Experimento 1	108
6.2	Experimento 2	111
6.2.1	Análise dos Resultados do Experimento 2	113
6.3	Experimento 3	117
6.3.1	Análise dos Resultados do Experimento 3	119
6.4	Experimento 4	122
6.4.1	Análise dos Resultados do Experimento 4	123
6.5	Experimento 5	127
6.5.1	Análise dos Resultados do Experimento 5	128
6.6	Avaliação do Modelo de Redes Neurais Treinado Usando Informações dos Experimentos de 1 a 5	131
6.7	Experimento 6	134
6.8	Avaliação do Modelo de Redes Neurais Retreinado Após o Experimento 6 .	135
6.9	Predições de Tempo de Execução e Custos	140
6.10	Síntese dos Resultados e Evolução do Orama ⁺⁺	143
6.11	Considerações Finais	145
7	Conclusão	146
7.1	Desafios	147
7.2	Ameaças à Validade	150
7.3	Trabalhos Futuros	151
7.4	Produção Científica Derivada desta Tese	152
	Referências	155
	Anexo	169
I	Guia Rápido do <i>framework</i> Orama⁺⁺	170
II	Certificado de Registro de Programa de Computador - <i>framework</i> Orama⁺⁺	172
III	Certificado de Registro de Programa de Computador - <i>framework</i> Node2FaaS	174
IV	Artigo publicado na conferência CARLA/2021	176
V	Artigo publicado na conferência WSCAD/2021	178
VI	Artigo publicado na conferência ERAD-CO/2021	180

VII	Artigo publicado na conferência ERAD-CO/2021	182
VIII	Artigo publicado na conferência ERAD-CO/2021	184
IX	Artigo publicado na conferência ERAD-CO/2021	186
X	Artigo publicado no periódico ACM Computing Surveys/2022	188
XI	Artigo publicado na conferência CLOSER/2022	190
XII	Artigo publicado na conferência CLOSER/2022	192
XIII	Artigo publicado na conferência CLOSER/2022	194
XIV	Artigo publicado no periódico Computer Networks/2022	196
XV	Artigo publicado na conferência WEBIST/2022	198
XVI	Artigo publicado no periódico Wiley/CCPE/2022	200
XVII	Artigo publicado no periódico ACM Computing Surveys/2023	202
XVIII	Artigo publicado no periódico SNCS/2023	204
XIX	Artigo publicado na conferência CLOSER/2023	206
XX	Artigo publicado na conferência CISTI/2023	208
XXI	Artigo publicado no periódico IEEE Access/2023	210
XXII	Artigo publicado na conferência WSCAD/2023	212
XXIII	Artigo publicado no periódico SNCS/2024	214
XXIV	Artigo publicado na conferência MobiQuitous/2024	216
XXV	Artigo publicado como capítulo em CCSS/2024	218
XXVI	Artigo publicado na conferência UCC/2024	220
XXVII	Artigo publicado na conferência SSCAD/2025	222
XXVIII	Artigo publicado na conferência ERI-GO/2025	224
XXIX	Artigo publicado na conferência CLOSER/2026	226

Lista de Figuras

2.1	Exemplos de serviços em nuvem de diferentes modelos.	13
2.2	Fluxo de execução de FaaS.	15
3.1	Exemplo de <i>boxplots</i> para as métricas <i>Root Mean Squared Error</i> (RMSE), <i>Mean Squared Error</i> (MSE), R^2	35
3.2	Exemplo de gráfico comparativo entre previsões e observações reais.	36
4.1	Visão dos componentes do <i>framework</i> Orama ⁺⁺	47
4.2	Modelo de dados do <i>framework</i> Orama ⁺⁺	50
4.3	Exemplo de distribuição de requisições entre os <i>Remote Benchmarkers</i>	52
4.4	Arquitetura do <i>framework</i> Orama ⁺⁺	57
4.5	Exemplo de cálculo de custo a partir das estimativas de execução no Orama ⁺⁺	61
4.6	Fluxo de trabalho do <i>framework</i> Orama ⁺⁺	62
4.7	Calculadora no AWS Lambda.	66
4.8	Calculadora no <i>Google Cloud Functions</i> (GCF).	66
4.9	Calculadora no <i>Azure Functions</i> (AZF).	66
4.10	Calculadora no <i>Alibaba Functions Compute</i> (AFC).	66
4.11	API → <i>Simple Storage Service</i> (S3) no AWS Lambda.	69
4.12	API → <i>Google Cloud Storage</i> (GCS) no GCF.	69
4.13	API → <i>Azure Blob Storage</i> (ABS) no AZF.	69
4.14	API → <i>Object Storage Service</i> (OSS) no AFC.	69
4.15	API → <i>DynamoDB</i> no AWS Lambda.	71
4.16	API → <i>Firestore</i> no GCF.	71
4.17	API → <i>CosmosDB</i> no AZF.	71
4.18	Alinhamento de sequência genética.	74
4.19	Passos para a construção do <i>dataset</i>	79
4.20	Passos da elaboração do modelo de Redes Neurais.	80
4.21	Correlação entre os parâmetros do <i>dataset</i> inicial.	81
4.22	Correlação entre os parâmetros do <i>dataset</i> com <i>Principal Component Analysis</i> (PCA).	82

6.1	Experimento 1 - arquitetura dos testes.	107
6.2	Experimento 1 - resultado do caso de uso Calculadora.	109
6.3	Experimento 1 - resultado do caso de uso API com armazenamento em objetos.	109
6.4	Experimento 1 - resultado do caso de uso API com DBaaS.	110
6.5	Experimento 1 - resultado do projeto <i>fatorial 2^k</i>	110
6.6	Experimento 2 - arquitetura dos testes.	112
6.7	Experimento 2 - resultados do caso de uso Calculadora.	113
6.8	Experimento 2 - resultados do caso de uso API com armazenamento em objetos.	115
6.9	Experimento 2 - resultados do caso de uso API com armazenamento em DBaaS.	116
6.10	Experimento 2 - resultados do projeto <i>fatorial 2^k</i>	117
6.11	Experimento 3 - cenário com <i>workers</i> concentrados.	118
6.12	Experimento 3 - cenário com <i>workers</i> distribuídos.	118
6.13	Experimento 3 - resultados do caso de uso Calculadora.	120
6.14	Experimento 3 - requisições de <i>warm-up</i>	121
6.15	Experimentos 4 e 5 - arquitetura dos testes.	124
6.16	Experimento 4 - resultados das requisições de <i>warm-up</i>	124
6.17	Experimento 4 - resultados do caso de uso Calculadora em cinco regiões dos provedores.	125
6.18	Experimento 5 - resultados do caso de uso API com DBaaS em cinco regiões dos provedores.	129
6.19	<i>Boxplot</i> do RMSE.	132
6.20	<i>Boxplot</i> do MSE.	132
6.21	<i>Boxplot</i> do <i>Mean Absolute Percentage Error</i> (MAPE).	132
6.22	<i>Boxplot</i> do R^2	132
6.23	Função perda (<i>Loss</i>) no decorrer do treinamento.	133
6.24	Comparativo entre observações e previsões.	134
6.25	Resultados da métrica <i>Loss</i> nos modelos Dense, <i>Long Short-Term Memory</i> (LSTM) e <i>Bidirectional Long Short-Term Memory</i> (BLSTM).	136
6.26	Resultados da métrica <i>Mean Absolute Error</i> (MAE) nos modelos Dense, LSTM e BLSTM.	136
6.27	Resultados da métrica MSE nos modelos Dense, LSTM e BLSTM.	136
6.28	Resultados da métrica RMSE nos modelos Dense, LSTM e BLSTM.	137
6.29	Resultados da métrica MAPE nos modelos Dense, LSTM e BLSTM.	137
6.30	Resultados da métrica R^2 nos modelos Dense, LSTM e BLSTM.	137

6.31	Comparativo entre predições e observações.	138
------	--	-----

Lista de Tabelas

2.1	Comparação entre serviços de FaaS.	16
2.2	Estimativa de tempo para execução manual das atividades por provedor. . .	21
4.1	Exemplo de configuração de custos por provedor no Orama ⁺⁺	59
4.2	Estimativa de esforço manual por caso de uso e por provedor.	75
4.3	Estimativa de esforço com o uso do Orama ⁺⁺ por caso de uso (multi-nuvem). .	75
4.4	Parâmetros do modelo.	83
4.5	Evolução do <i>Framework</i> Orama ⁺⁺	90
5.1	Comparação das características dos trabalhos de <i>benchmarking</i> em FaaS. . .	99
5.2	Comparação entre abordagens de modelagem e de predição em ambientes FaaS.	103
5.3	Comparação entre os trabalhos relacionados e o <i>framework</i> Orama ⁺⁺	104
6.1	Experimento 1 - parâmetros dos cenários.	107
6.2	Resultados dos testes <i>t</i> pareados no Experimento 1.	111
6.3	Experimento 2 - resultados do teste <i>t</i> pareado para o caso de uso Calculadora. .	114
6.4	Experimento 3 - resultados do teste <i>t</i> pareado entre provedores.	122
6.5	Experimentos 4 e 5 - regiões dos provedores envolvidos.	123
6.6	Sequências genéticas utilizadas no Experimento 6, categorizadas por escala de entrada.	135
6.7	Estimativa de tempo de execução e custos para diferentes provedores e casos de uso.	141
7.1	Publicações adjacentes ao tema da tese.	152
7.2	Publicações diretamente relacionadas à tese	153

Lista de Abreviaturas e Siglas

ABS *Azure Blob Storage.*

AFC *Alibaba Functions Compute.*

AIaaS *Artificial Intelligence-as-a-Service.*

ANOVA *Analysis of Variance.*

API *Application Programming Interface.*

AWS *Amazon Web Services.*

AZF *Azure Functions.*

BLAST *Basic Local Alignment Search Tool.*

BLSTM *Bidirectional Long Short-Term Memory.*

CNN *Convolutional Neural Network.*

DBaaS *Database-as-a-Service.*

DNA *DeoxyriboNucleic Acid.*

FaaS *Function-as-a-Service.*

GAE *Google App Engine.*

GCF *Google Cloud Functions.*

GCP *Google Cloud Platform.*

GCS *Google Cloud Storage.*

HDF5 *Hierarchical Data Format version 5.*

HPC *High-Performance Computing.*

IA *Inteligência Artificial.*

IaaS *Infrastructure-as-a-Service.*

IaC *Infrastructure-as-Code.*

INPI *Instituto Nacional da Propriedade Industrial.*

IQR *Interquartile Range.*

LSTM *Long Short-Term Memory.*

MAE *Mean Absolute Error.*

MAPE *Mean Absolute Percentage Error.*

MLP *Multilayer Perceptron.*

MSE *Mean Squared Error.*

NoSQL *Not Only SQL.*

OSS *Object Storage Service.*

PaaS *Platform-as-a-Service.*

PCA *Principal Component Analysis.*

RMSE *Root Mean Squared Error.*

RNA *RiboNucleic Acid.*

S3 *Simple Storage Service.*

SaaS *Software-as-a-Service.*

SDK *Software Development Kit.*

SECaaS *Security-as-a-Service.*

SQL *Structured Query Language.*

SSL *Secure Sockets Layer.*

STaaS *Storage-as-a-Service.*

SVM *Support Vector Machines.*

vCPU *Virtual Central Processing Unit.*

VNET *Virtual Network.*

VPC *Virtual Private Cloud.*

WMA *Weighted Moving Average.*

XaaS *Everything-as-a-Service.*

Capítulo 1

Introdução

A Computação em Nuvem transformou a forma como aplicações são desenvolvidas, implantadas e escaladas, oferecendo infraestrutura sob demanda e modelos de precificação baseados no consumo. Esse paradigma inclui diferentes modelos de serviço, tais como *Infrastructure-as-a-Service* (IaaS) [1], que disponibiliza recursos computacionais básicos como máquinas virtuais e armazenamento; *Platform-as-a-Service* (PaaS) [1], que oferece ambientes de desenvolvimento e execução de aplicações sem a necessidade de gerenciamento direto da infraestrutura; e *Software-as-a-Service* (SaaS) [1], que disponibiliza aplicações completas acessíveis pela Internet.

Dentro desse ecossistema, o paradigma *serverless* tem ganhado destaque ao abstrair ainda mais a gestão da infraestrutura [2]. Nesse modelo, os provedores de nuvem são responsáveis pelo provisionamento automático de recursos, permitindo que os desenvolvedores concentrem seus esforços exclusivamente na lógica da aplicação. Entre os serviços oferecidos nesse paradigma, destaca-se o modelo *Function-as-a-Service* (FaaS) [3], que permite a execução de funções sob demanda com escalabilidade automática e cobrança baseada no tempo efetivo de execução. Provedores como AWS [4], GCP [5], Microsoft Azure [6] e Alibaba Cloud [7] disponibilizam plataformas FaaS amplamente utilizadas na construção de aplicações modernas baseadas em microsserviços e arquiteturas orientadas a eventos.

Apesar das vantagens oferecidas por esse modelo, a adoção de serviços FaaS em cenários reais de produção impõe desafios relevantes. Desenvolvedores e arquitetos de sistemas frequentemente precisam escolher entre múltiplos provedores de nuvem para implantar suas aplicações, cada um apresentando diferentes características de infraestrutura, políticas de alocação de recursos, modelos de escalabilidade e estratégias de gerenciamento interno. Essas diferenças podem resultar em variações significativas no tempo de execução das funções e, conseqüentemente, no custo total da aplicação. Como o modelo de cobrança em FaaS é diretamente proporcional ao tempo de execução e à quantidade de recursos

utilizados, decisões inadequadas na escolha do provedor ou na configuração das funções podem impactar negativamente tanto o desempenho quanto os custos operacionais das aplicações.

Na prática, o processo de seleção de provedores e configuração de funções em ambientes FaaS costuma ser empírico e baseado em experimentação manual. Desenvolvedores frequentemente precisam implantar suas aplicações em diferentes plataformas, executar testes repetidos e analisar os resultados para compreender como o comportamento da aplicação varia entre provedores. Esse processo é trabalhoso, pouco escalável e frequentemente não reproduzível. Além disso, a ausência de ferramentas padronizadas para experimentação e *benchmarking* dificulta a comparação objetiva entre plataformas e limita a capacidade da comunidade científica de conduzir estudos sistemáticos sobre desempenho em ambientes *serverless*.

Diversos trabalhos recentes têm investigado o uso de técnicas de Aprendizado de Máquina para modelar ou prever o comportamento de aplicações FaaS. Abordagens como o FaaSTest [8] utilizam Aprendizado de Máquina para selecionar plataformas em tempo de execução com base em padrões históricos de comportamento dos serviços. Outras propostas, tal como Sizeless [9], exploram regressão multi-alvo para estimar tempos de execução sob diferentes configurações de memória utilizando dados de monitoramento. Mais recentemente, *frameworks* como SLOPE [10] aplicam Redes Neurais para estimar necessidades de escalabilidade em aplicações *serverless*, enquanto abordagens como ML-FaaS [11] utilizam modelos de Aprendizado de Máquina para detectar condições de sobrecarga em plataformas FaaS. Apesar dos avanços, muitas dessas soluções dependem fortemente de dados de monitoramento obtidos após a implantação das aplicações, ou focam predominantemente em otimização de recursos e gerenciamento de infraestrutura, não abordando diretamente o problema de predição de tempo de execução e custos em cenários multi-nuvem com base em características das aplicações.

Nesse contexto, a predição do tempo de execução e de estimativas de custo de funções torna-se um elemento fundamental para apoiar o processo de tomada de decisão em ambientes FaaS. A capacidade de estimar antecipadamente o comportamento de uma aplicação em diferentes provedores permite que desenvolvedores escolham configurações mais adequadas em termos de latência, escalabilidade e custo operacional. Além disso, como o modelo de cobrança está diretamente relacionado ao tempo de execução das funções, a predição desse tempo possibilita estimativas mais precisas de custos, antes da implantação das aplicações.

A construção de modelos preditivos confiáveis depende da disponibilidade de dados experimentais representativos e da aplicação de técnicas de análise apropriadas. Nesse cenário, o uso de *benchmarks* sistemáticos se torna essencial para coletar informações

sobre o comportamento das funções em diferentes provedores e configurações. Esses dados podem, então, ser utilizados para treinar modelos de Aprendizado de Máquina capazes de capturar padrões de execução e prever o desempenho das aplicações em novos cenários.

Neste trabalho, técnicas de Inteligência Artificial (IA) são utilizadas para construir modelos preditivos capazes de estimar o tempo de execução de aplicações FaaS em diferentes provedores de nuvem. Em particular, são empregadas Redes Neurais artificiais que possuem elevada capacidade de modelar relações não lineares, e capturar interações complexas entre variáveis associadas às características das aplicações e às configurações de execução. A partir das estimativas de tempo de execução produzidas pelos modelos, também são calculadas estimativas de custo para a execução das funções nos provedores suportados pelo *framework*, incluindo AWS, GCP, Microsoft Azure e Alibaba Cloud. Esse cálculo considera os principais componentes que compõem o modelo de precificação de plataformas FaaS, tais como o custo fixo por acionamento da função, o valor associado ao tempo efetivo de execução e a quantidade de recursos computacionais alocados, tais como memória e capacidade de processamento.

Para viabilizar esse processo, este trabalho propõe o Orama⁺⁺, um *framework* para execução padronizada de experimentos, coleta de dados de desempenho e construção de modelos preditivos em ambientes FaaS multi-nuvem. O Orama⁺⁺ automatiza o provisionamento de cenários experimentais, a execução repetitiva de *benchmarks* e a coleta sistemática de dados, permitindo a realização de estudos reproduzíveis sobre desempenho em plataformas *serverless*. Além disso, o *framework* incorpora mecanismos de análise estatística baseados em projeto *fatorial* 2^k [12] e testes *t* pareados [13], possibilitando a identificação de fatores que impactam o desempenho das aplicações.

Assim, a partir dos dados coletados, modelos preditivos são treinados para estimar o tempo de execução de funções em diferentes provedores de nuvem com base em características extraídas do código-fonte das funções que o desenvolvedor pretende implantar. Com essas estimativas, o Orama⁺⁺ também calcula projeções de custo considerando os modelos de precificação das plataformas suportadas, permitindo comparar cenários de execução em múltiplos provedores. Dessa forma, o *framework*, por meio de técnicas de Inteligência Artificial (IA), é capaz de estimar o tempo de execução e o custo de uma função em diferentes provedores e sob distintos níveis de concorrência. Isso possibilita que desenvolvedores e arquitetos de sistemas antecipem o comportamento de suas aplicações e identifiquem, ainda antes da implantação, qual provedor FaaS tende a oferecer o melhor desempenho aliado ao menor custo de provisionamento. O Orama⁺⁺ está registrado no Instituto Nacional da Propriedade Industrial (INPI) como “Programa de Computador” sob nº BR512026002022-0 (Anexo II), assim como o *framework* Node2FaaS [14], que é uma contribuição adjacente a esta tese, sob nº BR512026002212-6 (Anexo III).

1.1 Problema de Pesquisa

O modelo FaaS oferece vantagens significativas em termos de escalabilidade e simplicidade operacional. Todavia, o comportamento de aplicações pode variar consideravelmente entre diferentes provedores de nuvem. Essas variações decorrem de diferenças nas arquiteturas internas das plataformas, nas políticas de gerenciamento de recursos e nos mecanismos de escalonamento adotados por cada provedor. Como consequência, funções com implementações idênticas podem apresentar tempos de execução distintos quando implantadas em plataformas diferentes, impactando diretamente o desempenho e os custos de execução.

Apesar da relevância desse problema, a avaliação comparativa entre provedores FaaS ainda é limitada pela ausência de ferramentas padronizadas que permitam conduzir experimentos reproduzíveis (podendo ser executados novamente, sob as mesmas condições e parâmetros, produzindo resultados consistentes e comparáveis), coletar dados de forma sistemática e analisar os resultados de maneira estatisticamente fundamentada. Além disso, grande parte das abordagens existentes depende de dados de execução obtidos após a implantação das aplicações, o que reduz sua utilidade em cenários nos quais desenvolvedores precisam estimar previamente o comportamento de suas funções antes de decidir onde implantá-las.

Diante desse cenário, surge a necessidade de métodos e ferramentas capazes de apoiar a análise comparativa entre plataformas FaaS, e de construir modelos preditivos que permitam estimar o tempo de execução de funções em diferentes provedores. A disponibilidade de tais mecanismos pode contribuir para estimativas mais precisas de custos e para decisões mais informadas no processo de seleção de provedores em ambientes multi-nuvem.

Existem trabalhos voltados para modelagem de desempenho, otimização de recursos e seleção dinâmica de plataformas FaaS. Entretanto, a literatura ainda carece de abordagens integradas que combinem: (i) execução padronizada e relativamente reproduzível¹ de *benchmarks* em ambientes multi-nuvem; (ii) coleta sistemática de dados experimentais em diferentes provedores; e (iii) modelos preditivos capazes de estimar tempo de execução e custos antes da implantação das aplicações.

Essa lacuna está associada à complexidade dos ambientes de Computação em Nuvem, marcados por fatores como variabilidade de carga, *cold start*, heterogeneidade de hardware e diferenças arquiteturais entre provedores. Como consequência, muitas soluções ainda dependem de métricas obtidas apenas após a implantação das aplicações, reduzindo sua utilidade em cenários de planejamento antecipado.

Assim, a questão de pesquisa investigada nesta tese se resume em “é possível desenvolver um *framework* reproduzível capaz de executar *benchmarks* multi-nuvem em FaaS e,

¹Considerando as evoluções dos provedores, a capacidade de reprodução pode se degradar com o tempo, demandando atualizações nos artefatos e métodos de avaliação.

simultaneamente, prever tempo de execução e custo com precisão suficiente para apoiar a tomada de decisão?”. Essa questão central se subdivide nas seguintes questões:

- “Como reduzir a variabilidade experimental?”;
- “Como modelar o comportamento heterogêneo de diferentes provedores de nuvem?”;
- “Como generalizar a predição de tempo e custo entre os diferentes serviços de FaaS?”.

1.2 Objetivos

Esta pesquisa tem como objetivo geral desenvolver um *framework* para apoiar a análise comparativa e a seleção de provedores FaaS em ambientes multi-nuvem, por meio da execução estruturada de *benchmarks*, da aplicação de métodos estatísticos e do uso de técnicas de Aprendizado de Máquina para predição de tempo de execução e estimativa de custos de aplicações *serverless*. Para atingir esse objetivo geral, os seguintes objetivos específicos são definidos:

- Desenvolver um módulo de provisionamento automatizado de cenários experimentais (aplicações) em diferentes provedores FaaS, suportando desde funções simples até aplicações que interagem com serviços de armazenamento e bancos de dados, incluindo um caso de uso científico baseado em alinhamento de sequências genéticas;
- Projetar um mecanismo de orquestração para execução padronizada e reproduzível de *benchmarks* em ambientes FaaS multi-nuvem;
- Sistematizar a aplicação de métodos de análise estatística, incluindo projeto *fatorial* 2^k e testes *t* pareados, com vistas à identificação e análise de fatores que influenciam o desempenho das funções;
- Desenvolver modelos preditivos baseados em Redes Neurais artificiais capazes de estimar o tempo de execução de funções em diferentes provedores a partir de características extraídas do código-fonte e das configurações de execução;
- Implementar mecanismos para estimativa de custos de execução em plataformas FaaS, considerando os principais componentes do modelo de precificação, como custo por invocação, tempo de execução e quantidade de recursos computacionais alocados.

O conjunto de objetivos definidos busca estabelecer uma abordagem sistemática para a avaliação de desempenho em ambientes FaaS multi-nuvem, combinando experimentação

padronizada, análise estatística e modelagem preditiva baseada em IA. Por meio do desenvolvimento do *framework*, pretende-se oferecer uma infraestrutura que permita tanto a condução de estudos experimentais reproduzíveis, quanto a construção de modelos capazes de prever o tempo de execução e estimar os custos de aplicações *serverless*. Dessa forma, espera-se contribuir para apoiar decisões mais informadas no processo de seleção de provedores FaaS, reduzindo a necessidade de experimentação empírica e aumentando a previsibilidade do comportamento das aplicações antes de sua implantação.

1.3 Metodologia Geral desta Pesquisa

Esta pesquisa adotou uma abordagem experimental com o objetivo de investigar o comportamento de aplicações em ambientes FaaS multi-nuvem, e construir modelos capazes de prever seu desempenho e custo de execução. A metodologia combinou experimentação controlada, análise estatística e técnicas de IA para compreender os fatores que influenciaram o tempo de execução de funções e apoiar a comparação entre diferentes provedores de nuvem.

Inicialmente, foi realizada uma revisão bibliográfica sobre avaliação de desempenho em ambientes *serverless*, técnicas de *benchmarking* para plataformas FaaS e abordagens de modelagem preditiva aplicadas à Computação em Nuvem. Essa etapa permitiu identificar lacunas existentes na literatura e orientar o desenvolvimento da abordagem proposta neste trabalho.

Com base nesse levantamento, foi desenvolvido o Orama⁺⁺, um *framework* projetado para viabilizar a execução padronizada de experimentos em ambientes FaaS. O *framework* automatizou o provisionamento de cenários experimentais, a execução repetitiva de *benchmarks* e a coleta sistemática de métricas de desempenho em múltiplos provedores de nuvem.

Os experimentos foram conduzidos em diferentes plataformas FaaS, permitindo observar como funções com implementações idênticas se comportaram sob diferentes condições de execução e níveis de concorrência. Nesse contexto, concorrência refere-se à quantidade de invocações simultâneas de uma mesma função, o que impacta diretamente aspectos como escalabilidade, contenção de recursos e variação de latência. O Orama⁺⁺ possibilita a configuração explícita desse parâmetro, permitindo ao usuário definir o número de requisições executadas em paralelo durante a condução dos experimentos.

Durante a execução de um *benchmark* associado a um determinado caso de uso, o *framework* permitiu a parametrização de diferentes aspectos do experimento. Entre esses parâmetros, destacaram-se a possibilidade de realizar um disparo adicional de requisição com o objetivo de medir e isolar efeitos de *cold start*, a definição da quantidade de re-

quisições em paralelo, o intervalo entre requisições consecutivas e o número de repetições de cada bateria de testes. Essa flexibilidade permitiu a construção de cenários experimentais mais controlados e representativos, possibilitando análises mais robustas sobre o comportamento das aplicações em ambientes FaaS.

Durante essa etapa, foram coletadas métricas relacionadas ao tempo de execução das funções, parâmetros de configuração do ambiente e características extraídas do código-fonte das aplicações avaliadas, fornecendo a base necessária para análises estatísticas e para o treinamento dos modelos de predição propostos neste trabalho.

A partir dos dados coletados, foi construído um *dataset* experimental utilizado para o treinamento de modelos de IA baseados em Redes Neurais artificiais. Esses modelos foram empregados para prever o tempo de execução das funções em diferentes provedores de nuvem a partir de características das aplicações e dos parâmetros de execução. Com base nessas predições, também foram estimados os custos de execução nas plataformas suportadas, considerando os principais elementos do modelo de precificação de serviços FaaS.

Para garantir a validade dos resultados, os dados obtidos foram analisados utilizando métodos estatísticos, incluindo projeto *fatorial* 2^k e testes *t* pareados, permitindo identificar fatores que influenciaram o desempenho das aplicações. Por fim, os modelos de predição foram validados por meio da comparação entre os tempos de execução estimados e os valores observados nos experimentos reais.

Como resultado dessas etapas, estabeleceu-se uma abordagem metodológica estruturada para investigar o comportamento de aplicações FaaS em ambientes multi-nuvem e avaliar a capacidade do Orama⁺⁺ de apoiar decisões relacionadas à escolha de provedores com base em critérios de desempenho e custo.

1.4 Contribuições deste Trabalho

Esta pesquisa contribui para o avanço do estudo de ambientes FaaS por meio do desenvolvimento de artefatos tecnológicos, da realização de experimentos sistemáticos em múltiplos provedores de nuvem e da proposição de métodos para análise e predição de desempenho de aplicações *serverless*. As contribuições resultantes deste trabalho abrangem tanto aspectos tecnológicos quanto científicos, incluindo a criação de ferramentas experimentais, conjuntos de dados, modelos de predição e publicações científicas que consolidam os resultados obtidos ao longo da investigação.

As contribuições desta tese podem ser organizadas em três categorias principais: (i) aprimoramentos tecnológicos, relacionados ao desenvolvimento de ferramentas, artefatos experimentais e repositórios de pesquisa; (ii) contribuições científicas, associadas aos mé-

todos de análise, modelos propostos e resultados obtidos; e (iii) publicações científicas que disseminam e validam os achados desta pesquisa na comunidade acadêmica.

1.4.1 Aprimoramentos Tecnológicos

Um dos principais resultados desta pesquisa é o desenvolvimento de uma infraestrutura experimental capaz de automatizar a avaliação de ambientes FaaS em múltiplos provedores de nuvem. Essa infraestrutura é materializada no *framework* Orama⁺⁺, que integra diferentes subsistemas responsáveis pelo provisionamento de ambientes, execução de experimentos, coleta de métricas e análise de resultados. Assim, os principais avanços tecnológicos desenvolvidos neste trabalho incluem:

- **Artefatos de automação para provisionamento de casos de uso equivalentes em diferentes provedores de nuvem**, permitindo a implantação padronizada de aplicações em plataformas como AWS Lambda, GCF, AZF e AFC. Entre os casos de uso desenvolvidos destacam-se: calculadora simples para avaliação de latência e sobrecarga de invocação; API com armazenamento em *object storage*; API integrada a serviços DBaaS; e uma aplicação científica de alinhamento de sequências genéticas baseada no algoritmo de Hirschberg [15], implementável em ambientes FaaS e IaaS.
- **Subsistemas de experimentação e avaliação**, responsáveis pela implantação e desmobilização (remoção) automática de cenários experimentais, execução repetitiva de *benchmarks* e coleta sistemática de métricas de desempenho;
- **Infraestrutura de execução distribuída**, baseada em *workers* geograficamente distribuídos, permitindo avaliar o comportamento das funções sob diferentes condições de rede e localização;
- **Subsistema de análise estatística**, que integra métodos de avaliação baseados em projeto *fatorial* 2^k e testes estatísticos para análise comparativa de desempenho entre provedores;
- **Subsistema de predição de desempenho**, capaz de estimar o tempo de execução de funções FaaS a partir da análise de seu código-fonte.

Como resultado desses desenvolvimentos foi criado o *framework* Orama⁺⁺, que integra todos os componentes descritos anteriormente, e permite a condução de experimentos reprodutíveis em ambientes FaaS.

Assim sendo, como forma de garantir transparência, reprodutibilidade e disseminação dos artefatos desenvolvidos nesta pesquisa, os principais componentes do *framework* Orama⁺⁺ foram disponibilizados publicamente em repositórios de código aberto. Os repositórios disponibilizados incluem:

- **Orama⁺⁺²**: repositório principal do *framework*, contendo os módulos responsáveis pela configuração do ambiente experimental, execução de *benchmarks* e integração entre os diferentes subsistemas da plataforma;
- **Orama⁺⁺ Results³**: repositório que disponibiliza os resultados obtidos nos experimentos conduzidos com o *framework*. Esses dados foram utilizados para a construção do *dataset* experimental empregado no treinamento dos modelos de predição desenvolvidos neste trabalho.

A disponibilização desses repositórios contribui para a reprodutibilidade científica dos experimentos realizados, e permite que outros pesquisadores utilizem e estendam o *framework* em novas investigações envolvendo ambientes FaaS.

1.4.2 Contribuições Científicas

Além dos artefatos tecnológicos desenvolvidos, esta pesquisa também apresenta contribuições científicas relevantes relacionadas à análise de desempenho e à predição de comportamento de aplicações FaaS. Entre as principais contribuições científicas destacam-se:

- Construção de um *dataset* experimental contendo registros de 332.190 execuções de *benchmarks* em diferentes provedores de nuvem;
- Investigação experimental do comportamento de plataformas FaaS sob diferentes cenários de concorrência, configurações de aplicação e localizações geográficas;
- Proposição de uma abordagem de predição de tempo de execução baseada em métricas de complexidade de software, utilizando métricas de Halstead [16];
- Desenvolvimento e treinamento de modelos de IA capazes de estimar o tempo de execução de funções FaaS a partir de características extraídas do código-fonte;
- Avaliação estatística comparativa entre diferentes provedores de nuvem utilizando métodos experimentais formais.

Essas contribuições ampliam o entendimento sobre o comportamento de aplicações *serverless*, e fornecem mecanismos para apoiar a tomada de decisão na seleção de provedores de nuvem com base em critérios quantitativos de desempenho.

²<https://github.com/unb-faas/orama>

³<https://github.com/unb-faas/orama-results>

1.4.3 Publicações Científicas

Os resultados obtidos ao longo desta pesquisa foram disseminados por meio de 26 publicações científicas, abrangendo tanto trabalhos diretamente relacionados ao tema da tese quanto produções adjacentes que contribuíram para a consolidação do conhecimento na área. Deste total, 11 publicações estão diretamente vinculadas às contribuições centrais do doutorado, enquanto 15 correspondem a investigações complementares.

Em termos de tipo de publicação, observa-se predominância de artigos em conferências internacionais, totalizando 10 trabalhos, seguidos por 8 publicações em conferências nacionais. Adicionalmente, foram publicados 7 artigos em periódicos internacionais, evidenciando a maturidade e a relevância dos resultados alcançados, bem como sua validação por meio de processos rigorosos de revisão por pares. Complementarmente, destaca-se a publicação de 1 capítulo de livro internacional, ampliando o alcance e a disseminação dos resultados da pesquisa.

Tabelas descritivas consolidando as publicações adjacentes (Tabela 7.1) e principais (Tabela 7.2) são apresentadas no Capítulo 7, que trata das considerações finais desta tese. Esse conjunto de publicações evidencia não apenas a evolução da investigação ao longo do doutorado, mas também a consolidação das contribuições propostas, abrangendo desde estudos iniciais e exploratórios, até trabalhos mais robustos e diretamente alinhados aos objetivos centrais da tese.

1.5 Organização deste Documento

Esta tese está estruturada, além deste capítulo introdutório, em mais seis capítulos. O Capítulo 2 traz uma visão geral sobre a Computação em Nuvem e seus principais modelos de serviço. Na sequência, o Capítulo 3 aborda aspectos de Inteligência Artificial, *Benchmarking* e análise estatística. No Capítulo 4 é apresentado, em detalhes, o *framework* Orama⁺⁺, incluindo aspectos de infraestrutura. O Capítulo 5 apresenta os trabalhos relacionados. No Capítulo 6 são mostradas as estratégias e as técnicas de avaliação do Orama⁺⁺ e os resultados obtidos. Por fim, o Capítulo 7 traz as considerações finais, os trabalhos futuros e a produção científica derivada da pesquisa realizada nesta tese.

Capítulo 2

Computação em Nuvem e Avaliação Experimental de FaaS

Este capítulo apresenta uma visão geral da Computação em Nuvem, abordando seus conceitos fundamentais e modelos de serviço. A Seção 2.1 introduz o conceito de Computação em Nuvem, destacando suas principais características, benefícios e impacto no desenvolvimento de aplicações. Em seguida, a Seção 2.2 discute as diferentes categorias de serviços oferecidos pelos provedores. A Seção 2.3 explora o paradigma *serverless*, com ênfase no modelo FaaS, abordando seu funcionamento, vantagens e desafios na execução de funções sob demanda sem a necessidade de gerenciamento explícito de servidores. Posteriormente, a Seção 2.4 aborda técnicas para a realização de *benchmarks* e análise estatística em cenários de FaaS. Na Seção 2.5 são abordados os aspectos da avaliação dos serviços FaaS. Por fim, a Seção 2.6 apresenta as considerações finais deste capítulo.

2.1 Computação em Nuvem

A Computação em Nuvem [1] é uma das maiores revoluções tecnológicas das últimas décadas, permitindo o acesso a recursos computacionais sob demanda, de forma escalável e flexível. Sua origem remonta aos anos 60, quando John McCarthy propôs a ideia de que o poder computacional poderia ser distribuído como um serviço público, semelhante à eletricidade [17]. No entanto, o conceito começou a se materializar somente no início dos anos 2000, com o avanço da virtualização e a popularização da Internet de alta velocidade [18]. Empresas como Amazon, Google e Microsoft impulsionaram essa tecnologia, oferecendo serviços de infraestrutura (*Infrastructure-as-a-Service* (IaaS) [1]), plataformas (*Platform-as-a-Service* (PaaS) [1]) e software (*Software-as-a-Service* (SaaS) [1]) na nuvem, o que transformou a maneira como sistemas e aplicações são desenvolvidos, implantados e gerenciados.

Atualmente, a Computação em Nuvem é essencial para empresas de todos os setores, proporcionando maior agilidade, redução de custos operacionais e acesso a tecnologias avançadas, como IA e *Big Data*. Com a possibilidade de escalar recursos conforme a demanda, organizações podem otimizar seu desempenho e garantir alta disponibilidade de serviços, sem a necessidade de grandes investimentos em infraestrutura física. Além disso, a Computação em Nuvem fomenta a inovação ao permitir que *startups* e desenvolvedores independentes experimentem novas soluções com rapidez, segurança e baixo custo. Com sua evolução contínua, impulsionada por avanços como a computação sem servidor (*serverless*) [2] e arquiteturas *multicloud*, a Computação em Nuvem segue como um pilar fundamental da transformação digital [19], que é o processo de integração de tecnologias digitais em todas as áreas de uma organização, promovendo mudanças fundamentais em seus modelos de negócio, operações e na forma como entrega valor aos clientes.

2.2 Modelos de Serviço em Nuvem

A Computação em Nuvem oferece três principais modelos de serviço [1]: *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) e *Software-as-a-Service* (SaaS). Esses modelos diferem no nível de controle e de responsabilidade do usuário sobre os recursos computacionais. Cada um desses modelos atende a diferentes necessidades de empresas e desenvolvedores, possibilitando maior flexibilidade e escalabilidade, conforme os requisitos de cada aplicação.

O modelo IaaS fornece recursos de computação virtualizados sob demanda, viabilizando que usuários aluguem servidores, armazenamento, redes e outros componentes essenciais, sem a necessidade de manter hardware físico [1]. Provedores como AWS [20], Microsoft Azure [21] e GCP [22] disponibilizam esses serviços com escalabilidade automática e cobrança baseada no uso. Esse modelo é amplamente utilizado por empresas que necessitam de infraestrutura flexível para hospedar aplicações, sites ou ambientes de testes e desenvolvimento, reduzindo custos operacionais e aumentando a eficiência na alocação de recursos.

O PaaS oferece um ambiente de desenvolvimento e execução de aplicações sem que os usuários precisem gerenciar infraestrutura subjacente [1]. Esse modelo inclui ferramentas para desenvolvimento, bancos de dados, *middleware* e serviços de integração, facilitando a criação, teste e implantação de aplicações. Serviços como *Google App Engine* (GAE) [23], Heroku [24] e Microsoft Azure App Services [25] oferecem soluções *Platform-as-a-Service* que eliminam a complexidade de configurar servidores e bancos de dados, o que possibilita que os desenvolvedores foquem na codificação e inovação. Esse modelo é ideal

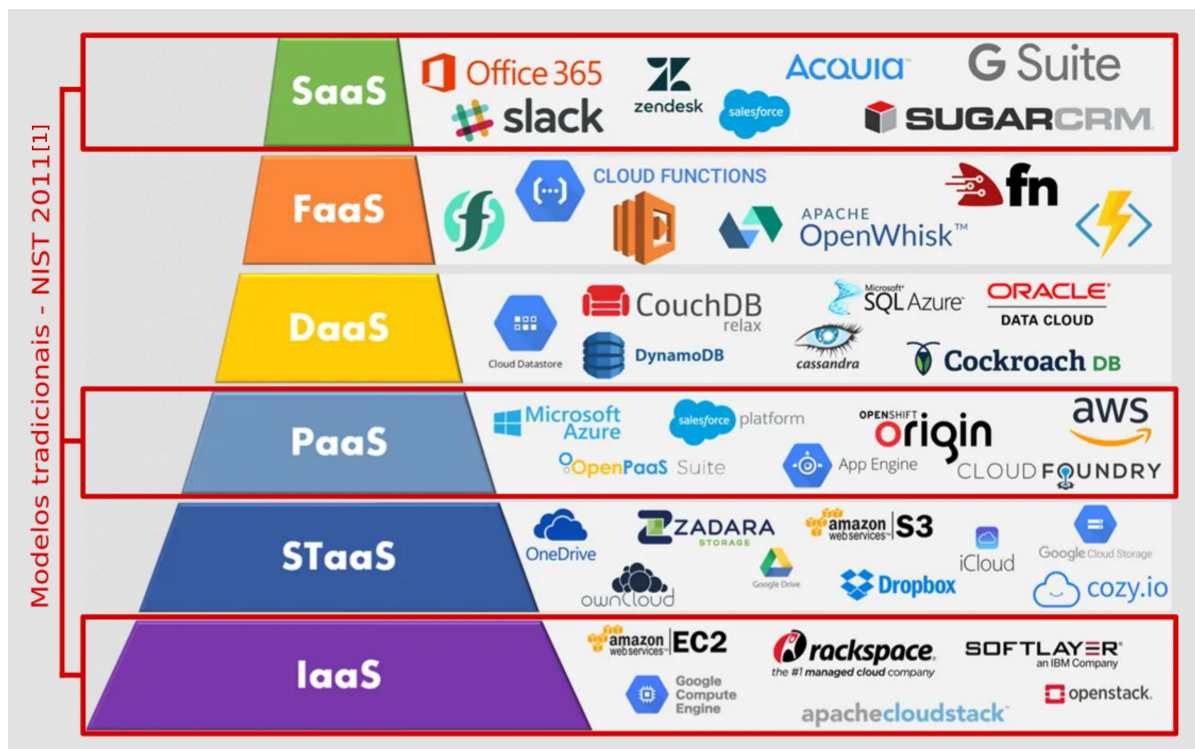


Figura 2.1: Exemplos de serviços em nuvem de diferentes modelos.

para *startups* e equipes de desenvolvimento que buscam rapidez na entrega de produtos ao mercado.

O SaaS fornece aplicações completas hospedadas na nuvem e acessíveis por meio da Internet, sem necessidade de instalação ou manutenção por parte do usuário [1]. Exemplos populares incluem Google Workspace (Gmail, Google Drive) [26], Microsoft 365 [27] e Salesforce [28]. Esse modelo permite que empresas e indivíduos utilizem softwares sofisticados sem precisar se preocupar com atualizações, segurança ou infraestrutura. O SaaS é amplamente adotado em diversos setores devido à sua praticidade, acessibilidade em múltiplos dispositivos e custo previsível baseado em assinaturas, sendo uma solução eficiente para organizações que buscam produtividade e colaboração remota.

Além dos modelos tradicionais de serviço em nuvem, nos últimos anos têm surgido diversas outras categorias no conceito de *Everything-as-a-Service* (XaaS) [29], que amplia a oferta de serviços para atender a demandas específicas. Exemplos incluem DBaaS [30], que fornece bancos de dados gerenciados; *Security-as-a-Service* (SECaaS) [31], voltado para soluções de segurança em nuvem; e *Artificial Intelligence-as-a-Service* (AIaaS) [32], que disponibiliza modelos de IA pré-treinados para aplicações.

A Figura 2.1 apresenta uma visão hierárquica dos principais modelos de serviços em Computação em Nuvem. Na figura estão representados os modelos tradicionais definidos pelo NIST em 2011 [1], tais como IaaS, PaaS e SaaS. Há ainda a representação do mo-

delo de serviço *Storage-as-a-Service* (STaaS), voltado ao fornecimento de armazenamento escalável e sob demanda para arquivos, objetos e dados persistentes em ambientes distribuídos. Acima do PaaS, encontra-se o *DaaS*, que também pode ser encontrado como *Database-as-a-Service* (DBaaS), focado na oferta de bancos de dados gerenciados. A figura também evidencia o modelo FaaS, associado ao paradigma *serverless*, no qual funções são executadas sob demanda sem a necessidade de gerenciamento explícito da infraestrutura subjacente, proporcionando maior eficiência e escalabilidade. O modelo de serviço FaaS será abordado com mais profundidade na Seção 2.3.

2.3 *Serverless e Function-as-a-Service (FaaS)*

A arquitetura *serverless* [2] é um modelo de Computação em Nuvem no qual os desenvolvedores podem construir e implantar aplicações sem a necessidade de gerenciar diretamente servidores. Nesse paradigma, os recursos de infraestrutura são alocados e escalados automaticamente pelo provedor de nuvem, o que possibilita que os usuários paguem apenas pelo tempo efetivo de execução do código, reduzindo custos associados à manutenção de servidores ociosos. Essa abordagem simplifica a implementação de aplicações, reduzindo a complexidade operacional e permitindo maior foco no desenvolvimento de funcionalidades.

O modelo *serverless* [2] abrange uma ampla variedade de serviços em nuvem, desde serviços para armazenamento e processamento de dados, sem a necessidade de provisionar infraestrutura, até mesmo bancos de dados, mensageria, orquestração de fluxos de trabalho e processamento de dados em tempo real. Além disso, soluções de *backend* gerenciado oferecem funcionalidades para autenticação, sincronização de dados e integração com outros serviços na nuvem. Essa abordagem permite que desenvolvedores foquem na lógica de negócios das aplicações, otimizando custos e simplificando a escalabilidade.

Dentro do paradigma *serverless*, um dos principais modelos de serviço é o *Function-as-a-Service* (FaaS), que permite a execução de funções de forma independente, acionadas por eventos específicos, tais como requisições HTTP, mensagens em filas ou mudanças em bancos de dados [3], conforme fluxo de execução mostrado na Figura 2.2. O FaaS elimina a necessidade de manter processos em execução contínua, pois as funções são instanciadas sob demanda e encerradas após a execução. Isso proporciona maior eficiência em termos de custo e escalabilidade, o que torna essa solução ideal para aplicações baseadas em eventos, APIs *serverless* e processamento assíncrono.

Os principais provedores de FaaS incluem a AWS, o GCP e o Azure. O provedor AWS oferta o AWS Lambda [4], que oferece suporte a diversas linguagens de programação, integração nativa com serviços da AWS e escalabilidade automática. O GCF [5], por sua vez, foca na simplicidade e na integração com o ecossistema do GCP, sendo uma excelente

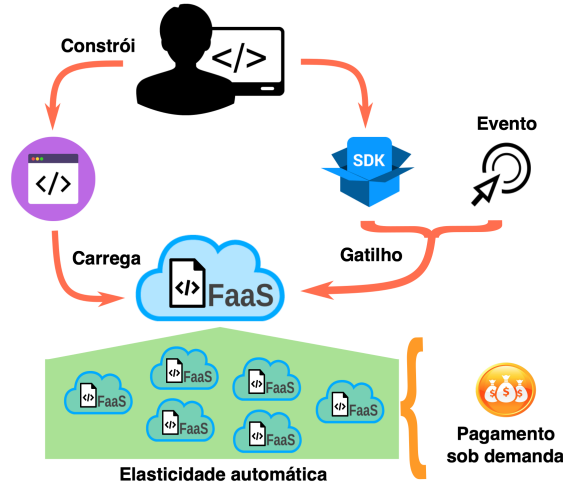


Figura 2.2: Fluxo de execução de FaaS.

escolha para aplicações que utilizam serviços do Google, tais como Firebase [33] ou Google Kubernetes Engine [34]. Já o AZF [6], da Microsoft, permite integração com serviços do Azure e execução de código em múltiplos gatilhos, como filas do Azure Storage [35] e eventos do Event Grid [36]. Além desses, o AFC [7] se destaca na Ásia, oferecendo uma alternativa competitiva para aplicações *serverless* em mercados emergentes.

Esses provedores diferenciam-se em termos de suporte a linguagens, integração com outros serviços e modelo de precificação, possibilitando que empresas escolham a melhor solução para suas necessidades. Um breve comparativo das principais características dos serviços de FaaS mencionados é apresentado na Tabela 2.1. Essa tabela organiza a comparação a partir de critérios como: tempo máximo de execução, linguagens suportadas, modelo de cobrança, gatilhos nativos, *cold start* e suporte a ambientes privados, evidenciando diferenças relevantes entre os serviços. O AWS Lambda, por exemplo, permite execuções de até 15 minutos, enquanto que o GCF é mais restritivo (9 minutos), e o AZF e o AFC suportam execuções mais longas, dependendo do plano. O AFC destaca-se pelo suporte a C++, e o AZF pela integração com o ecossistema Microsoft. Em relação ao modelo de cobrança, o AZF inclui o número de chamadas, diferentemente dos demais. Por fim, cada plataforma apresenta forte integração com seus serviços nativos e variações no comportamento de inicialização e suporte a redes privadas.

O crescimento do FaaS reflete a tendência de modernização das aplicações na nuvem, permitindo que empresas desenvolvam soluções mais ágeis, escaláveis e econômicas. Todavia, com a evolução desse modelo surgiram complicadores como o gerenciamento do estado das aplicações, a latência na ativação de funções (*cold start*) e a necessidade de ferramentas especializadas para observabilidade e depuração [3]. No entanto, a adoção continua de padrões como *Event-driven Architectures* [37], e o suporte a ambientes híbri-

Tabela 2.1: Comparação entre serviços de FaaS.

Característica	AWS Lambda	GCF	AZF	AFC
Tempo Máximo de Execução	15 min	9 min	5 min (consumo) / 60 min (dedicado)	60 min
Linguagens Suportadas	Node.js, Python, Java, Go, Ruby, .NET, PowerShell	Node.js, Python, Go, Java, .NET, PHP, Ruby	Node.js, Python, Java, .NET, PowerShell, TypeScript	Node.js, Python, Java, PHP, .NET, C++
Modelo de Cobrança	Tempo de execução e memória usada	Tempo de execução e recursos alocados	Tempo de execução, memória e chamadas	Tempo de execução e recursos alocados
Gatilhos Nativos	S3, API Gateway, DynamoDB, SNS, SQS, EventBridge	Cloud Storage, Pub/Sub, Firestore, HTTP	Event Grid, Blob Storage, Service Bus, HTTP	OSS, API Gateway, Log Service, MNS
Tempo de Inicialização (<i>Cold Start</i>)	Moderado	Moderado-alto	Variável (dependendo do plano)	Moderado
Suporte a Ambientes Privados	AWS <i>Virtual Private Cloud</i> (VPC)	VPC <i>Connector</i>	<i>Virtual Network</i> (VNET) <i>Integration</i>	VPC <i>Binding</i>

dos e multi-nuvem indicam que o FaaS continuará a desempenhar um papel fundamental no futuro da Computação em Nuvem.

A avaliação de soluções baseadas em FaaS depende de métodos que permitam comparar desempenho e escalabilidade sob diferentes condições. Nesse contexto, *benchmarks* fornecem métricas padronizadas e a análise estatística permite interpretar os resultados de forma consistente. Assim, a próxima seção apresenta esses conceitos e sua aplicação neste trabalho.

2.4 *Benchmarks* e Análise Estatística

Benchmarks e análise estatística estão fortemente interligados, pois a avaliação de desempenho depende de métodos rigorosos para interpretar e comparar resultados com confiabilidade. Enquanto um fornece medições sobre diferentes condições, o outro permite validar os dados, identificando padrões, tendências e variações significativas. O uso de técnicas adequadas garante que as conclusões sejam fundamentadas e não influenciadas por fatores aleatórios, favorecendo decisões mais precisas e embasadas em evidências. Nas próximas seções serão aprofundados aspectos sobre avaliação de ambientes distribuídos e métodos estatísticos importantes para avaliação desse tipo de ambiente.

2.4.1 Avaliação de Ambientes Distribuídos

A execução de *benchmarks* em ambientes distribuídos é essencial para avaliar o desempenho, a escalabilidade e a eficiência dos serviços e da infraestrutura. Em nuvem, diferentes provedores possuem arquiteturas, mecanismos de escalonamento e modelos de cobrança distintos, o que pode impactar diretamente o desempenho das funções e no custo da operação. Medir e analisar métricas tais como tempo de resposta, latência de inicialização (*cold start*), consumo de memória e uso de *Virtual Central Processing Unit* (vCPU) permitem compreender como cada plataforma se comporta sob diferentes condições de carga, e identificar possíveis gargalos.

Além disso, *benchmarks* bem estruturados auxiliam na tomada de decisões sobre qual provedor escolher para uma determinada aplicação, considerando não apenas desempenho, mas também confiabilidade e custo-benefício. Testes rigorosos e reproduzíveis são fundamentais para validar hipóteses sobre o comportamento das funções em diferentes cenários, garantindo que desenvolvedores e arquitetos de software tenham dados concretos para otimizar suas implementações. Assim, sem esse tipo de análise, empresas podem acabar utilizando serviços que não são ideais para suas necessidades, resultando em desperdício de recursos e degradação da experiência do usuário.

Para que as comparações entre provedores sejam justas e significativas, é necessário garantir que os testes sejam realizados em condições equivalentes. Isso inclui a utilização de funções idênticas em termos de código, configuração e consumo de recursos, além da execução de testes sob cargas semelhantes. Diferentes provedores podem ter otimizações específicas que influenciam o desempenho, tal como técnicas internas para minimizar *cold starts* [3] ou estratégias de escalonamento, tornando essencial a definição de um ambiente de testes padronizado para evitar vieses.

Outro ponto importante na comparação entre casos de uso equivalentes é a escolha das métricas de avaliação. Dependendo do objetivo do *benchmark*, pode ser mais relevante medir a latência média, a variabilidade dos tempos de resposta ou o custo total para uma determinada carga de requisições. Além disso, a repetição dos experimentos em diferentes momentos do dia, e sob diferentes condições de rede pode fornecer uma visão mais abrangente sobre o comportamento dos provedores. Comparações injustas, baseadas em cenários distintos, podem levar a conclusões equivocadas e prejudicar a escolha da melhor plataforma para um determinado contexto.

2.4.2 Métodos Estatísticos para Avaliar Desempenho

Métodos estatísticos [38] são técnicas utilizadas para coletar, organizar, analisar e interpretar dados, viabilizando a extração de informações relevantes e a tomada de decisões

embasadas. Eles são amplamente empregados em diversas áreas, incluindo a avaliação de desempenho de sistemas computacionais. Dentre os principais métodos estatísticos, destacam-se a Análise Descritiva [39], que resume e apresenta os dados de forma compreensível por meio de medidas como média, mediana e desvio padrão; os testes de hipóteses, como o teste t [13] pareado e o teste de Mann-Whitney [40], que verificam diferenças significativas entre grupos; a Regressão Estatística [41], utilizada para modelar relações entre variáveis; e o planejamento experimental, incluindo o projeto *fatorial* 2^k [12], que avalia o impacto de múltiplos fatores simultaneamente.

A avaliação de desempenho por meio de métodos estatísticos permite extrair conclusões confiáveis a partir de experimentos e medições. Técnicas como testes de hipótese [42], análise de variância (*Analysis of Variance* (ANOVA), na sigla em inglês) [43] e intervalos de confiança [44] são amplamente utilizadas para verificar se diferenças observadas entre configurações ou cenários são estatisticamente significativas. Além disso, o uso de estatísticas descritivas, tais como médias, medianas e desvios padrão, auxilia na compreensão da distribuição dos dados e na identificação de tendências. Esses métodos garantem que a análise de desempenho não seja influenciada por variações aleatórias, proporcionando uma base sólida para a tomada de decisões [38].

Outro aspecto importante é a escolha adequada do modelo estatístico conforme a natureza dos dados e do experimento. Métodos estatísticos podem ser classificados em paramétricos [43] e não paramétricos [45]. Métodos paramétricos são aqueles que assumem que os dados seguem uma distribuição probabilística específica, como a distribuição normal, e dependem de parâmetros dessa distribuição para a realização das inferências. Já os métodos não paramétricos não fazem suposições rígidas sobre a distribuição dos dados, sendo mais flexíveis e indicados para cenários em que tais pressupostos não podem ser garantidos ou são desconhecidos.

Esses métodos podem ser empregados dependendo das suposições feitas sobre a distribuição dos dados. Além disso, técnicas como regressão e análise de correlação permitem investigar relações entre variáveis e prever comportamentos em diferentes condições [38]. A combinação dessas abordagens possibilita uma avaliação detalhada do desempenho de sistemas, auxiliando na identificação de gargalos e oportunidades de otimização de forma sistemática e embasada [38].

Dentre os diversos métodos estatísticos existentes, destacam-se nesta pesquisa o projeto fatorial 2^k e o teste t pareado, por fornecerem ferramentas adequadas à avaliação de ambientes distribuídos. A escolha desses métodos se justifica pela necessidade de analisar múltiplos fatores de configuração e suas interações no contexto desta tese, bem como por permitir comparações controladas entre diferentes execuções sob condições experimentais correlatas. Esses métodos serão abordados em detalhes nas próximas subseções.

2.4.2.1 Projeto *Fatorial* 2^k

O projeto *fatorial* 2^k [12] é uma abordagem estatística amplamente utilizada para analisar o impacto de múltiplos fatores em um experimento, possibilitando avaliar suas interações e efeitos individuais. Diferente de métodos que testam um fator por vez, essa técnica considera combinações de variáveis, proporcionando uma visão mais completa do comportamento do sistema. Dessa forma, é possível identificar não apenas quais fatores influenciam significativamente o desempenho, mas também como a interação entre eles pode amplificar ou atenuar seus efeitos. Esse método é especialmente útil em contextos nos quais a complexidade do sistema impede uma análise isolada de variáveis.

Uma das principais vantagens do projeto *fatorial* 2^k é a eficiência na coleta de dados, pois permite obter informações detalhadas com um número reduzido de experimentos. O uso de configurações como o fatorial completo [12], que testa todas as combinações possíveis de fatores, e o fatorial fracionado [12], que reduz o número de testes mantendo a representatividade dos resultados, possibilita uma análise robusta mesmo em cenários com restrições de tempo ou recursos. Além disso, técnicas estatísticas, como a Análise de Variância (*Analysis of Variance* (ANOVA)) [46], são frequentemente aplicadas para validar os efeitos observados e determinar sua significância dentro do experimento.

No contexto da avaliação de desempenho de sistemas computacionais, o projeto *fatorial* 2^k é amplamente empregado para otimizar parâmetros de configuração, identificar gargalos e prever comportamentos sob diferentes condições operacionais. Em ambientes como Computação em Nuvem, nos quais múltiplos fatores podem influenciar simultaneamente o tempo de resposta, a latência e a utilização de recursos, essa abordagem se torna essencial para embasar decisões estratégicas. Ao fornecer um entendimento detalhado sobre o impacto das variáveis testadas, o projeto *fatorial* 2^k contribui para a melhoria contínua de sistemas e a maximização da eficiência operacional.

2.4.2.2 Teste t Pareado

O teste t pareado [13] é uma técnica estatística utilizada para comparar duas condições de um experimento quando as medições são realizadas nos mesmos sujeitos ou unidades amostrais. Esse método é especialmente útil em cenários nos quais se deseja avaliar o impacto de uma intervenção ou mudança, reduzindo a influência de fatores externos, pois cada observação serve como seu próprio controle. A análise é baseada na diferença entre os pares de observações, verificando se a média dessas diferenças é estatisticamente significativa, pois isso permite identificar variações reais no desempenho ou comportamento do sistema avaliado.

Na avaliação de desempenho de sistemas computacionais, os testes t pareados são amplamente aplicados para comparar configurações antes e depois de uma otimização, analisar o impacto de diferentes parâmetros, ou avaliar a eficácia de técnicas de melhoria. Como a variação natural entre amostras é controlada pelo pareamento, a análise se torna mais sensível a pequenas diferenças, aumentando a precisão dos resultados. No entanto, é fundamental garantir que as suposições do teste sejam atendidas, como a normalidade das diferenças e a independência das amostras, para que as conclusões obtidas sejam estatisticamente válidas.

Os cálculos envolvidos tanto no teste t pareado quanto no projeto *fatorial* 2^k podem se tornar bastante complexos, especialmente quando se lida com grandes volumes de dados e múltiplas variáveis interagindo simultaneamente. Executar essas operações manualmente ou com planilhas pode ser propenso a erros e inviável em cenários de maior escala. Por isso, contar com uma ferramenta automatizada, capaz de realizar esses cálculos de forma sistemática, precisa e livre de falhas humanas, é essencial para garantir análises confiáveis e decisões bem fundamentadas, especialmente, em domínios como avaliação de desempenho de sistemas computacionais.

A aplicação desses métodos estatísticos e experimentais fornece a base necessária para uma análise estruturada e confiável do desempenho em ambientes em nuvem. No entanto, para que essas técnicas sejam efetivamente empregadas, é fundamental definir como os experimentos são conduzidos em cenários reais de Computação em Nuvem, considerando diferentes provedores e configurações de infraestrutura. A próxima seção apresenta esse processo, detalhando a avaliação de provedores em ambientes FaaS e a definição de cenários experimentais que permitem exercitar os principais aspectos da infraestrutura sob análise.

2.5 Avaliação de Provedores em Ambientes FaaS

A escolha de um provedor de nuvem para implantação de aplicações em ambientes FaaS envolve um conjunto de atividades técnicas que, na prática, são conduzidas de forma empírica por desenvolvedores e arquitetos de software [47]. Esse processo exige a configuração de ambientes equivalentes em múltiplos provedores, a execução de experimentos controlados e a análise dos resultados obtidos. No entanto, a ausência de padronização nessas etapas torna a avaliação trabalhosa, sujeita a vieses e de difícil reprodução.

De forma geral, o processo de avaliação pode ser estruturado em um conjunto de atividades recorrentes, independentemente do tipo de aplicação considerada. Essas atividades incluem [48]: (i) modelagem do caso de uso, (ii) implementação e adaptação da aplicação para diferentes provedores, (iii) provisionamento de recursos, (iv) configuração de parâ-

metros de execução, (v) definição e execução de experimentos, (vi) coleta de métricas e (vii) análise dos resultados.

Para melhor compreender o custo operacional envolvido na avaliação manual de provedores FaaS, nesta tese é apresentada uma estimativa do tempo necessário para a execução das principais atividades, obtida por meio de julgamento de especialistas. As estimativas consideram um desenvolvedor com experiência intermediária em Computação em Nuvem e familiaridade básica com os provedores avaliados. Estudos sobre seleção sistemática de provedores *serverless* indicam que a comparação entre múltiplas plataformas demanda esforço significativo nas etapas de modelagem, configuração, medição e análise, o qual tende a crescer com o número de provedores considerados [49].

A Tabela 2.2 organiza esse esforço em um conjunto de atividades sequenciais que cobrem todo o ciclo experimental, desde a modelagem do caso de uso até a análise dos resultados. Observa-se que as etapas iniciais, como modelagem e implementação/adaptação da função, já concentram uma parcela relevante do tempo, refletindo a necessidade de adequação da aplicação ao paradigma *serverless*. Em seguida, atividades de infraestrutura, como provisionamento de recursos e configuração de integrações, apresentam variabilidade considerável, evidenciando diferenças na complexidade operacional entre provedores. As fases experimentais: definição de cenários; execução dos experimentos; e coleta de métricas, também contribuem de forma significativa para o esforço total, especialmente devido à necessidade de repetição e controle das execuções. Por fim, a análise dos resultados representa uma etapa não trivial, exigindo interpretação cuidadosa dos dados coletados.

Tabela 2.2: Estimativa de tempo para execução manual das atividades por provedor.

Atividade	Tempo Estimado ¹
Modelagem do caso de uso	1 – 2 horas
Implementação/adaptação da função	2 – 6 horas
Provisionamento de recursos	1 – 3 horas
Configuração de parâmetros de execução	30 min – 1 hora
Configuração de integrações (storage/DB)	1 – 4 horas
Definição de cenários experimentais	1 – 2 horas
Execução dos experimentos	2 – 8 horas
Coleta e organização de métricas	1 – 3 horas
Análise dos resultados	2 – 6 horas
Total estimado por provedor	11,5 – 35 horas

De forma consolidada, a estimativa total por provedor varia entre 11,5 e 35 horas, indicando uma ampla faixa de esforço dependendo da complexidade do cenário e do nível de detalhamento da avaliação. Em contextos multi-nuvem, esse custo tende a crescer de

¹Os intervalos de tempo apresentados correspondem a estimativas baseadas em experiência de desenvolvedores e arquitetos de software, considerando um cenário típico de avaliação em ambiente FaaS, e não a medições reportadas em trabalhos prévios. Variações podem ocorrer conforme perfil de desenvolvedor, complexidade da aplicação e grau de integração com outros serviços cloud.

maneira linear ou até superlinear, devido à heterogeneidade entre plataformas, reforçando a necessidade de abordagens automatizadas para reduzir o esforço manual e aumentar a reprodutibilidade dos experimentos [47].

Além da análise retrospectiva dos resultados experimentais, uma questão adicional em ambientes FaaS está relacionada à predição de desempenho e custo de execução. Diferentemente de ambientes tradicionais, o comportamento de aplicações *serverless* é influenciado por fatores dinâmicos e, muitas vezes, não determinísticos, como variabilidade de carga [3], políticas internas de escalonamento dos provedores e compartilhamento de infraestrutura subjacente.

A dependência do provedor também introduz heterogeneidade significativa, uma vez que diferentes plataformas adotam estratégias distintas de alocação de recursos, gerenciamento de *cold start* [50] e isolamento de execução. O fenômeno de *cold start*, em particular, adiciona latência imprevisível às invocações, dificultando a modelagem precisa do tempo de resposta, especialmente em cenários com baixa frequência de requisições ou alta elasticidade.

Adicionalmente, a escalabilidade automática, embora seja uma das principais vantagens do modelo FaaS, contribui para a complexidade da predição, pois o desempenho observado pode variar conforme o nível de paralelismo, a concorrência e as condições momentâneas do ambiente de execução. Esses fatores tornam desafiadora a construção de modelos preditivos robustos e generalizáveis.

Apesar dessas dificuldades, a capacidade de prever desempenho e custo é fundamental para apoiar a tomada de decisão na seleção de provedores e na configuração de aplicações, favorecendo a otimização de recursos e a redução de custos operacionais. Nesse contexto, abordagens sistemáticas baseadas em experimentação controlada e modelagem estatística tornam-se essenciais para viabilizar a predição em ambientes FaaS.

Para demonstrar o processo de avaliação de FaaS, nesta tese foram considerados quatro casos de uso representativos, que refletem diferentes padrões de aplicação em ambientes FaaS e serão descritos nas próximas seções.

2.5.1 Calculadora Simples

Este caso de uso representa uma função básica, sem dependência de serviços externos, sendo frequentemente utilizada como *baseline* para avaliação de desempenho em ambientes FaaS. As atividades operacionais necessárias para avaliar esse cenário incluem:

- Implementação da função em múltiplas linguagens ou adaptação para compatibilidade com os provedores;

- Configuração de parâmetros, tais como memória alocada e tempo máximo de execução;
- Implantação da função em diferentes plataformas FaaS;
- Definição de cenários de carga (número de requisições e concorrência);
- Execução repetida de invocações para coleta de métricas de latência e tempo de execução;
- Consolidação e análise estatística dos resultados.

Apesar de sua simplicidade, esse caso é particularmente relevante por isolar o comportamento intrínseco da infraestrutura FaaS, permitindo evidenciar diferenças de *overhead*, *cold start* e desempenho base entre provedores. A análise desses fatores é importante, pois o *overhead* representa o custo adicional introduzido pela plataforma para gerenciar a execução da função, impactando diretamente a latência observada, especialmente em aplicações sensíveis a tempo de resposta [51].

Já o *cold start* influencia cenários com baixa frequência de invocação ou alta elasticidade, podendo degradar significativamente a experiência do usuário em funções acionadas sob demanda [50]. O desempenho base reflete a eficiência do ambiente de execução subjacente, sendo determinante para cargas intensivas e execuções repetidas.

Por não envolver dependências externas, este cenário elimina interferências de rede ou serviços de terceiros, permitindo uma comparação mais precisa entre provedores e servindo como referência para experimentos mais complexos. Em relação ao esforço experimental, as estimativas indicam um custo relativamente reduzido, porém não desprezível:

- Tempo total estimado: 8 a 12 horas por provedor;
- Principal custo: execução de experimentos e análise de métricas;
- Baixa complexidade de configuração.

Observa-se que a maior parte do tempo não está associada à implementação da função, mas sim à execução controlada dos experimentos e à análise estatística dos resultados. A necessidade de múltiplas repetições para obtenção de significância estatística, aliada à variação inerente dos ambientes FaaS, contribui para esse custo. Ainda assim, trata-se de um dos cenários mais eficientes em termos de esforço por informação obtida, sendo adequado como etapa inicial em estudos comparativos entre provedores.

2.5.2 API com Armazenamento em *Object Storage*

Neste cenário, a aplicação consiste em uma *Application Programming Interface* (API) que realiza operações de escrita e leitura em serviços de armazenamento de objetos [52]. Esses serviços armazenam dados como objetos independentes, identificados por uma chave única e acessados via API, oferecendo alta escalabilidade, durabilidade e suporte a grandes volumes de dados não estruturados. As atividades operacionais necessárias para avaliar esse cenário incluem:

- Implementação da API com integração a serviços como S3, Cloud Storage ou Blob Storage;
- Configuração de credenciais e permissões de acesso aos serviços de armazenamento;
- Adaptação da aplicação às APIs específicas de cada provedor;
- Provisionamento dos serviços de armazenamento em cada ambiente;
- Definição de cargas de trabalho representativas (tamanho de objetos, frequência de acesso);
- Execução de testes considerando operações de leitura e de escrita;
- Coleta de métricas de latência, *throughput* e custo associado às operações.

Diferentemente de cenários puramente computacionais, este caso evidencia o impacto direto da comunicação com serviços externos, refletindo-se na latência de rede, na variabilidade de desempenho e na composição do custo operacional. A latência passa a depender não apenas do ambiente de execução da função, mas também da proximidade e da qualidade da comunicação com o serviço de armazenamento. O *throughput* torna-se um fator determinante em cenários com maior volume de dados, enquanto o custo operacional incorpora múltiplas dimensões, incluindo operações de leitura, escrita, armazenamento e transferência. Além disso, diferenças nas APIs e *Software Development Kit* (SDK) entre provedores introduzem variações tanto no desempenho quanto no esforço de desenvolvimento e manutenção.

A introdução de armazenamento externo aumenta significativamente a complexidade do cenário, tanto na fase de configuração quanto na execução dos experimentos. As estimativas de esforço são apresentadas a seguir:

- Tempo total estimado: 12 a 20 horas por provedor;
- Principal custo: configuração de permissões e integração com APIs específicas;
- Variabilidade devido a diferenças entre SDKs e políticas de acesso.

Observa-se que o aumento no tempo estimado, em comparação a cenários mais simples, está diretamente associado à necessidade de configuração segura de credenciais, à adaptação às particularidades de cada provedor e à validação de diferentes padrões de acesso aos dados. Adicionalmente, a maior variabilidade dos resultados experimentais, decorrente da dependência de rede e de serviços externos, tende a exigir um número maior de repetições para garantir significância estatística, ampliando o esforço total. Esse cenário, portanto, representa um nível intermediário de complexidade e é fundamental para a avaliação de aplicações reais que dependem de persistência de dados em ambientes FaaS.

2.5.3 API com Armazenamento em DBaaS

Neste caso, a aplicação interage com *Database-as-a-Service* (DBaaS) [30], um modelo de serviço em nuvem que fornece bancos de dados totalmente gerenciados, viabilizando criar, operar e escalar bases de dados sem a necessidade de administrar a infraestrutura subjacente. Esse cenário é amplamente utilizado em aplicações corporativas e introduz uma camada adicional de complexidade relacionada à persistência estruturada de dados. As atividades operacionais necessárias para avaliar esse cenário incluem:

- Modelagem do esquema de dados e configuração do banco de dados;
- Implementação da API com acesso ao banco utilizando *drivers* específicos;
- Configuração de conexões seguras (VPC, *Secure Sockets Layer* (SSL), credenciais);
- Ajuste de parâmetros de conexão, tais como *pooling* e *timeout*;
- Execução de cargas de trabalho com operações de leitura e de escrita;
- Monitoramento de latência, tempo de resposta e uso de recursos do banco;
- Análise do impacto da proximidade geográfica e da configuração de rede.

Diferentemente do uso de armazenamento em objetos, este cenário evidencia particularidades adicionais associadas à gestão de conexões e ao comportamento dos sistemas gerenciadores de banco de dados. A latência não depende apenas da rede, mas também do processamento interno do banco e da eficiência das consultas executadas. O gerenciamento de conexões torna-se um fator crítico, especialmente em ambientes FaaS, nos quais a natureza efêmera das funções pode levar à criação excessiva de conexões, impactando diretamente o desempenho e a escalabilidade. Além disso, parâmetros como *pooling* e *timeout* influenciam significativamente a estabilidade e o tempo de resposta da aplicação.

Outro aspecto relevante é a variabilidade de desempenho entre provedores, decorrente de diferenças nas implementações dos serviços de DBaaS, mecanismos de armazenamento,

políticas de escalonamento e configurações padrão. A proximidade geográfica entre a função e o banco de dados, bem como a configuração de rede, também exerce forte influência sobre a latência e a previsibilidade dos resultados. Em conjunto, esses fatores tornam esse cenário mais representativo de aplicações reais, porém mais difícil de avaliar de forma controlada. Dessa forma, este cenário apresenta maior complexidade operacional, conforme indicado nas estimativas a seguir:

- Tempo total estimado: 15 a 28 horas por provedor;
- Principal custo: configuração de banco de dados, rede e conexões;
- Possíveis dificuldades com latência, segurança e *tuning* de conexões.

O aumento no tempo estimado reflete a necessidade de atividades adicionais, como modelagem de dados, configuração de infraestrutura de rede, estabelecimento de conexões seguras e ajuste fino de parâmetros de acesso ao banco. Além disso, a execução experimental tende a demandar maior esforço devido à variabilidade introduzida pelo sistema de banco de dados e pela rede, exigindo maior controle dos experimentos e um número mais elevado de repetições para garantir significância estatística. Dessa forma, este cenário representa um nível avançado de complexidade e é essencial para a avaliação de aplicações que dependem de persistência estruturada em ambientes FaaS.

2.5.4 Alinhamento de Sequências Genéticas

Alinhamento de sequências é o processo de organizar sequências de *DeoxyriboNucleic Acid* (DNA), *RiboNucleic Acid* (RNA) ou proteínas para identificar regiões de similaridade que podem refletir relações funcionais, estruturais ou evolutivas entre elas [53]. Essa técnica maximiza as correspondências entre nucleotídeos ou aminoácidos, considerando lacunas por meio de sistemas de pontuação. Neste contexto, o caso de uso representa uma aplicação computacional intensiva, sendo adequado para avaliar o comportamento de funções com elevado consumo de vCPU e memória em ambientes FaaS. As atividades operacionais necessárias para avaliar esse cenário incluem:

- Implementação do algoritmo em uma linguagem suportada pelos provedores, respeitando limitações como tempo máximo de execução e quantidade de recursos computacionais disponíveis;
- Definição de conjuntos de dados de entrada com diferentes tamanhos e níveis de complexidade;
- Configuração de parâmetros de execução, especialmente memória alocada;

- Implantação da aplicação em múltiplos provedores FaaS e, opcionalmente, em ambientes IaaS para comparação;
- Execução de experimentos variando o tamanho das entradas e níveis de paralelismo;
- Coleta de métricas de tempo de execução, consumo de recursos e custo;
- Análise da escalabilidade e eficiência computacional.

Este cenário evidencia de forma clara o impacto das arquiteturas de execução sobre aplicações *CPU-bound*, nas quais o desempenho é fortemente condicionado à capacidade de processamento disponível. Diferentes provedores FaaS adotam estratégias distintas de alocação de vCPU em função da memória configurada, o que pode resultar em variações significativas no tempo de execução. Além disso, limitações como *timeouts* máximos e restrições de recursos podem inviabilizar execuções mais longas ou exigir particionamento das tarefas, afetando diretamente a eficiência global.

Outro aspecto relevante é a escalabilidade. Embora o modelo FaaS favoreça a execução paralela, o ganho efetivo depende da granularidade das tarefas, do custo de orquestração e do *overhead* associado à inicialização das funções. Em aplicações intensivas, esse *overhead* tende a ser amortizado, tornando o cenário adequado para avaliar a relação entre paralelismo, custo e desempenho. Adicionalmente, a comparação com ambientes IaaS permite identificar limites de eficiência do modelo FaaS em cargas de trabalho de alto desempenho.

Do ponto de vista operacional, este cenário apresenta elevada complexidade, tanto pela natureza computacional quanto pela necessidade de controle experimental rigoroso. As estimativas de esforço são apresentadas a seguir:

- Tempo total estimado: 18 – 32 horas por provedor;
- Principal custo: execução de experimentos com diferentes tamanhos de entrada e níveis de paralelismo;
- Necessidade de controle rigoroso de variáveis para análise de escalabilidade e desempenho.

O tempo estimado é superior aos demais cenários devido à necessidade de múltiplas execuções com variações sistemáticas de entrada e configuração, bem como ao maior tempo individual de execução das funções. Além disso, a análise dos resultados exige maior rigor metodológico para avaliar escalabilidade, eficiência computacional e relação custo-benefício. Esse cenário, portanto, representa um nível avançado de complexidade e é essencial para compreender o comportamento de aplicações intensivas em ambientes FaaS.

Dessa forma, considerando um cenário com quatro provedores de nuvem, o esforço total pode tornar a avaliação impraticável, mesmo para casos de uso moderadamente complexos. Além disso, esse processo tende a ser iterativo, uma vez que ajustes de configuração frequentemente exigem novas execuções experimentais. Outro fator relevante é que parte significativa desse tempo é consumida por atividades operacionais repetitivas, como provisionamento de infraestrutura, configuração de ambientes e execução manual de testes. Essas atividades não agregam diretamente valor analítico, mas são necessárias para viabilizar a experimentação.

Dessa forma, observa-se que o custo temporal do processo manual constitui uma barreira significativa para a realização de estudos abrangentes e reproduzíveis em ambientes FaaS multi-nuvem. Esse cenário reforça a motivação para o desenvolvimento de abordagens automatizadas, tal como o *framework* Orama⁺⁺ proposto nesta tese, capaz de reduzir substancialmente o esforço necessário para condução desses experimentos.

2.6 Considerações Finais

A Computação em Nuvem transformou a maneira como aplicações são desenvolvidas, implantadas e escaladas, oferecendo diferentes modelos de serviço que atendem a variadas necessidades de desenvolvimento e operação. Dentro desse cenário, a abordagem *serverless* e, em particular, o modelo FaaS, emergiu como uma solução flexível e eficiente, eliminando a necessidade de gerenciamento de infraestrutura. No entanto, apesar das vantagens em termos de escalabilidade e otimização de custos, a diversidade de implementações entre provedores impõe desafios na avaliação de desempenho e previsibilidade das funções.

Para mitigar essas incertezas, a realização de *benchmarks* estruturados e a aplicação de técnicas de análise estatística são essenciais para uma comparação fidedigna entre diferentes ambientes FaaS. Além disso, com o auxílio de modelos de IA, é possível utilizar as informações dos *benchmarks* para prever o tempo de execução de uma aplicação em *FaaS*. O próximo capítulo explorará os principais aspectos de IA que foram utilizados na elaboração desta tese.

Capítulo 3

Redes Neurais e Técnicas de Predição de Desempenho

Este capítulo aborda aspectos fundamentais para o desenvolvimento e a avaliação de modelos de Inteligência Artificial em diferentes contextos. Assim, a Seção 3.1 apresenta uma visão geral sobre o tema. Na sequência, a Seção 3.2 apresenta os fundamentos de Inteligência Artificial e Aprendizado de Máquina. A Seção 3.3 apresenta abordagens para a construção de modelos capazes de realizar previsões com base em padrões extraídos dos dados. Na Seção 3.4 são discutidos os métodos para preparar os dados antes do treinamento, garantindo maior qualidade e eficiência na modelagem. Por fim, a Seção 3.5 apresenta as considerações finais deste capítulo.

3.1 Contextualização

Inteligência Artificial (IA) [54] é um campo da Ciência da Computação dedicado ao desenvolvimento de sistemas capazes de realizar tarefas que, tradicionalmente, exigem inteligência humana, tais como aprendizado, raciocínio, percepção e tomada de decisão. Esse domínio abrange diferentes abordagens, incluindo sistemas baseados em regras, métodos probabilísticos e, mais recentemente, técnicas de Aprendizado de Máquina [55].

Na engenharia de sistemas, a IA tem assumido um papel relevante ao apoiar o projeto, a operação e a evolução de sistemas complexos [56]. O seu uso permite automatizar processos de monitoramento, detectar falhas, otimizar fluxos de trabalho e melhorar a confiabilidade de arquiteturas distribuídas [56]. Além disso, contribui para a análise de grandes volumes de dados operacionais, auxiliando na tomada de decisão e no aumento da eficiência em ciclos de desenvolvimento e operação [56].

Na Computação em Nuvem, a IA é amplamente aplicada para otimização de recursos, previsão de demanda e melhoria da eficiência de serviços distribuídos [57]. Em arquite-

turas modernas, como *serverless* e FaaS, técnicas baseadas em dados são utilizadas para ajustar dinamicamente a alocação de recursos, identificar padrões de uso e reduzir custos operacionais [3] [8]. Esses modelos também permitem a análise contínua de métricas de desempenho, contribuindo para maior escalabilidade e estabilidade dos sistemas.

Nesse cenário, modelos de IA são aplicados em uma ampla variedade de problemas, tais como classificação de dados, análise preditiva e estimação de comportamentos futuros com base em históricos [58]. Além disso, também são utilizados em tarefas como análise de sentimentos, sistemas de recomendação e detecção de anomalias [59]. De forma geral, esses modelos se beneficiam de grandes volumes de dados para identificar padrões que não são facilmente capturados por abordagens tradicionais.

3.2 Fundamentos de Inteligência Artificial e Aprendizado de Máquina

O campo da Inteligência Artificial (IA) baseia-se na construção de sistemas computacionais capazes de executar tarefas associadas à inteligência humana. Essas soluções podem ser desenvolvidas a partir de diferentes paradigmas, incluindo abordagens simbólicas (baseado em lógica e regras) [60], estatísticas (probabilístico) [60] e orientadas a dados [60], sendo estas últimas predominantes nas aplicações contemporâneas. O avanço da área está diretamente relacionado à capacidade de processar grandes volumes de dados e extrair conhecimento útil a partir deles, permitindo a automação de tarefas complexas e a melhoria contínua de desempenho.

O Aprendizado de Máquina [55], subárea da IA, concentra-se no desenvolvimento de modelos capazes de aprender padrões diretamente dos dados, sem a necessidade de regras explícitas programadas para cada situação. Esse paradigma tem impulsionado significativamente a evolução da área, apoiado pelo aumento da disponibilidade de dados, pelo avanço de algoritmos e pela expansão do poder computacional [61]. Como resultado, técnicas de IA têm sido amplamente aplicadas em domínios como automação industrial, análise de dados e personalização de serviços digitais [62].

No contexto da Computação em Nuvem, técnicas de Aprendizado de Máquina são utilizadas para previsão de demanda, otimização de recursos e melhoria da eficiência de aplicações distribuídas [32]. Em particular, na avaliação de desempenho de ambientes FaaS, esses modelos auxiliam na análise automatizada de métricas, na identificação de gargalos e na previsão de tempos de execução sob diferentes condições operacionais, contribuindo para a redução de latência e custos [8].

De maneira geral, o Aprendizado de Máquina pode ser dividido em três paradigmas principais [55, 60]: supervisionado, não supervisionado e por reforço. No aprendizado

supervisionado, o modelo é treinado com dados rotulados, aprendendo uma relação entre entradas e saídas. No aprendizado não supervisionado, busca-se identificar estruturas ocultas em dados não rotulados, como agrupamentos e padrões latentes. Já no aprendizado por reforço, o modelo aprende por interação com o ambiente, sendo guiado por recompensas e penalidades ao longo do tempo.

Os processos de Aprendizado de Máquina voltados à predição seguem um fluxo estruturado composto por etapas interdependentes [60]. Inicialmente, realiza-se a coleta de dados, seguida pelo pré-processamento, que envolve limpeza, normalização e transformação. Em seguida, os dados são divididos em conjuntos de treinamento, validação e teste, permitindo tanto o ajuste dos modelos quanto sua avaliação de generalização. Posteriormente, ocorre a definição e o treinamento dos modelos, incluindo a escolha de algoritmos e a otimização de hiperparâmetros. Por fim, o modelo é avaliado por meio de métricas quantitativas e análises gráficas que auxiliam na interpretação de seu desempenho [63].

Esse processo, em especial a divisão dos dados de treinamento, é essencial para reduzir problemas como *overfitting* e *underfitting* [64], nos quais o modelo se ajusta excessivamente aos dados de treinamento ou apresenta capacidade insuficiente de representação, comprometendo sua generalização.

Nesse contexto, Redes Neurais artificiais se destacam como uma das abordagens mais relevantes para tarefas de predição, especialmente em cenários complexos e de alta dimensionalidade [62]. A próxima seção apresenta os principais conceitos relacionados a Redes Neurais com foco em problemas de predição.

3.3 Redes Neurais para Predição

A análise preditiva [65] é uma aplicação da IA voltada à estimativa de eventos ou comportamentos futuros a partir de dados históricos. Essa abordagem explora padrões e relações subjacentes nos dados para antecipar tendências, falhas, demandas de recursos e, em particular, métricas de desempenho em ambientes de Computação em Nuvem, como tempos de execução e consumo de recursos. No contexto de Sistemas Distribuídos, sua aplicação é essencial para apoiar decisões operacionais, permitindo ajustes dinâmicos na alocação de recursos computacionais, com impacto direto em custo, escalabilidade e *performance*.

Diversas técnicas podem ser empregadas em modelagem preditiva, incluindo regressão linear [66], árvores de decisão [67], máquinas de vetores de suporte (*Support Vector Machines* (SVM)) [68] e métodos de *ensemble*, como *Random Forest* [69] e *Gradient Boosting* [70]. Modelos lineares são adequados para relações mais simples e interpretáveis, enquanto árvores de decisão capturam interações não lineares entre variáveis. Métodos baseados em *ensemble* combinam múltiplos modelos para aumentar robustez e reduzir

variância, sendo amplamente utilizados em problemas com alta complexidade. A escolha da técnica depende da natureza dos dados, do grau de não linearidade do problema e dos objetivos de predição.

Redes Neurais artificiais [71] destacam-se nesse cenário por sua capacidade de modelar relações altamente não lineares e interações complexas entre variáveis, especialmente em conjuntos de dados de alta dimensionalidade. Diferentes arquiteturas podem ser utilizadas conforme a estrutura dos dados e a natureza do problema preditivo.

As redes do tipo *Multilayer Perceptron* (MLP), também conhecidas como redes densamente conectadas (*Dense*) [72], são amplamente utilizadas em problemas de regressão e classificação. Essas redes realizam transformações sucessivas dos dados por meio de camadas totalmente conectadas, sendo particularmente adequadas quando os dados podem ser representados de forma tabular e não apresentam dependência temporal explícita. No contexto deste trabalho, MLPs são relevantes por sua eficiência computacional e capacidade de servir como *baseline* robusto para predições de desempenho em ambientes FaaS.

Por outro lado, arquiteturas recorrentes como *Long Short-Term Memory* (LSTM) [73], são projetadas para capturar dependências temporais em sequências de dados. A sua estrutura interna com mecanismos de memória permite preservar informações relevantes ao longo do tempo, o que é essencial em cenários nos quais o desempenho de funções pode variar de acordo com padrões históricos de execução, carga do sistema e variações de entrada. Essa característica torna as LSTMs particularmente adequadas para modelagem de séries temporais em ambientes de Computação em Nuvem.

As redes *Bidirectional Long Short-Term Memory* (BLSTM) [74], estendem esse conceito ao considerar simultaneamente dependências passadas e futuras na sequência. Essa abordagem bidirecional permite uma representação mais rica do contexto temporal, sendo útil em cenários de análise *offline*, nos quais todo o histórico de execução está disponível. No contexto desta tese, essa característica é especialmente relevante para melhorar a precisão de predições baseadas em janelas completas de observação de métricas.

Além dessas arquiteturas, outras variações podem ser consideradas, tais como redes convolucionais (*Convolutional Neural Network* (CNN), na sigla em inglês) [75], quando há estrutura espacial ou padrões locais nos dados, e modelos híbridos que combinam CNNs e LSTMs para capturar simultaneamente padrões locais e dependências temporais. Mais recentemente, arquiteturas baseadas em mecanismos de atenção [76] e *Transformers* [76] também têm sido exploradas em problemas de séries temporais devido à sua capacidade de modelar dependências de longo alcance de forma eficiente.

No contexto deste trabalho, a escolha da arquitetura neural considerou a natureza das métricas coletadas em ambientes FaaS, as quais podiam apresentar tanto comportamento

temporal quanto variações não lineares complexas. Nesse cenário, redes do tipo *Dense* são adequadas para cenários estáticos ou agregados, LSTMs para capturar dinâmica temporal de execução, e BLSTMs para explorar de forma mais completa o histórico disponível, aumentando a capacidade de generalização do modelo preditivo.

Diante da aplicação de diferentes técnicas e da definição do modelo preditivo adotado, torna-se fundamental estabelecer critérios objetivos para avaliar sua qualidade e confiabilidade. A análise do desempenho de modelos preditivos permite verificar sua capacidade de generalização, identificar problemas como *overfitting* e *underfitting* [64] e comparar diferentes abordagens de forma sistemática. Nesse contexto, a próxima seção apresenta métodos de avaliação baseados em métricas quantitativas e análises gráficas, que possibilitam uma interpretação mais aprofundada dos resultados obtidos e subsidiam a escolha do modelo mais adequado para o problema investigado.

3.3.1 Avaliação de Modelos Preditivos

A avaliação de modelos preditivos é uma etapa fundamental no processo de desenvolvimento, pois permite mensurar a eficácia e a precisão do modelo em relação ao problema abordado [58]. Essa avaliação é realizada por meio de métricas como acurácia, precisão, *recall* e *F1-score*, cuja escolha depende do tipo de tarefa, tais como classificação, regressão ou detecção de anomalias.

Nesse contexto, a validação cruzada destaca-se como uma técnica amplamente utilizada, na qual os dados são particionados em subconjuntos, possibilitando múltiplos ciclos de treinamento e teste e, assim, uma estimativa mais robusta do desempenho do modelo [55]. Além de fornecer uma avaliação mais confiável, essa abordagem contribui para a identificação de possíveis casos de superajuste, ao verificar a capacidade do modelo de generalizar para dados não vistos.

A escolha da métrica de avaliação é crucial porque diferentes problemas podem exigir enfoques distintos, tais como priorizar a precisão em tarefas de diagnóstico médico ou o *recall* em sistemas de recomendação. Além das métricas de desempenho, a avaliação de modelos de IA também envolve a análise de seu comportamento em cenários reais, e sua capacidade de lidar com dados não vistos ou com ruídos. O teste com dados fora da amostra de treinamento, também conhecido como conjunto de validação ou teste [55], permite verificar se o modelo é robusto e consistente em diversas condições. Nesse sentido, várias técnicas podem ser aplicadas. Tais técnicas serão detalhadas nas próximas seções.

3.3.1.1 Função de Perda (*Loss*)

A função de perda (*Loss*) é uma medida quantitativa que expressa a diferença entre as previsões de um modelo de Aprendizado de Máquina e os valores reais ou esperados [77]. Ela desempenha papel central no processo de treinamento, ao fornecer um indicador do desempenho do modelo. Durante o treinamento, busca-se minimizar essa função por meio do ajuste dos parâmetros do modelo com base nos gradientes calculados, reduzindo progressivamente o erro observado.

A escolha da função de perda depende do tipo de problema e influencia diretamente a capacidade do modelo de aprender padrões relevantes e generalizar para novos dados. Em tarefas de regressão, é comum o uso do erro quadrático médio (*Mean Squared Error* (MSE), na sigla em inglês [63]), que penaliza erros maiores de forma mais intensa, ou do erro absoluto médio (*Mean Absolute Error* (MAE), na sigla em inglês [78]), que apresenta menor sensibilidade a *outliers*. Em problemas de classificação, destaca-se a entropia cruzada [79], que mede a discrepância entre a distribuição de probabilidades prevista pelo modelo e a distribuição real dos dados, sendo amplamente utilizada em cenários binários e multiclasse.

Além dessas, a perda *hinge* [80] é frequentemente empregada em classificadores de margem, como máquinas de vetor de suporte (*Support Vector Machines* (SVM) [68]), ao penalizar previsões incorretas ou com baixa margem em relação ao rótulo correto. Em contextos mais específicos, podem ser adotadas funções de perda personalizadas, como versões ponderadas para lidar com desbalanceamento de classes ou perdas adversariais em redes generativas adversariais [81]. Nesses cenários, uma rede geradora e uma discriminadora são treinadas de forma competitiva para produzir dados sintéticos realistas. Assim, a definição adequada da função de perda, em conjunto com técnicas de otimização eficientes, é determinante para a construção de modelos robustos e precisos. Por esse motivo, nesta tese, para implementação do componente de predição do *framework*, essas métricas foram fornecidas como opções para a ferramenta de otimização que definiu os metaparámetros do treinamento.

3.3.1.2 Análise Gráfica

A análise gráfica desempenha um papel fundamental na avaliação de modelos de Aprendizado de Máquina, pois complementa as métricas numéricas ao permitir a identificação visual de padrões, dispersões e anomalias nos dados e nas previsões [55]. No contexto do Orama⁺⁺, essa análise é particularmente relevante, uma vez que os modelos são utilizados para prever métricas de desempenho em ambientes FaaS, nos quais a variabilidade pode

ser influenciada por diversos fatores externos, tais como provedor de nuvem, tipo de carga e características do código.

Os gráficos do tipo *Boxplots* são amplamente utilizados para analisar a distribuição das métricas de erro ao longo de diferentes execuções experimentais. O *Boxplot* do *Root Mean Squared Error* (RMSE), na sigla em inglês [63], permite observar a magnitude dos erros penalizando desvios maiores, sendo útil para identificar cenários em que o modelo apresenta falhas significativas. De forma complementar, o *Boxplot* do MSE [63] evidencia ainda mais a presença de erros extremos, devido à natureza quadrática da métrica. Já o *Boxplot* do *Mean Absolute Percentage Error* (MAPE), na sigla em inglês [63], fornece uma visão relativa do erro, permitindo comparar o desempenho do modelo em diferentes escalas de dados, o que é especialmente importante em cenários heterogêneos.

O *Boxplot* do coeficiente de determinação (R^2) [63], por sua vez, oferece uma perspectiva complementar e particularmente relevante para a avaliação global do modelo. Diferentemente das métricas baseadas em erro absoluto [78] ou quadrático [63], o R^2 quantifica a proporção da variância dos dados observados que é explicada pelo modelo. Isso o torna especialmente adequado para cenários em que se deseja avaliar não apenas a proximidade entre valores reais e preditos, mas também a capacidade do modelo de capturar a estrutura subjacente dos dados.

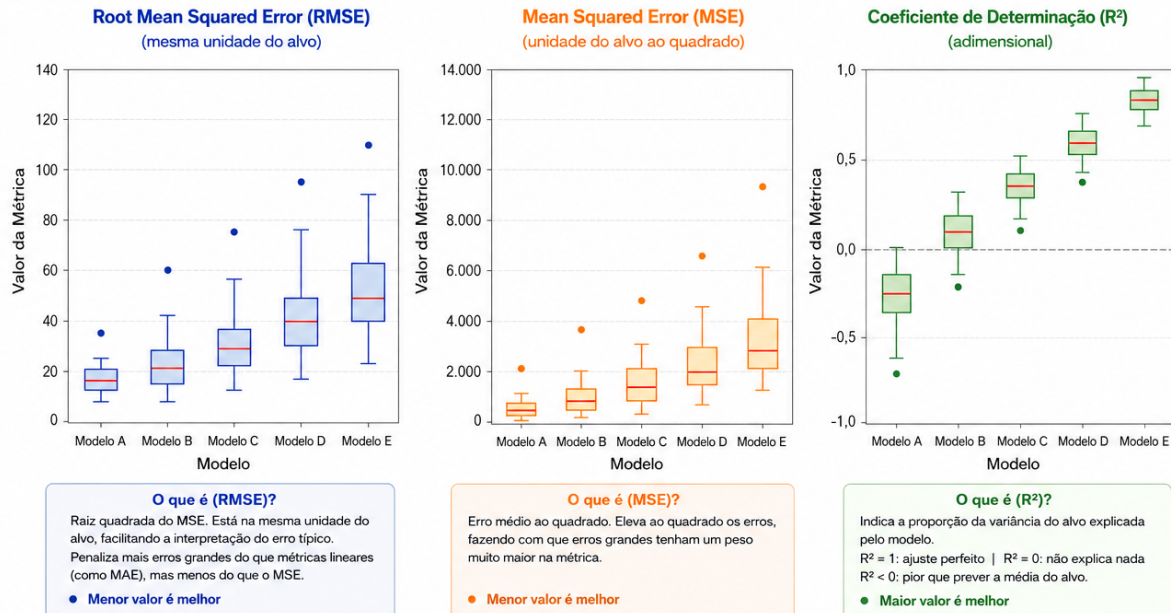


Figura 3.1: Exemplo de *boxplots* para as métricas RMSE, MSE, R^2 .

A Figura 3.1 apresenta um exemplo de visualização gráfica utilizando gráficos do tipo *Boxplot* para representar a distribuição dos resultados obtidos em diferentes modelos. Os *Boxplots* permitem identificar aspectos importantes, como mediana, dispersão, variabi-

lidade, presença de *outliers* e estabilidade das métricas ao longo das execuções. Dessa forma, eles auxiliam na comparação entre modelos, possibilitando analisar não apenas o desempenho médio, mas também a consistência e a robustez das previsões.

No contexto do Orama⁺⁺, essa característica é essencial, pois os modelos preditivos são utilizados para estimar métricas de desempenho (como tempo de execução e custo) a partir de variáveis estáticas e configurações de execução. Como esses dados frequentemente apresentam alta variabilidade, decorrente de fatores como *cold starts*, flutuações de infraestrutura e interferência de cargas concorrentes, uma métrica baseada apenas em erro pode não refletir adequadamente a qualidade do modelo. O R^2 , ao avaliar o quanto da variabilidade total é explicada, permite verificar se o modelo está efetivamente capturando os padrões relevantes do sistema, mesmo na presença de ruído.

Além disso, o uso do *Boxplot* do R^2 , ao longo de múltiplas execuções experimentais, permite analisar a estabilidade do modelo. Valores consistentemente elevados indicam que o modelo generaliza bem para diferentes cenários, enquanto alta dispersão pode indicar sensibilidade a determinadas condições experimentais. Valores negativos de R^2 , quando presentes, também são informativos, pois indicam que o modelo está performando pior do que uma abordagem trivial baseada na média, evidenciando problemas de ajuste.

Complementarmente, o gráfico de observações *versus* previsões constitui uma ferramenta essencial para avaliação qualitativa do modelo. Esse gráfico permite comparar diretamente os valores reais com os estimados, sendo idealmente esperado que os pontos se concentrem ao longo da reta identidade ($y = x$). Desvios sistemáticos dessa reta indicam padrões de erro, como tendências de subestimação ou superestimação em determinadas faixas de valores.

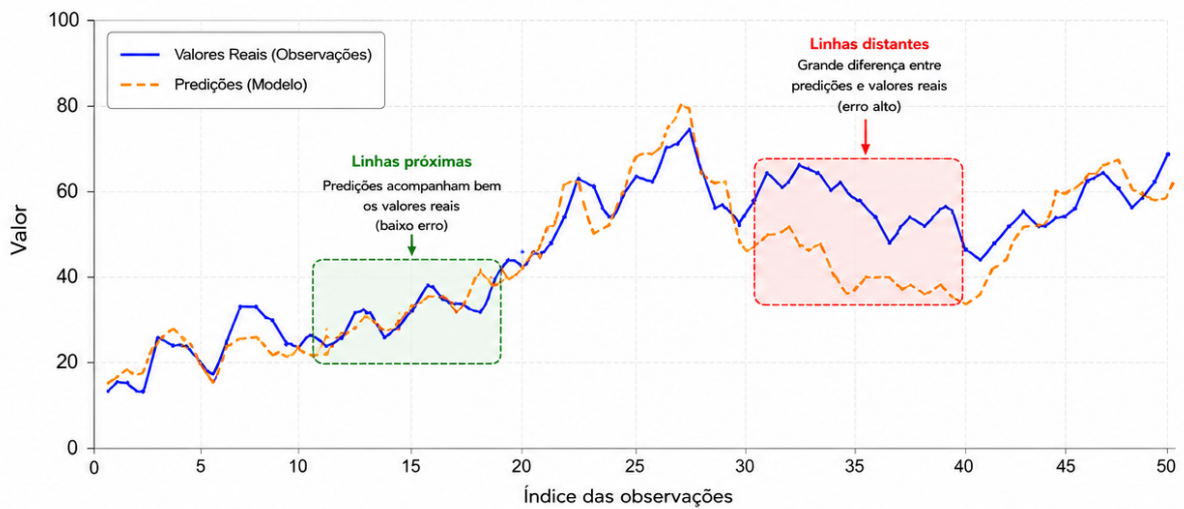


Figura 3.2: Exemplo de gráfico comparativo entre previsões e observações reais.

A Figura 3.2 apresenta um exemplo comparativo entre os valores reais observados e as previsões geradas por um modelo ao longo de 50 observações. As linhas mostram momentos em que as previsões acompanham de forma próxima os valores reais, indicando baixo erro e boa capacidade de ajuste do modelo, bem como regiões em que ocorre grande distanciamento entre as curvas, evidenciando erros mais elevados. Esse tipo de visualização é útil para analisar o comportamento do modelo ao longo das observações, permitindo identificar padrões de acerto, desvios e possíveis dificuldades de generalização.

No contexto do *framework* Orama⁺⁺, esse tipo de visualização é particularmente útil para identificar comportamentos não capturados pelos modelos, como efeitos não lineares ou limitações na representatividade das variáveis de entrada. Além disso, o gráfico auxilia na identificação de problemas clássicos como *overfitting*, quando o modelo apresenta bom desempenho apenas em subconjuntos específicos dos dados, e *underfitting*, quando não consegue capturar a dinâmica geral do sistema.

Em conjunto, essas análises gráficas fornecem uma visão abrangente da qualidade preditiva dos modelos, permitindo não apenas quantificar erros, mas também compreender suas causas e padrões. Essa abordagem é essencial no contexto do *framework* Orama⁺⁺, onde a interpretabilidade e a confiabilidade das previsões são fundamentais para apoiar decisões relacionadas à seleção de provedores e configuração de aplicações FaaS.

A partir das análises gráficas apresentadas, torna-se possível identificar limitações, padrões de erro e oportunidades de melhoria nos modelos preditivos. Esses *insights* fornecem subsídios importantes para a etapa de otimização, na qual técnicas sistemáticas são empregadas para ajustar hiperparâmetros, selecionar configurações mais adequadas e aprimorar o desempenho preditivo de forma consistente em diferentes cenários. Vale ressaltar que o modelo resultante do processo de otimização é novamente analisado usando as ferramentas descritas nesta seção. A próxima seção aborda o processo e as ferramentas comumente utilizadas para otimização de modelos preditivos.

3.3.2 Otimização de Modelos Preditivos

A fase de otimização no treinamento de modelos preditivos é essencial para refinar o desempenho e a precisão das previsões. Esse processo envolve o ajuste de hiperparâmetros, que são configurações externas ao modelo, com o objetivo de encontrar combinações que maximizem a *performance* em dados não vistos [64]. Uma estratégia de otimização bem conduzida contribui para reduzir tanto o *overfitting* quanto o *underfitting*, promovendo maior capacidade de generalização.

Diversas abordagens podem ser utilizadas para a otimização de hiperparâmetros. Métodos clássicos, como o *Grid Search* [82], realizam uma busca exaustiva sobre o espaço de parâmetros, enquanto o *Random Search* explora esse espaço de forma aleatória, frequen-

temente alcançando bons resultados com menor custo computacional. Abordagens mais avançadas incluem técnicas de otimização bayesiana [83], algoritmos evolutivos e métodos baseados em gradiente, que direcionam a busca de maneira mais eficiente.

Além das abordagens conceituais, existem ferramentas consolidadas que operacionalizam essas técnicas. O Optuna [84] oferece um mecanismo eficiente de otimização com suporte a *pruning*, permitindo interromper execuções pouco promissoras. O Ray Tune [85] destaca-se pela escalabilidade e suporte a execução distribuída em larga escala. Já o Scikit-Optimize [86] integra métodos de otimização bayesiana com o ecossistema do Scikit-Learn, facilitando sua adoção em fluxos de trabalho já estabelecidos em Python.

Nesse contexto, o Hyperopt [87] destaca-se como uma solução baseada em otimização bayesiana, utilizando algoritmos como *Tree-structured Parzen Estimators* para explorar o espaço de hiperparâmetros de forma adaptativa. Essa abordagem prioriza regiões mais promissoras ao longo das iterações, reduzindo o número de avaliações necessárias quando comparada a métodos exaustivos.

A escolha pelo Hyperopt em detrimento das demais ferramentas pode ser justificada por um conjunto de características práticas. Sua interface é relativamente simples e direta, o que reduz o esforço de integração em *pipelines* experimentais. O suporte nativo a espaços de busca complexos e hierárquicos permite modelar dependências condicionais entre hiperparâmetros de forma mais natural. Além disso, o Hyperopt apresenta menor sobrecarga operacional quando comparado a soluções voltadas à execução distribuída em larga escala, sendo adequado para experimentos locais ou moderadamente paralelos.

Outro aspecto relevante é sua flexibilidade na definição de funções de perda personalizadas, o que favorece sua aplicação em cenários experimentais e de benchmark, nos quais métricas específicas são frequentemente necessárias. O suporte à execução paralela também contribui para a redução do tempo de busca, sem exigir infraestruturas complexas. Por esse motivo, nesta tese, utilizou-se o Hyperopt como ferramenta de otimização do modelo preditivo.

Por fim, embora a escolha da técnica de otimização influencie diretamente o desempenho do modelo, seus resultados estão intrinsecamente relacionados à qualidade e à representatividade dos dados utilizados no treinamento. Assim, torna-se necessário compreender como esses dados são construídos, organizados e preparados, uma vez que tais aspectos impactam tanto o processo de otimização quanto a capacidade de generalização do modelo. Nesse contexto, a próxima seção aborda a composição dos dados de treinamento, detalhando os principais fatores que influenciam sua qualidade e adequação ao problema em análise.

3.3.3 Composição do Treinamento

O treinamento de modelos preditivos é o processo em que algoritmos de Aprendizado de Máquina ajustam seus parâmetros a partir de dados históricos, com o objetivo de identificar padrões e minimizar erros de previsão [55]. Durante o treinamento, o modelo passa por várias iterações, ajustando seus parâmetros por meio do processo de otimização. No caso de Redes Neurais, esse ajuste é feito utilizando o algoritmo de *backpropagation* [71, 77], que calcula o erro entre a saída prevista e a saída real, propagando esse erro de volta por meio das camadas da rede para atualizar os pesos dos neurônios. Técnicas como a descida do gradiente [88] e suas variantes são usadas para minimizar a função de perda, que mede a diferença entre as previsões do modelo e os valores reais.

Ao longo do treinamento, métricas como acurácia, erro médio quadrático e perda logística são monitoradas para avaliar o desempenho do modelo. Assim, caso sejam necessários, hiperparâmetros como taxa de aprendizado, número de camadas e número de neurônios, podem ser ajustados para melhorar a *performance*. Após a finalização do treinamento, o modelo é testado em um conjunto de testes independente para verificar sua capacidade de generalização. Caso os resultados sejam satisfatórios, o modelo pode ser implementado em produção para realizar previsões em novos dados. Se o desempenho não for adequado, pode ser necessário refazer algumas etapas, ajustando o conjunto de dados, alterando a arquitetura do modelo ou explorando novas técnicas de regularização para evitar problemas como *overfitting* ou *underfitting* [64].

A qualidade do processo de treinamento está diretamente condicionada à forma como os dados são preparados antes de sua utilização pelos algoritmos de Aprendizado de Máquina. Aspectos como limpeza, normalização, tratamento de valores ausentes e codificação de variáveis influenciam de maneira significativa o desempenho final do modelo, afetando tanto sua capacidade de aprendizado quanto sua generalização [60]. Dessa forma, torna-se necessário detalhar as etapas envolvidas na preparação dos dados, uma vez que elas estabelecem a base sobre a qual todo o processo de modelagem é construído. Nesse contexto, a próxima seção apresenta os principais procedimentos de tratamento dos dados e pré-processamento.

3.4 Tratamento dos Dados e Pré-processamento

A fase de pré-processamento de dados é uma etapa fundamental no desenvolvimento de modelos de IA, pois garante que as informações utilizadas para treinamento sejam consistentes, limpas e relevantes [89]. Dados brutos geralmente contêm inconsistências, valores ausentes, redundâncias e ruídos que podem comprometer a qualidade dos modelos

preditivos. Técnicas como normalização, padronização, remoção de *outliers*¹ e imputação de valores ausentes são essenciais para garantir que os algoritmos aprendam padrões reais, e não sejam influenciados por distorções nos dados [90, 91]. Dessa forma, sem um pré-processamento adequado, a IA pode gerar resultados imprecisos, enviesados ou pouco generalizáveis para novos cenários.

Além da limpeza e da organização, o pré-processamento também envolve a transformação e a extração de características que melhor representam as variáveis do problema. A seleção de atributos relevantes e a redução de dimensionalidade podem melhorar a eficiência dos algoritmos, reduzindo o tempo de processamento e aumentando a interpretabilidade dos modelos [89]. No contexto da avaliação de desempenho de sistemas, um pré-processamento bem estruturado pode facilitar a identificação de padrões em métricas, tais como tempo de resposta, uso de recursos e variações de carga, permitindo que a IA gere *insights* mais precisos e contribua para otimizações mais eficazes.

Dessa forma, investir em um pré-processamento robusto é essencial para maximizar a qualidade dos modelos e garantir decisões mais confiáveis baseadas em IA. Para isso, diversos procedimentos precisam ser realizados. Tais procedimentos serão detalhados nas próximas seções.

3.4.1 Tratamento de Valores Ausentes

O tratamento de valores ausentes [92] é uma etapa essencial no pré-processamento de dados, pois a ausência de informações pode comprometer a qualidade e a precisão dos modelos de IA. Dependendo do volume e do padrão de ausência, diferentes estratégias podem ser aplicadas, como remoção de registros incompletos, imputação com medidas estatísticas (média, mediana ou moda) ou o uso de algoritmos avançados que estimam valores com base em padrões observados nos dados.

A escolha do método adequado depende do contexto e da distribuição dos dados, uma vez que a remoção excessiva pode levar à perda de informações valiosas, enquanto a imputação inadequada pode introduzir viés nos resultados. Em aplicações como a avaliação de desempenho de sistemas, lidar corretamente com valores ausentes é fundamental para garantir que análises e previsões sejam baseadas em um conjunto de dados confiável e representativo da realidade [92].

¹*Outliers* são observações que apresentam comportamento significativamente diferente do padrão predominante dos dados, podendo indicar anomalias, erros ou eventos raros em modelos de IA [90].

3.4.2 Categorização

A categorização [93] é outro processo essencial no pré-processamento de dados que consiste em transformar variáveis qualitativas em representações estruturadas para serem utilizadas por algoritmos de Aprendizado de Máquina. Em ambientes de Computação em Nuvem, muitas métricas e configurações podem ser categorizadas para facilitar análises e previsões. Por exemplo, provedores de nuvem podem ser categorizados em AWS, Azure, GCP e Alibaba Cloud; tipos de instâncias podem ser classificados em pequenas, médias e grandes; e estados de execução de funções *serverless* podem ser rotulados como bem-sucedido, com erro ou com *timeout*. A categorização correta permite que modelos de IA identifiquem padrões e tomem decisões mais adequadas, tornando a análise de desempenho em nuvem mais eficiente e interpretável.

Além disso, a escolha adequada das categorias e a definição precisa de seus limites impactam diretamente na qualidade das análises realizadas. Categorias mal definidas ou excessivamente generalizadas podem levar à perda de informações relevantes, reduzindo a capacidade dos modelos de IA de identificar padrões significativos. Por outro lado, categorizações excessivamente específicas podem resultar em um aumento da complexidade computacional e em uma possível sobrecarga de dados, dificultando a generalização dos modelos. Dessa forma, diversas técnicas são frequentemente utilizadas para transformar variáveis categóricas em formatos numéricos compatíveis com algoritmos de Aprendizado de Máquina.

3.4.3 Normalização

A normalização [91] é uma técnica que ajusta escalas de variáveis para garantir que todas possuam a mesma influência nos modelos de Aprendizado de Máquina. Uma das abordagens mais comuns é a normalização pelo $z - score$ [94], que transforma os valores de uma variável em uma distribuição com média zero e desvio padrão igual a um. Isso é particularmente útil quando os dados possuem diferentes ordens de grandeza ou distribuições distintas, como no caso da avaliação de desempenho em nuvem, no qual métricas como tempo de resposta, consumo de memória e número de requisições podem variar significativamente. Aplicar a normalização pelo $z - score$ melhora a estabilidade numérica dos algoritmos, evita viés devido a escalas discrepantes e facilita a convergência de modelos que dependem de gradientes para otimização [94].

Além do $z - score$, outra técnica amplamente utilizada é a normalização $min - max$ [95], que reescala os valores de uma variável para um intervalo específico, geralmente, entre 0 e 1. Essa abordagem é especialmente útil quando os dados possuem limites conhecidos e se deseja preservar a interpretabilidade relativa das magnitudes originais. No contexto da

avaliação de desempenho em nuvem, a normalização *min – max* pode ser vantajosa para métricas como a latência de resposta, permitindo comparações diretas entre diferentes instâncias e serviços. No entanto, essa técnica pode ser sensível a *outliers*, pois valores extremos afetam a escala dos demais dados. Assim, a escolha entre normalização pelo *z – score* ou *min – max* depende do tipo de distribuição das variáveis e das características do modelo de Aprendizado de Máquina empregado [91].

3.4.4 Tratamento de *Outliers*

O tratamento de *outliers* [90] é uma etapa crucial no pré-processamento de dados, pois valores extremamente discrepantes podem distorcer análises e comprometer a precisão dos modelos de Aprendizado de Máquina. Esses valores podem surgir por erros de medição, falhas no sistema ou condições excepcionais do ambiente e, se não forem tratados adequadamente, podem influenciar estatísticas como médias e desvios padrão, além de afetar a convergência de algoritmos.

Métodos comuns para lidar com *outliers* incluem a remoção de registros que ultrapassam limites definidos por técnicas como o intervalo interquartil (*Interquartile Range* (IQR), na sigla em inglês) [96] e a normalização pelo *z – score*, bem como a substituição desses valores por estimativas baseadas na distribuição dos dados. Além disso, a winsorização [97] é outra abordagem eficiente, na qual os valores extremos são substituídos por limites definidos, preservando o tamanho da amostra e mitigando o impacto de *outliers* sem descartá-los.

3.4.5 Análise de Correlação

A análise de correlação [98] permite identificar relações entre variáveis e orientar a seleção de atributos relevantes para modelos de Aprendizado de Máquina. Ao avaliar a dependência entre diferentes características é possível reduzir redundâncias, eliminando variáveis altamente correlacionadas que não agregam novas informações ao modelo. Métodos como os coeficientes de Pearson [99], Spearman [99] e Kendall [99] ajudam a quantificar essas relações, diferenciando associações lineares e não lineares. No contexto do pré-processamento, a análise de correlação é útil para evitar problemas como multicolinearidade em modelos preditivos, garantindo que os dados utilizados sejam informativos e contribuam para a robustez das análises.

Além da identificação de relações entre variáveis, a análise de correlação também pode auxiliar na engenharia de atributos, orientando a criação de novas variáveis que capturem melhor as interações presentes nos dados. Quando duas ou mais variáveis apresentam correlação moderada, pode ser benéfico combiná-las por meio de transformações mate-

máticas, como somas, multiplicações ou componentes principais, para extrair informações latentes. Ademais, em aplicações práticas de Aprendizado de Máquina, a seleção criteriosa de atributos baseada na correlação pode melhorar a interpretabilidade dos modelos, reduzindo a complexidade computacional e minimizando o risco de *overfitting*. Dessa forma, ao integrar a análise de correlação no *pipeline* de pré-processamento, os modelos tornam-se mais eficientes, generalizáveis e alinhados às particularidades dos dados utilizados.

3.4.6 Redução de Escala

A redução de escala [100] é útil quando se lida com grandes volumes de informações e múltiplas variáveis que podem ter escalas ou unidades diferentes. Uma das técnicas mais utilizadas para reduzir a dimensionalidade dos dados e facilitar a análise é a Análise de Componentes Principais (*Principal Component Analysis* (PCA), na sigla em inglês) [100].

O PCA transforma as variáveis originais em um conjunto menor de variáveis, chamadas componentes principais, que explicam a maior parte da variabilidade dos dados. Essa técnica não só simplifica os dados, removendo redundâncias, como também pode melhorar a eficiência dos modelos de Aprendizado de Máquina, já que reduz o risco de sobreajuste e acelera o tempo de treinamento.

Além disso, a escolha do número adequado de componentes principais é uma etapa crucial na aplicação do PCA, pois impacta diretamente no equilíbrio entre a redução de dimensionalidade e a preservação da variabilidade dos dados. Normalmente, utiliza-se a análise do gráfico de variância explicada ou a técnica do limiar acumulado, que define um percentual mínimo de variância a ser mantido, como 95% ou 99%.

Outra vantagem do PCA é sua aplicabilidade em diferentes domínios, como processamento de imagens, compressão de dados e Bioinformática, nos quais a simplificação das variáveis pode revelar padrões ocultos e facilitar a interpretação dos dados. Contudo, é importante destacar que o PCA assume que as relações entre as variáveis são lineares, o que pode limitar sua eficácia em alguns contextos nos quais métodos não lineares podem ser mais apropriados.

3.4.7 Remoção de *Spikes*

A remoção de *spikes*² [101] é uma técnica utilizada para eliminar valores anômalos ou picos inesperados nos dados, que podem distorcer análises e prejudicar a construção de modelos de Aprendizado de Máquina. *Spikes* são frequentemente causados por erros de

²*Spikes* são variações abruptas e pontuais nos dados, caracterizadas por aumentos ou quedas repentinas que se desviam do comportamento esperado ao longo do tempo [101].

medição, falhas temporárias no sistema ou flutuações incomuns que não representam o comportamento normal dos dados.

Uma abordagem comum para suavizar essas variações extremas é o uso da média ponderada móvel (*Weighted Moving Average* (WMA), na sigla em inglês) [101], que calcula a média de um conjunto de pontos de dados, atribuindo pesos maiores aos valores mais recentes ou mais relevantes. Essa técnica ajuda a atenuar os efeitos de *spikes* ao suavizar a série temporal ou as métricas analisadas, mantendo a integridade dos dados e melhorando a precisão dos modelos, especialmente em cenários de monitoramento de desempenho, como na análise de métricas de sistemas em nuvem.

As técnicas apresentadas nesta seção constituem etapas fundamentais para garantir a qualidade e a adequação dos dados utilizados em análises posteriores. O tratamento de *outliers* e *spikes*, a análise de correlação e a redução de escala contribuem para a construção de conjuntos de dados mais consistentes, informativos e livres de ruídos, reduzindo vieses e redundâncias que podem comprometer os resultados [91]. Ao preparar os dados de forma estruturada e criteriosa, cria-se uma base sólida para a etapa de modelagem preditiva, na qual algoritmos de Aprendizado de Máquina são aplicados com maior eficiência, estabilidade e capacidade de generalização, potencializando a extração de padrões e a geração de previsões mais precisas.

3.5 Considerações Finais

Este capítulo abordou a IA de forma ampla, iniciando com uma contextualização do tema e avançando pelos fundamentos de IA, especialmente Aprendizado de Máquina. Também foram apresentadas técnicas de modelagem preditiva, métricas de avaliação, funções de perda, análises gráficas de desempenho e estratégias de otimização e ajuste fino dos modelos.

O pré-processamento foi destacado como etapa fundamental, incluindo tratamento de valores ausentes, categorização, normalização e análise de *outliers*, visando garantir a qualidade dos dados utilizados em modelos preditivos. Além disso, técnicas como análise de correlação, redução de escala e remoção de *spikes* foram exploradas para otimizar a representação dos dados e melhorar o desempenho dos algoritmos de IA.

Conforme discutido neste capítulo, as técnicas de IA são fundamentais para os objetivos deste trabalho, especialmente na implementação de um modelo preditivo capaz de estimar o tempo de execução e o custo de soluções orientadas a FaaS. Essas técnicas permitem extrair conhecimento dos dados de desempenho dos provedores de FaaS, contribuindo para a construção de preditores que apoiem processos decisórios nesse contexto. Assim, o próximo capítulo apresentará o *framework* proposto nesta tese.

Capítulo 4

Framework Orama⁺⁺

Este capítulo apresenta uma análise abrangente do *Framework* Orama⁺⁺, explorando seus principais conceitos, funcionamento e aplicações. Inicialmente, a Seção 4.1 define o *framework*, destacando seus objetivos e principais características. A Seção 4.2 detalha a arquitetura do Orama⁺⁺, explicando seus componentes e interações internas. Na Seção 4.3 é detalhado o cálculo para composição dos custos. Em seguida, a Seção 4.4 descreve seu fluxo de trabalho, detalhando as etapas envolvidas em sua operação. Na sequência, a Seção 4.5 aborda os casos de uso pré-configurados, demonstrando as aplicações práticas do *framework*. A Seção 4.6 explora os recursos de análise estatística disponíveis, enquanto a Seção 4.7 discute o modelo preditivo embutido no Orama⁺⁺. Além disso, a Seção 4.8 apresenta as tecnologias empregadas no desenvolvimento do *framework*. Em seguida, a Seção 4.9 apresenta os passos necessários para a implantação do Orama⁺⁺, enquanto a Seção 4.10 apresenta as suas evoluções e a Seção 4.11 aborda as limitações da ferramenta. Por fim, a Seção 4.12 reúne as considerações finais, sintetizando os pontos-chave abordados ao longo deste capítulo.

4.1 Definição

Com o objetivo de auxiliar os processos de tomada de decisão envolvendo a implantação de aplicações FaaS em diferentes provedores de nuvem, neste trabalho é apresentado o *framework* Orama⁺⁺. Esse *framework*, por meio de IA, é capaz de estimar o tempo de execução e o custo de uma função, em diferentes provedores, e perante diversos níveis de concorrência.

Além disso, o *framework* permite a execução e a avaliação estatística de *benchmarks* em ambientes orientados a FaaS, seja por meio de casos de uso pré-configurados ou sobre ambientes customizados. No contexto do Orama⁺⁺, um *benchmark* corresponde a um conjunto de metaparámetros que define como os casos de uso (aplicações) devem ser exe-

cutados e avaliados experimentalmente. Esses metaparámetros especificam, por exemplo, a quantidade de requisições concorrentes submetidas às funções, o número de repetições de cada bateria de testes para composição de amostras estatisticamente representativas e os intervalos entre requisições, permitindo a análise de fenômenos característicos de ambientes FaaS, como o *cold start*. O *framework* também possibilita a execução de requisições de aquecimento (*warm-up*), utilizadas para capturar e isolar o impacto do *cold start* nos resultados experimentais. Dessa forma, os *benchmarks* fornecem uma configuração padronizada e reproduzível para a avaliação de desempenho, custo e comportamento de aplicações em diferentes provedores e cenários de execução.

“*Orama*” é uma palavra de origem grega que significa “Visão”. O nome escolhido para o *framework* demonstra a intenção de ampliar a visão sobre o comportamento de ambientes FaaS e, principalmente, de enxergar o futuro por meio de previsões.

O *framework* Orama⁺⁺ consiste em uma composição de diversas soluções coordenadas e orquestradas para implantar casos de uso padronizados em serviços de FaaS em diversos provedores, tais como AWS Lambda, GCF, AZF e AFC. Além disso, o *framework* permite executar *benchmarks* padronizados por meio do acionamento sistemático das APIs disponibilizadas pelos ambientes de FaaS. Uma vez coletados os resultados dos *benchmarks*, é possível analisar os resultados usando técnicas estatísticas disponibilizadas pelo *framework*. Por último, o Orama⁺⁺ utiliza o conhecimento adquirido acerca do desempenho dos provedores para estimar tempos de execução em cada um dos provedores, por meio de Redes Neurais otimizadas para realizar previsões a partir de métricas coletadas no código-fonte de uma função. As principais características do *framework* são:

- Código aberto;
- Casos de uso embutidos;
- Provisionamento e desprovisionamento automático de casos de uso embutidos;
- Configuração de casos de uso personalizados (sem provisionamento);
- Execução simultânea de *benchmarks*, controlada por semáforo;
- Possibilidade de configurar *worker* distribuídos, permitindo o acionamento das APIs em diferentes localidades;
- Resultados detalhados das execuções de *benchmarks*;
- Possibilidade de reexecução de *benchmarks*;
- Controle do número de requisições feitas pelos *benchmarks*;
- Opção de requisição adicional para contornar o *cold start*;

- Módulo de projeto *fatorial* 2^k para análise estatística entre dois *benchmarks*;
- Teste *t* pareado para analisar a diferença entre dois *benchmarks*;
- Módulo de predição de tempo de execução que estima o tempo de execução em diferentes provedores a partir do código-fonte de uma função FaaS;
- Predição do custo, considerando a composição de cobrança para FaaS em cada provedor.

4.2 Arquitetura

Para atender aos propósitos pretendidos pelo *framework* Orama⁺⁺, foi desenvolvida uma solução arquitetural composta por nove componentes, os quais são: *Frontend*, *Backend*, *Database*, *Cloud Orchestrator*, *Benchmark*, *Remote Benchmark*, Apache Kafka, *Halsteader* e *Predictor*, conforme mostrado na Figura 4.1. Os componentes do *framework* foram projetados para serem executados sobre um *runtime* de contêineres. A maior parte dos componentes se comunica via protocolo HTTP, enquanto o acionamento da execução do *benchmark* é processada por *streams* de dados. As interações com os provedores de nuvem também ocorrem via protocolo HTTP, gerenciadas por um orquestrador de nuvem. Os componentes do *framework* Orama⁺⁺ são detalhados nas próximas seções.

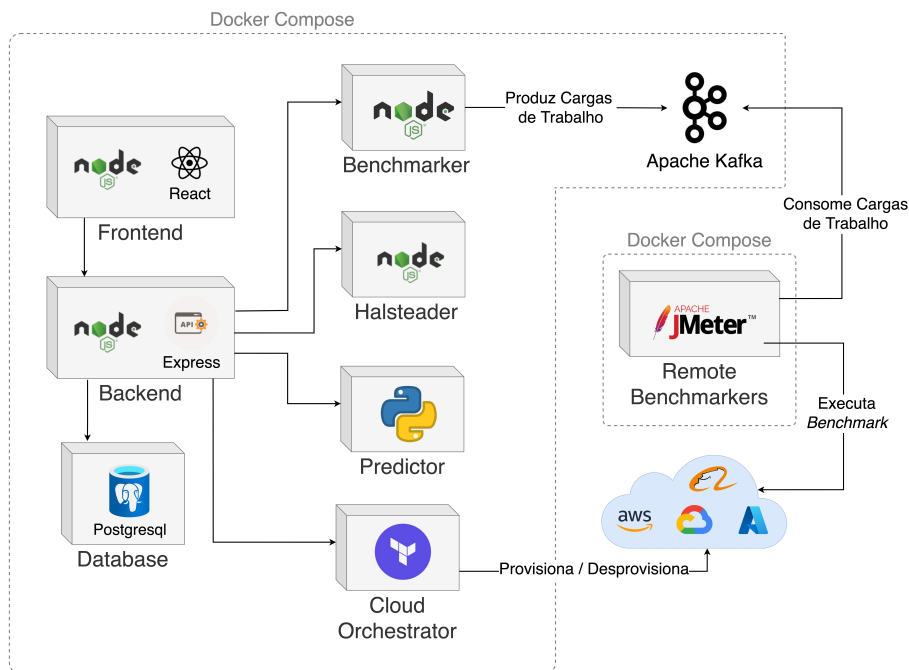


Figura 4.1: Visão dos componentes do *framework* Orama⁺⁺.

4.2.1 *Frontend*

O *Frontend* consiste em uma aplicação desenvolvida em JavaScript [102] utilizando os *frameworks* React [103] e Material UI [104], cujo objetivo é fornecer uma interface gráfica amigável entre o usuário e o *framework*, permitindo o gerenciamento de ativos da aplicação, tais como provedores, casos de uso, *benchmarks* e projetos fatoriais.

A escolha das tecnologias utilizadas no *Frontend* foi motivada pela necessidade de construir uma interface modular, responsiva e de fácil manutenção. O React [103] foi adotado devido ao seu modelo baseado em componentes reutilizáveis, o que facilita a implementação de páginas independentes para gerenciamento de provedores, casos de uso, *benchmarks* e análises estatísticas, além de favorecer a evolução incremental da ferramenta. O Material UI [104] foi utilizado para padronizar os elementos visuais da aplicação e acelerar o desenvolvimento da interface gráfica, fornecendo componentes prontos compatíveis com princípios modernos de usabilidade e responsividade. Já o JavaScript [102] foi escolhido por sua ampla adoção no desenvolvimento de aplicações web e pela integração nativa com o ecossistema React, permitindo maior flexibilidade na comunicação entre o *Frontend* e os serviços disponibilizados pelos demais componentes do *framework*. Essas decisões contribuem para a construção de uma interface acessível, extensível e adequada à operacionalização dos experimentos e análises conduzidos pelo Orama.

A partir do *Frontend* também é possível realizar a execução dos processos de provisionamento e desprovisionamento, bem como a execução de *benchmarks* e a visualização dos resultados das análises estatísticas. A ferramenta de predição também é operacionalizada por meio deste componente, que disponibiliza uma página para escrita da função e apresentação das estimativas de tempo de execução, além das estimativas de custo em cada provedor que o *framework* possui integração, inclusive em diferentes perfis de oferta dos serviços (*tiers*).

4.2.2 *Backend*

O *Backend* consiste em uma aplicação desenvolvida em Node.js [105] utilizando o *framework* Express.js [106], cuja finalidade é fornecer uma API centralizada para o *Frontend* (*gateway*), garantindo que todas as requisições sejam tratadas por esse componente antes de serem redirecionadas para outro serviço dentro do ecossistema do Orama⁺⁺. Este componente também é responsável por interagir com o banco de dados da aplicação. A API do *Backend* é documentada e pode ser acionada por meio de uma interface gráfica gerada pelo Swagger [107].

A escolha das tecnologias utilizadas no *Backend* foi motivada pela necessidade de implementar um componente centralizado capaz de processar requisições concorrentes,

integrar múltiplos serviços e coordenar operações distribuídas de forma eficiente. O Node.js [105] foi adotado devido ao seu modelo de execução assíncrona e orientado a eventos, que favorece aplicações com grande volume de operações de entrada e saída, como chamadas de API, acesso ao banco de dados e comunicação com provedores de nuvem. O *framework* Express.js [106] foi utilizado por oferecer uma estrutura leve e flexível para construção de serviços RESTful [108], simplificando a implementação das rotas, mecanismos de autenticação, regras de negócio e integração entre os componentes do ecossistema Orama⁺⁺. A utilização do Swagger contribui para a padronização e documentação da API, facilitando sua utilização, manutenção e evolução. Essas decisões tecnológicas permitem que o *Backend* atue como um núcleo de orquestração escalável e extensível, responsável por coordenar os experimentos, consolidar os resultados e operacionalizar os mecanismos de predição e estimativa de custo do *framework*.

As regras de negócio do *framework* estão concentradas no *Backend* e controlam desde os aspectos básicos do sistema como a configuração de provedores (inclusive a composição do custo), casos de uso, *benchmarks* e projetos fatoriais, a até a orquestração da execução dos testes e a coleta e consolidação dos dados. É tarefa do *Backend* a conversão do código-fonte de uma função (fornecida) em métricas e a solicitação da predição do tempo de execução em um determinado provedor, a partir de um nível de concorrência definido. O *Backend* é, ainda, encarregado que calcular as estimativas de custo, usando como base nas estimativas de tempo de execução e composição do custo em cada provedor. A maioria das operações realizadas por este componente interage, de alguma forma, com o *Database*.

4.2.3 *Database*

O *Database* é a camada de armazenamento de dados do *framework*, composta pelo banco de dados PostgreSQL [109] e o modelo de dados (mostrado na Figura 4.2), devidamente aplicado. Essa camada recebe e armazena os dados transacionais gerados pelos demais componentes da solução.

Por meio do *framework* Knex.js [110] ocorre um mapeamento objeto relacional [111], que é uma técnica de programação que permite a conversão automática entre objetos de uma aplicação e tabelas de um banco de dados relacional, facilitando a manipulação de dados sem a necessidade de escrever *Structured Query Language* (SQL) manualmente. Esse mapeamento ocorre no *Backend* e permite que os objetos manipulados por ele sejam persistidos no *Database*. Além disso, há uma definição de *migrations* que popula os dados iniciais do banco de dados.

Esse mecanismo permite que a plataforma seja inicializada com dados pré-configurados para os diferentes aspectos, tais como provedores, casos de uso e *benchmarks*. Dessa forma, as versões mais recentes do *framework* podem agregar novas configurações de *benchmarks*

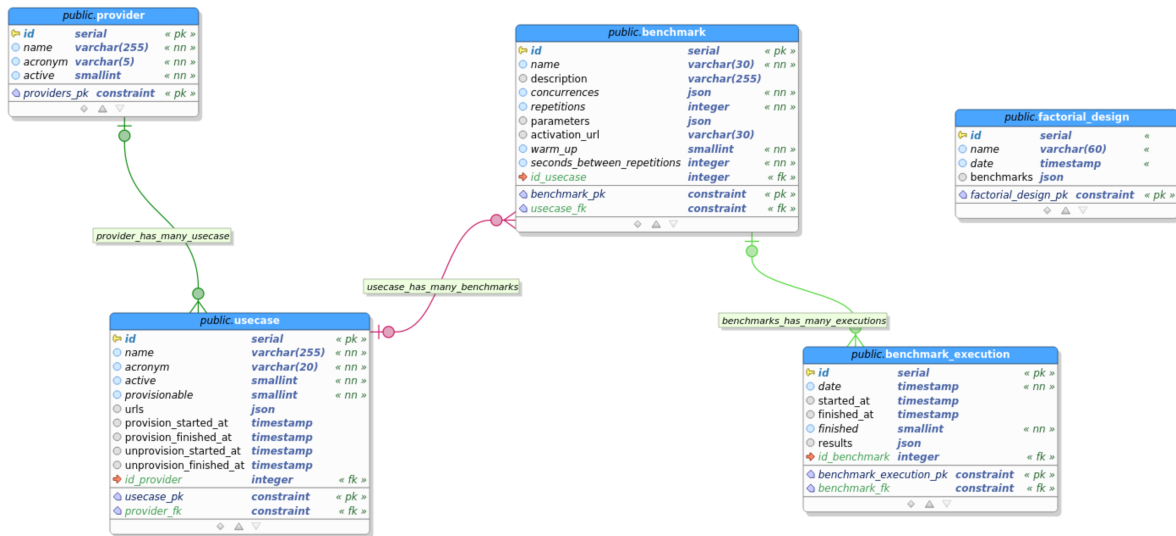


Figura 4.2: Modelo de dados do *framework* Orama⁺⁺.

sem, necessariamente, alterar a estrutura do *framework*. Por outro lado, para inclusão de suporte a provedores ou novos casos de uso, é necessário incluir artefatos de automação ao arcabouço do *framework*.

A escolha do PostgreSQL [112] foi motivada pela necessidade de um banco de dados relacional robusto e confiável para armazenar os dados transacionais e experimentais do *framework*. Sua capacidade de garantir integridade, concorrência e escalabilidade o torna adequado ao gerenciamento dos resultados dos *benchmarks*. O Knex.js [110] foi adotado para simplificar a persistência de dados por meio do mapeamento objeto relacional e facilitar a manutenção do *Backend*. Já o uso de *migrations* contribui para a reprodutibilidade e evolução controlada da plataforma, permitindo versionar alterações estruturais e inicializar automaticamente configurações padronizadas do Orama⁺⁺.

4.2.4 Cloud Orchestrator

O *Cloud Orchestrator* é uma camada de abstração para o provisionamento em diferentes provedores. Consiste em uma aplicação desenvolvida em Node.js utilizando o *framework* Express.js, cujo objetivo é fornecer uma API para o provisionamento e o desprovisionamento do ambiente por meio do Terraform [113].

Esse componente requer a existência dos artefatos de definição da infraestrutura de um caso de uso para aplicá-los no provedor selecionado. Os casos de uso podem ser facilmente estendidos apenas criando novas estruturas de diretório com os respectivos arquivos de definição do Terraform, e a configuração adequada do *framework* para acioná-los. A API do *Cloud Orchestrator* também é documentada e pode ser acionada por meio de uma interface gráfica gerada pelo Swagger [107], assim como o *Backend*.

A escolha do Node.js e do Express.js no *Cloud Orchestrator* foi motivada pela necessidade de disponibilizar uma camada leve e extensível para automação do provisionamento em múltiplos provedores. O Terraform foi adotado por permitir a definição declarativa da infraestrutura e por oferecer portabilidade entre plataformas de nuvem, facilitando a padronização e reprodutibilidade dos ambientes experimentais. Já o uso do Swagger contribui para a documentação e utilização da API, simplificando a integração entre os componentes do Orama⁺⁺ e a inclusão de novos casos de uso.

4.2.5 *Benchmark*

O *Benchmark* é o componente responsável por coordenar a execução dos *benchmarks*. Ele consiste em uma aplicação desenvolvida em Node.js utilizando o *framework* Express.js, cujo objetivo é fornecer uma API para a execução de *benchmarks*. Vale ressaltar que o *Benchmark* não executa diretamente as requisições, de modo a não engargalar o fluxo de execução dos testes sobre os ambientes FaaS. Ao receber uma solicitação de execução de *benchmarks* vinda do *Backend*, o *Benchmark* analisa os parâmetros da solicitação e define a estratégia de execução. Para definir essa estratégia, o *Benchmark* verifica quais *Remote Benchmarks* estão ativos (conforme configurado no *framework*) e saudáveis naquele momento (enviaram mensagens recentemente) e, então, divide as rajadas¹ entre esses *workers*, conforme mostrado na Figura 4.3.

Para disparar uma rodada de um *benchmark*, o *Benchmark* inclui mensagens de acionamento das respectivas URLs e níveis de concorrência em tópicos no Apache Kafka [114], que são monitorados pelos *Remote Benchmarks*. A partir desse momento, o *Benchmark* passa a monitorar outro tópico no Apache Kafka no qual os *workers* vão publicar os seus resultados parciais. Quando todos os resultados parciais são recebidos, o *Benchmark* os consolida e registra o resultado final da execução no *Database*, finalizando aquela rodada do *benchmark*.

O *Benchmark* também gera uma página web com métricas dos testes a partir de um arquivo CSV, acessível pelo *Frontend* após a execução do *benchmark*. O componente permite que múltiplos *benchmarks* sejam submetidos simultaneamente. Entretanto, para evitar interferência entre os experimentos e garantir o isolamento das execuções, o *framework* controla o acesso aos recursos de execução por meio de um semáforo, assegurando que apenas um *benchmark* seja executado por vez. Dessa forma, evita-se que a execução de um experimento afete os resultados dos demais. Tal qual as APIs do *Backend* e do

¹Uma rajada consiste em um conjunto de requisições que são feitas a um serviço de FaaS de forma concorrente. Cada *worker* realiza uma parte das requisições que devem fazer parte de um teste. Por exemplo, caso o teste esteja configurado para acionar o serviço com 512 requisições concorrentes e 4 *workers* estejam ativos e saudáveis, então cada *worker* ficará responsável por realizar 128 requisições.

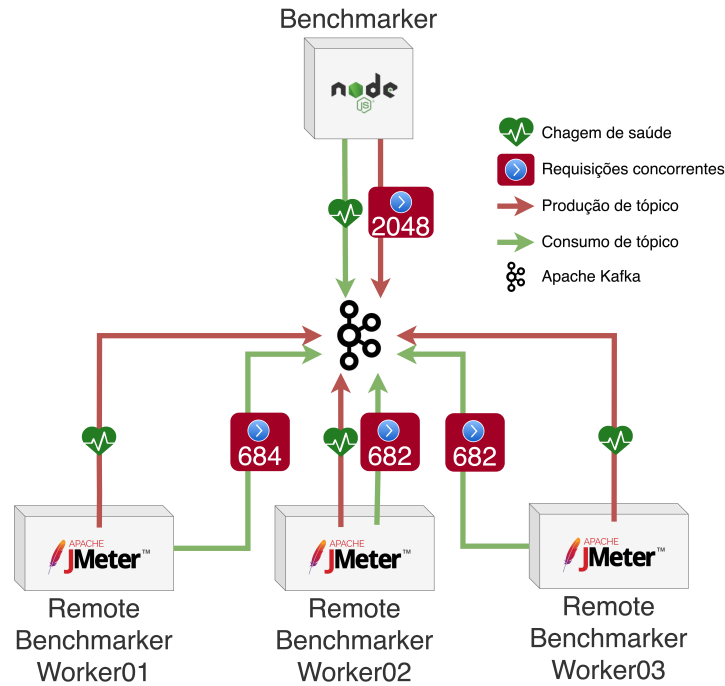


Figura 4.3: Exemplo de distribuição de requisições entre os *Remote Benchmarkers*.

Cloud Orchestrator, a API do *Benchmarker* também é documentada e pode ser acionada por meio de uma interface gráfica gerada pelo Swagger [107].

A escolha do Node.js e do Express.js no *Benchmarker* foi motivada pela necessidade de coordenar execuções concorrentes e distribuídas de forma eficiente. O Apache Kafka foi adotado como mecanismo de mensageria devido à sua arquitetura distribuída, alta escalabilidade horizontal e capacidade de processar grandes volumes de mensagens com baixa latência, características importantes para a distribuição das rajadas de requisições entre os *Remote Benchmarkers* e consolidação assíncrona dos resultados.

4.2.6 *Remote Benchmark*

O *Remote Benchmark* é uma aplicação desenvolvida em Node.js que atua como um executor remoto (*worker*) de requisições sobre uma API. Ele pode ser instanciado no mesmo ambiente no qual os demais componentes do *framework* Orama⁺⁺ são instanciados e/ou em ambientes separados. Isso permite que a dinâmica dos *benchmarks* seja bastante flexível, já que a origem dos disparos pode ser estrategicamente definida para avaliar algum determinado aspecto, como o comportamento de serviços em regiões geográficas específicas ou o desempenho perante acionamentos geograficamente dispersos, por exemplo.

Internamente, o *Remote Benchmark* utiliza a ferramenta JMeter [115] para executar as requisições. Para isso, ele possui um artefato genérico de execução de *benchmarks* para o JMeter. Esse artefato é associado a diversos parâmetros enviados pelo *Benchmark*,

tais como URL, quantidade de requisições e parâmetros para envio junto às requisições (*payload*). Os resultados gerados pelo JMeter são coletados e encaminhados para o *Benchmark* por meio do Apache Kafka.

A escolha do Node.js no *Remote Benchmark* foi motivada pela necessidade de implementar um componente leve, portátil e facilmente distribuível em diferentes ambientes de execução. O Apache JMeter foi adotado devido à sua ampla utilização em testes de carga e desempenho, além de oferecer mecanismos robustos para geração de requisições concorrentes, coleta de métricas e parametrização dinâmica dos testes. Sua utilização também reduz a necessidade de implementação manual de mecanismos de geração de carga, permitindo maior confiabilidade e reprodutibilidade experimental.

4.2.7 Apache Kafka

O Apache Kafka [114] é o gerenciador de *stream* de dados que intermedeia a comunicação entre o *Benchmark* e os *Remote Benchmarks*. Por utilizar um protocolo mais leve do que o HTTP, esse *stream* confere maior velocidade no disparo das requisições para os serviços alvo, mitigando atrasos de comunicação que podem ocorrer no protocolo HTTP, conferindo ao *benchmark*, executado por meio desse mecanismo, maior fidedignidade. Assim sendo, são configurados três tipos de tópicos no Apache Kafka [114], conforme descritos a seguir:

- **Checagem de saúde:** quando um *Remote Benchmark* é configurado, então é criado um tópico específico para monitorar a sua saúde. Periodicamente, esse *worker* inclui uma mensagem de *status* da sua saúde nesse tópico. O *Backend* fica subscrito ao tópico de saúde daquele *worker* e, com isso, recebe a notificação de saúde e registra no *Database*;
- **Execução de rajadas de requisição:** para cada *Remote Benchmark* é criado um tópico para que o *Benchmark* informe o *worker* que ele deve executar uma rajada de requisições concorrentes sobre uma URL. O nível de concorrência, a URL e o *payload* são parâmetros que também são incluídos neste tópico;
- **Retorno de resultados:** uma vez que o *worker* tenha sido acionado para realizar uma rajada de requisições, seus resultados são colocados em um tópico específico para que o *Benchmark* possa receber os resultados parciais de uma execução. Conforme os *workers* incluem seus resultados parciais nesse tópico, o *Benchmark* os recebe para, no final, realizar a consolidação.

Diferentemente de soluções tradicionais de mensageria, como o RabbitMQ [116], que são mais voltadas para filas transacionais e roteamento complexo, o Kafka apresenta

melhor desempenho em cenários de alto volume e processamento contínuo de eventos, além de oferecer mecanismos nativos de persistência, tolerância a falhas e replicação distribuída [117]. Essas características tornam o Kafka mais adequado para suportar a execução escalável e reproduzível dos *benchmarks* no Orama⁺⁺.

4.2.8 *Halsteader*

A predição de tempo de execução em FaaS a partir do código-fonte permite otimizar o desempenho e o custo de aplicações *serverless*. Para realizar essa predição, é necessário extrair métricas relevantes do código-fonte, que servirão como base para o treinamento de modelos preditivos. Essas métricas podem incluir informações sobre a complexidade do código, o número de operações, o tamanho do código e outros fatores que influenciam o tempo de execução. Uma das métricas amplamente utilizadas nesse contexto é o conjunto de métricas de Halstead [16], que fornece uma medida quantitativa da complexidade do software.

As métricas de Halstead, desenvolvidas por Maurice Halstead [16], são um conjunto de medidas de software que avaliam a complexidade e o esforço de programação com base em operadores e operandos presentes no código-fonte. Essas métricas fornecem informações valiosas sobre o tamanho, a dificuldade e o esforço de implementação de um programa, permitindo estimar o tempo de desenvolvimento, o número de erros e outros aspectos relacionados à qualidade do software. As métricas de Halstead são calculadas a partir de quatro medidas básicas: número de operadores distintos, número total de operadores, número de operandos distintos e número total de operandos.

As métricas de Halstead são úteis para diversas finalidades, como estimar o esforço de programação, avaliar a complexidade do código, identificar áreas de alto risco e prever o número de erros. Além disso, essas métricas podem ser utilizadas para comparar diferentes implementações de um mesmo algoritmo, auxiliando na escolha da solução mais eficiente. No contexto da predição de tempo de execução em FaaS, as métricas de Halstead podem ser combinadas com outras métricas de código-fonte e características da função para treinar modelos de Aprendizado de Máquina capazes de prever o tempo de execução com precisão, semelhante a abordagens para predição de falhas em software [118].

Além das métricas de Halstead, este trabalho também considera o uso de métricas baseadas no indicador de qualidade da TIOBE [119]. Essas métricas são voltadas à avaliação da qualidade de código-fonte, abrangendo aspectos como complexidade, duplicação, tamanho de métodos, cobertura de testes, acoplamento e aderência a boas práticas de desenvolvimento. Diferentemente das métricas de Halstead, que se concentram na estrutura léxica do código, as métricas TIOBE fornecem uma visão mais ampla da qualidade

estrutural e organizacional do software, sendo frequentemente utilizadas em processos de análise estática para identificação de problemas de manutenibilidade e confiabilidade.

No contexto deste trabalho, foram realizadas comparações experimentais entre diferentes configurações de entrada para o modelo de predição baseado em Redes Neurais [120]: (i) utilizando apenas métricas de Halstead, (ii) utilizando apenas métricas TIOBE e (iii) utilizando a combinação de ambas. Os resultados obtidos indicaram que o melhor desempenho preditivo foi alcançado com o uso exclusivo das métricas de Halstead, sugerindo que, para o problema de predição de tempo de execução e custo em ambientes FaaS, as características léxicas do código extraídas pelo Halstead possuem maior poder explicativo do que métricas estruturais extraídas pelo TIOBE.

Assim sendo, no *framework* Orama⁺⁺, o *Halsteader* é o componente responsável por extrair métricas Halstead a partir de um código-fonte fornecido. Consiste em uma aplicação desenvolvida em Node.js que utiliza a biblioteca EScomplex [121] para executar a geração das métricas. O componente também utiliza a biblioteca Express [106] a fim de disponibilizar uma API para o seu acionamento. As métricas de Halstead incluem:

- **Dificuldade (*difficulty*)**: mede a complexidade do código com base nos operadores e operandos únicos;
- **Volume**: representa a quantidade de informação necessária para implementar ou compreender um programa. Ela busca medir o “tamanho cognitivo” do código, considerando não apenas quantas instruções existem, mas também a diversidade dos elementos utilizados;
- **Esforço (*effort*)**: estima o esforço necessário para escrever ou entender o código;
- **Tempo (*time*)**: aproximação do tempo necessário para desenvolver o código;
- **Número de erros esperados (*bugs*)**: uma estimativa da quantidade de defeitos potenciais no código, obtida a partir da divisão do Volume por um fator fixo;
- **Número de operandos**: quantidade de operandos no código;
- **Número de operadores**: quantidade de operadores no código;
- **Vocabulário**: representa o número único de operadores (n_1) e operandos (n_2) distintos no código, ou seja, quanto maior o vocabulário, maior a diversidade de elementos utilizados no código;
- **Comprimento**: representa o total de operadores (n_1) e operandos (n_2) presentes no código, contando as repetições. Um comprimento maior indica que o código contém mais elementos e pode ser mais difícil de entender e manter.

A utilização do EScomplex foi motivada pela disponibilidade de mecanismos automatizados para extração dessas métricas em aplicações JavaScript, enquanto o uso do Node.js e do Express.js permitiu disponibilizar o processo de extração de métricas por meio de uma API leve, extensível e facilmente integrável aos demais componentes do Orama⁺⁺.

4.2.9 *Predictor*

O *Predictor* é o componente responsável por estimar o tempo de execução de uma função em um determinado provedor, a partir de métricas Halstead obtidas por meio do código-fonte da função, utilizando um modelo de IA pré-treinado. Ele consiste em uma aplicação Python que utiliza a biblioteca Flask [122] para fornecer uma API para seu acionamento.

O modelo de IA utilizado pelo *Predictor* foi gerado por um processo de treinamento, que será detalhado na Seção 4.7. O resultado desse treinamento foi incorporado ao componente *Predictor* no formato *Hierarchical Data Format version 5* (HDF5) [123], que é um formato de arquivo projetado para armazenar e organizar grandes volumes de dados de maneira eficiente. Ele é amplamente utilizado em Ciência de Dados, Aprendizado de Máquina e Computação Científica devido à sua capacidade de lidar com conjuntos de dados complexos e estruturados [123].

As bibliotecas Numpy [124], Pandas [124] e Sklearn [125] são utilizadas para realizar um pré-processamento das informações antes de submeter ao modelo de IA. O modelo de IA é carregado pela biblioteca Tensorflow [126] que, então, executa a predição. O *Predictor* retorna o tempo estimado de execução para a função em um determinado nível de concorrência (informado na requisição).

Note que o *Predictor* executa a predição para apenas um provedor por vez, ou seja, para obter as estimativas em diferentes provedores são executadas predições específicas para cada um deles. O *Frontend* executa essas predições em paralelo e as consolida em uma única visualização para comodidade do usuário, permitindo que ele efetue uma comparação de forma simplificada. Conforme ilustrado na Figura 4.1, os componentes do *framework* Orama⁺⁺ foram desenvolvidos para serem executados em contêineres *Docker* [127], gerenciados por *Docker Compose* [127], permitindo uma simples e rápida instanciação do *framework*.

A escolha do Python para implementação do *Predictor* foi motivada pela ampla adoção dessa linguagem em aplicações de Ciência de Dados e Aprendizado de Máquina, além da disponibilidade de bibliotecas consolidadas para manipulação de dados e execução de modelos preditivos. O Flask foi utilizado por oferecer uma estrutura leve e flexível para disponibilização da API, facilitando a integração do componente com os demais serviços do Orama⁺⁺. As bibliotecas Numpy, Pandas e Sklearn foram adotadas devido à eficiência no pré-processamento e transformação dos dados de entrada, enquanto o TensorFlow foi

escolhido por fornecer suporte robusto à execução de modelos de redes neurais treinados previamente. O uso do formato HDF5 permite armazenar os modelos de forma eficiente e portátil, favorecendo a reutilização e implantação do processo de predição em diferentes ambientes. Por fim, a containerização com Docker e Docker Compose contribui para a padronização, portabilidade e rápida implantação do *framework*.

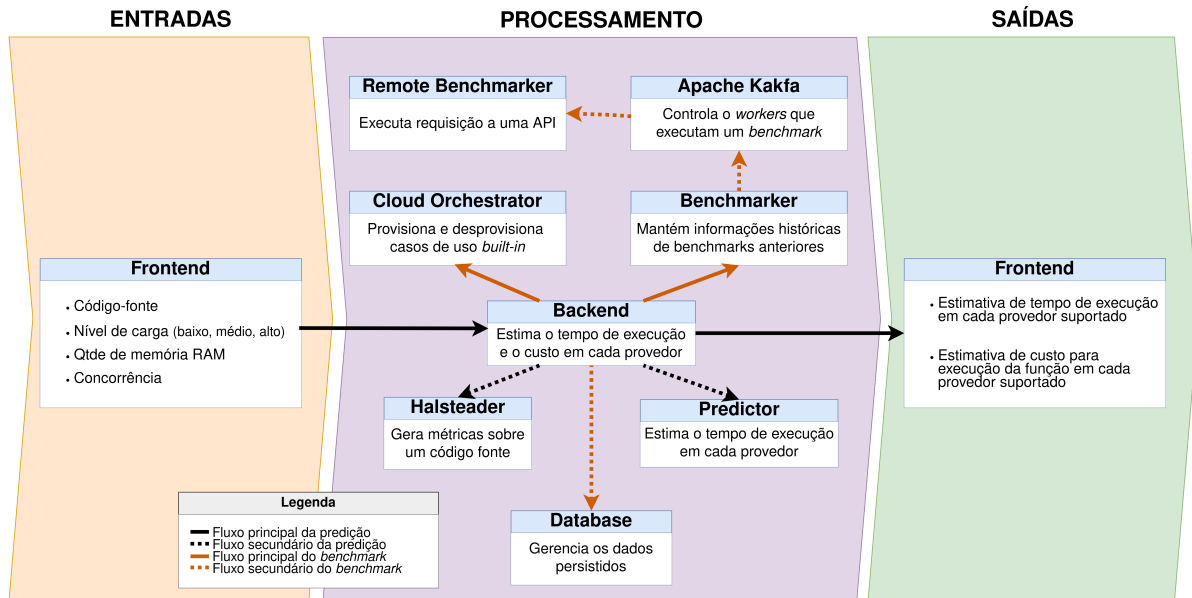


Figura 4.4: Arquitetura do *framework* Orama⁺⁺.

A Figura 4.4 apresenta uma visão consolidada do funcionamento do *framework* Orama⁺⁺ sob a perspectiva de fluxo de dados, organizada em três macroetapas: Entradas, Processamento e Saídas. Na figura, os fluxos principais da predição são representados por uma linha preta sólida e consistem nos passos que perpassam o *framework* com o propósito de obtenção das estimativas de tempo de execução e custos, partindo do código-fonte e dos parâmetros operacionais, tais como nível de carga, quantidade de memória e concorrência. Esse fluxo é complementado por fluxos secundários representados por linhas pretas tracejadas que acionam componentes intermediários como o *Halsteader* e o *Predictor*. Os fluxos para realização de *benchmarks* seguem pelas linhas alaranjadas, sendo as linhas sólidas o fluxo principal, que aciona os componentes principais, e a tracejada o respectivo fluxo secundário, que utiliza os componentes subjacentes. Essa visão complementa a descrição dos componentes ao evidenciar como as informações trafegam entre eles para viabilizar a predição de desempenho e o custo em ambientes FaaS.

Na Etapa de Entradas, o usuário fornece por meio do *Frontend* os principais parâmetros que caracterizam a execução da função, incluindo o código-fonte, o nível de carga (baixo, médio ou alto), a quantidade de memória RAM alocada e o grau de concorrência. Esses elementos representam tanto características intrínsecas do software quanto parâme-

tros operacionais do ambiente de execução, sendo fundamentais para capturar diferentes comportamentos de desempenho.

A Etapa de Processamento concentra os componentes responsáveis pela transformação dessas entradas em conhecimento preditivo e pela gestão da infraestrutura de testes. O *Backend* atua como o núcleo integrador, comunicando-se com o *Database* para gerenciar os dados persistidos. Para a análise do código, o *Halsteader* extrai métricas estruturais, enquanto o *Predictor* utiliza esses dados para estimar o tempo de execução. Adicionalmente, o *Backend* complementa as previsões de tempo de execução com as estimativas de custo, com base nas configurações de cada provedor.

No fluxo de *benchmark*, o *Cloud Orchestrator* é responsável por provisionar e desprovisionar casos de uso integrados. O *Benchmarker* mantém as informações históricas e interage com o *Apache Kafka*, que controla os *workers* encarregados de executar os testes. Estes, por sua vez, acionam o *Remote Benchmarker* para executar as requisições às APIs. Todo esse ecossistema permite que o *Backend* consolide a estimativa de tempo e o cálculo de custo para cada provedor.

Na Etapa de Saídas, o *framework* disponibiliza novamente no *Frontend* as estimativas de tempo de execução e custo associadas a cada provedor de nuvem. Isso permite uma análise comparativa que subsidia a tomada de decisão quanto à melhor configuração de implantação. Essa separação clara entre etapas reforça a modularidade da solução e evidencia o encadeamento lógico entre extração de características, execução de *benchmarks*, predição e análise de custo.

A partir dessa visão arquitetural orientada a fluxo, torna-se possível compreender não apenas os componentes isoladamente, mas também suas interações dinâmicas. Nesse contexto, a próxima seção detalha o fluxo de trabalho do *framework* Orama⁺⁺, descrevendo passo a passo como essas interações ocorrem durante a execução de um cenário completo, desde a submissão de uma função até a obtenção das estimativas finais.

4.3 Cálculo de Custo em Provedores de Nuvem

O custo de execução da função, em diferentes provedores de nuvem, é calculado pelo *framework* Orama⁺⁺ por meio do tempo de execução estimado pelo *Predictor*. Esse cálculo é fundamental para apoiar a tomada de decisão, permitindo não apenas avaliar o desempenho, mas também o impacto econômico de cada alternativa de implantação em ambientes FaaS.

De forma geral, o modelo de custo adotado considera três componentes principais: (i) o custo por invocação da função (*hit*), (ii) o custo proporcional ao tempo de execução e à quantidade de memória alocada (expresso em GB por segundo), e (iii) o custo associado

ao uso de CPU, quando aplicável. Assim, o custo total de execução pode ser compreendido como uma função do número de invocações, do tempo estimado de execução, da memória configurada e dos coeficientes de preço definidos por cada provedor.

Os modelos de cobrança variam entre os provedores, podendo incluir diferentes regiões geográficas, faixas de consumo (*tiers*) ou configurações específicas de recursos. Para lidar com essa heterogeneidade, o Orama⁺⁺ adota uma abordagem parametrizável, na qual os valores de custo são definidos por meio de configurações estruturadas, permitindo fácil atualização, conforme mudanças nos modelos de precificação dos provedores.

A Tabela 4.1 apresenta um exemplo de configuração de custos utilizada pelo *framework*, contemplando diferentes provedores e seus respectivos modelos de cobrança. No caso da AWS, por exemplo, os custos variam conforme a região (*sa-east-1*) e a quantidade de invocações mensais, sendo organizados em faixas progressivas. Já o GCP e o Alibaba Cloud utilizam modelos baseados em *tiers*, enquanto o Microsoft Azure apresenta variações associadas a diferentes planos de execução.

Tabela 4.1: Exemplo de configuração de custos por provedor no Orama⁺⁺.

Provedor	Região/Tier	Invocação (<i>hit</i>)	GB/s	vCPU/s
AWS	sa-east-1-first-6-billions	0.0000002	0.0000166667	0
AWS	sa-east-1-next-9-billions	0.0000002	0.000015	0
AWS	sa-east-1-over-15-billions	0.0000002	0.0000133334	0
GCP	tier-1	0.0000004	0.0000023283	0.00002400
GCP	tier-2	0.0000004	0.0000032592	0.00003360
Azure	all-regions	0.0000002	0.000016	0
Azure	all-regions-flex	0.0000004	0.000037	0
Alibaba	tier-1	0.00000015	0.000003	0.000002
Alibaba	tier-2	0.0000001275	0.00000255	0.0000017
Alibaba	tier-3	0.000000105	0.0000021	0.0000014

Com base nessas configurações, o *Backend* combina os valores de tempo estimado fornecidos pelo *Predictor* com os parâmetros de execução definidos pelo usuário, tais como número de invocações, nível de concorrência e quantidade de memória alocada, para calcular o custo total em cada provedor suportado. O cálculo do custo total de execução de uma função, no *framework* Orama⁺⁺, é formalizado na Equação 4.1:

$$C_{\text{total}} = N \cdot C_{\text{hit}} + N \cdot (T_{\text{exec}} \cdot M_{\text{GB}} \cdot C_{\text{GB/s}}) + N \cdot (T_{\text{exec}} \cdot C_{\text{vCPU/s}}) \quad (4.1)$$

na qual:

- N representa o número de invocações da função;
- C_{hit} é o custo por invocação;
- T_{exec} é o tempo de execução estimado da função;

- M_{GB} corresponde à quantidade de memória alocada (em GB);
- $C_{\text{GB/s}}$ é o custo por GB por segundo;
- $C_{\text{vCPU/s}}$ é o custo por uso de CPU por segundo.

Essa formulação reflete os três principais componentes do modelo de cobrança adotado pelos provedores de nuvem, os quais são o custo por invocação, o custo pelo tempo de retenção de memória e o custo pelo tempo de retenção de vCPU. A multiplicação desses termos pelo número de invocações permite capturar cenários de execução em larga escala, nos quais pequenas variações no tempo estimado ou nos coeficientes de preço podem resultar em diferenças significativas no custo total. Dessa forma, a equação integra as estimativas de desempenho fornecidas pelo *Predictor* com os parâmetros de configuração e os modelos de precificação dos provedores, permitindo uma análise comparativa precisa entre diferentes alternativas de implantação.

A Figura 4.5 ilustra um exemplo de cálculo de custo realizado pelo *framework*. Nessa figura, são apresentados os parâmetros de entrada, incluindo o código-fonte, número de invocações, quantidade de CPU e memória; bem como o tempo de execução estimado. A partir dessas informações, o sistema calcula automaticamente os custos correspondentes para diferentes provedores e configurações, permitindo uma comparação direta entre as alternativas.

Assim sendo, é possível observar que os custos variam significativamente entre os provedores e seus respectivos modelos de cobrança, mesmo quando submetidos às mesmas condições de execução. Isso evidencia a importância de considerar não apenas o desempenho, mas também o custo associado, reforçando a proposta do Orama⁺⁺ como uma ferramenta de apoio à tomada de decisão em ambientes multi-nuvem. Dessa forma, o cálculo de custo representa a etapa final do processamento no *framework*, transformando estimativas de desempenho em indicadores econômicos que permitem ao usuário selecionar a melhor configuração de implantação com base em critérios de custo-benefício.

4.4 Fluxo de Trabalho

O fluxo de trabalho do *framework* Orama⁺⁺ é estruturado em cinco fases principais: configuração, infraestrutura, *benchmark*, análise e predição, conforme mostrado na Figura 4.6. Esse processo garante a automação e a padronização da avaliação de desempenho em ambientes FaaS, permitindo desde a definição de provedores e casos de uso, até a execução de testes controlados e a aplicação de análises estatísticas e a predição de desempenho. Com isso, o Orama⁺⁺ assegura reprodutibilidade e precisão na comparação entre dife-

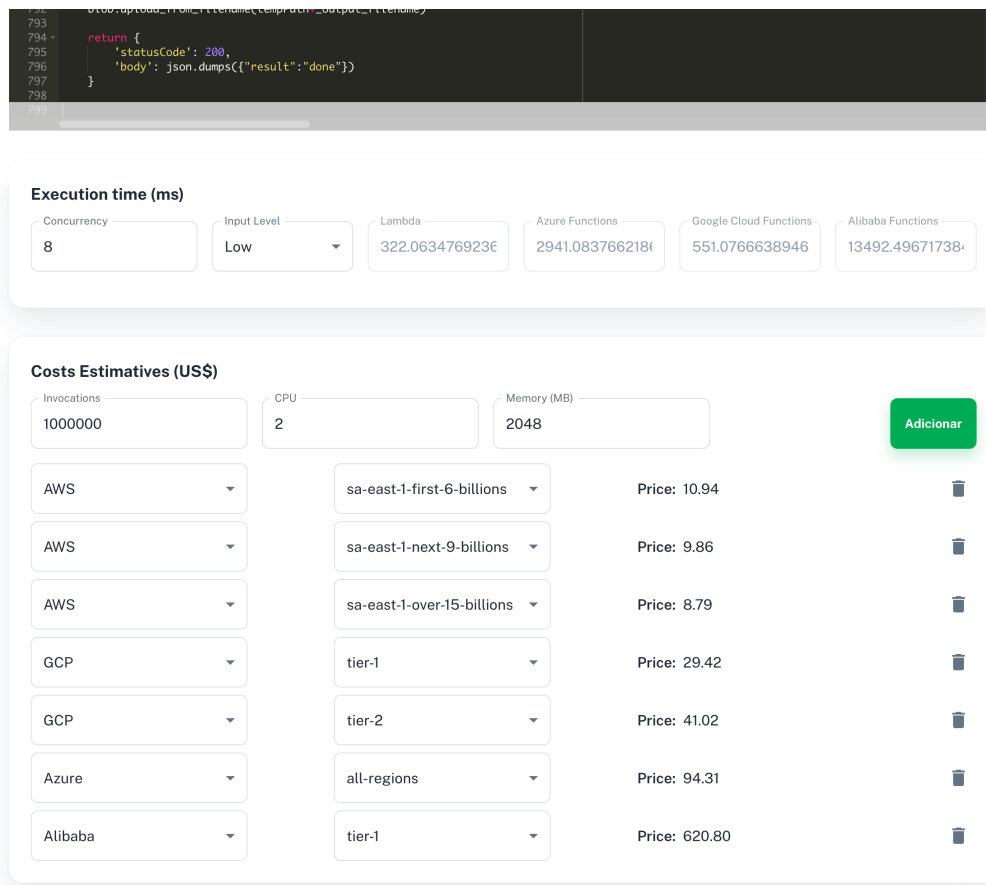


Figura 4.5: Exemplo de cálculo de custo a partir das estimativas de execução no Orama⁺⁺.

rentes provedores e configurações. O diagrama da Figura 4.6 contém as cinco raias que representam as principais fases do *framework*, conforme descritas a seguir:

- **Configuração:** consiste na fase de cadastramento dos objetos que participam da operação do *framework*, tais como provedor, caso de uso e *benchmark*. Ao instalar o *framework*, esses objetos podem ser criados automaticamente, permitindo que objetos estejam prontos para uso imediatamente;
- **Infraestrutura:** nesta fase ocorre o provisionamento do ambiente do caso de uso que será alvo de *benchmark*. Embora o *framework* possua casos de uso pré-configurados, que podem ser provisionados automaticamente, também é possível configurar casos de uso pré-implantados (que foram configurados sem o auxílio do *framework*) a partir dos *endpoints* a serem acionados na fase de *benchmark*;
- **Benchmark:** uma vez disponibilizado o ambiente, nessa fase o *framework* orquestra o *benchmark* acionando sistematicamente as APIs do ambiente do caso de uso, seguindo a configuração do *benchmark* que indica aspectos, tais como nível de concorrência, quantidade de repetições, *payload*, entre outras configurações;

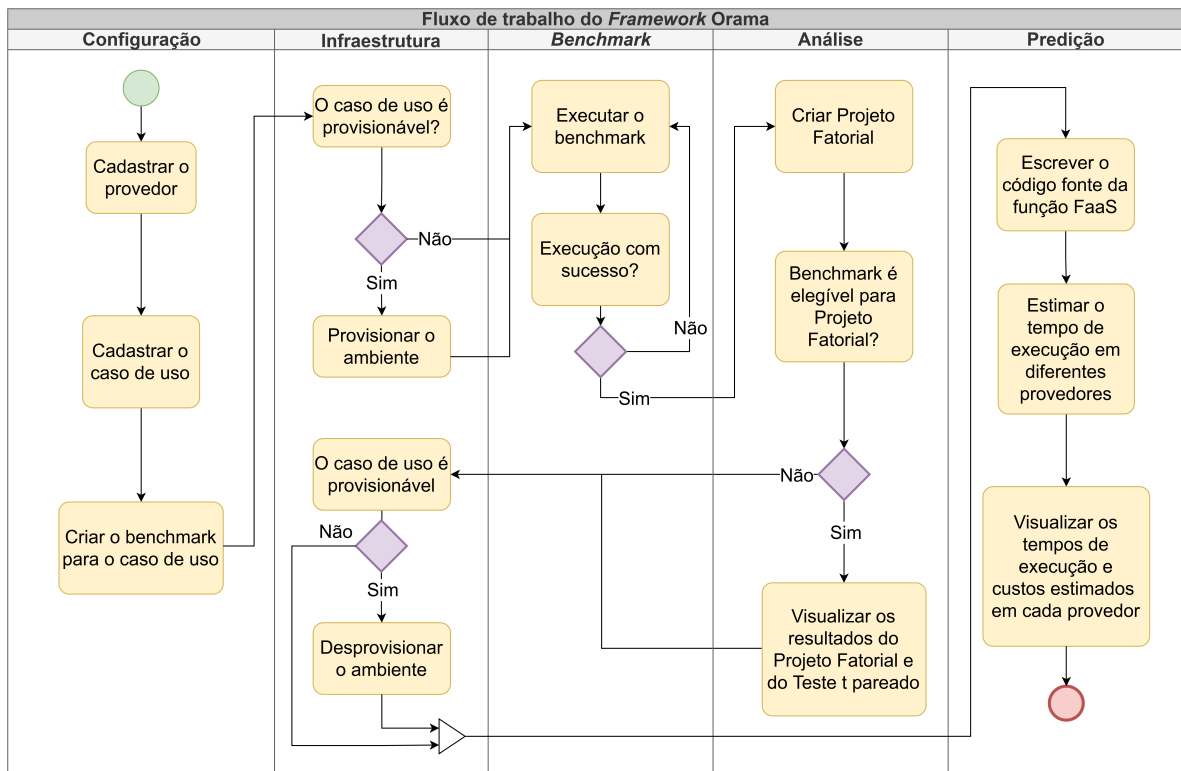


Figura 4.6: Fluxo de trabalho do *framework* Orama⁺⁺.

- **Análise:** ao finalizar com sucesso os *benchmarks*, os dados resultantes podem ser analisados usando ferramentas estatísticas. Nessa fase é criado um projeto *fatorial* 2^k para comparar dois *benchmarks*. Caso ambos *benchmarks* sejam elegíveis para execução de projeto *fatorial* 2^k , ou seja, as configurações sejam compatíveis e a quantidade de repetições não ultrapasse 30 (por conta da limitação da distribuição *t* – *student* usada nos cálculos), então os resultados do projeto *fatorial* 2^k e do teste *t* pareado serão calculados e apresentados ao usuário;
- **Predição:** nesta fase o usuário pode escrever uma função que deseja publicar em serviços de FaaS, e o *framework* apresentará estimativas de tempo de execução e de custo em cada provedor suportado pela plataforma. Essas informações podem auxiliar o usuário em um processo de decisão sobre em qual provedor deve ser implantada a sua função.

Ao instalar o *framework* é possível executar um comando para criar automaticamente casos de uso embutidos e alguns cenários de *benchmark*. Também é possível, no momento da instalação, configurar as credenciais dos provedores, permitindo o provisionamento de casos de uso relacionados a esses serviços. As instruções de instalação do *framework* estão disponíveis no repositório público do Orama⁺⁺². Além disso, também é possível instalar

²<https://github.com/unb-faas/orama>

o *framework* utilizando imagens *Docker* pré-construídas e publicadas no *Docker Hub*³.

Uma vez instalado e configurado corretamente, o próximo passo é provisionar o ambiente no qual os testes serão executados, e isso pode ser feito diretamente pela interface gráfica da plataforma. Os casos de uso embutidos possuem todos os artefatos de automação necessários para serem implantados nos respectivos provedores, e esses artefatos são aplicados nesta fase. O *framework* Orama⁺⁺ permite ainda a configuração de casos de uso personalizados que não precisam ser provisionados. Nesse caso, o usuário configura diretamente no *framework* as URLs que fazem parte do caso de uso e seus respectivos verbos: GET, POST, PUT ou DELETE.

Após serem provisionados (ou configurados), os ambientes estão aptos a serem testados. Nessa fase, o usuário pode executar os *benchmarks* quantas vezes forem necessárias. Um *benchmark* consiste na execução de rajadas de requisições sobre uma URL do ambiente do caso de uso. É possível configurar uma lista de rajadas dentro de um *benchmark*, assim como o número de repetições desse cenário, para obtenção de amostras para análise estatística.

É possível configurar, no *benchmark*, a execução de uma requisição adicional antes do início dos testes, como um aquecimento do serviço, para evitar o *cold start*. Também é possível configurar intervalos entre as execuções das rajadas, caso o objetivo seja analisar o *cold start* ou outra necessidade que exija um tempo de espera entre as requisições.

Quando os *benchmarks* são finalizados com sucesso, é possível realizar análises comparativas entre eles, desde que certas condições sejam respeitadas. O *framework* permite comparar *benchmarks* que envolvam diferentes provedores, dois níveis iguais de concorrência e o mesmo número de repetições. Uma vez que esses critérios são atendidos, será possível visualizar os resultados do projeto *fatorial 2^k* e do teste *t* pareado.

O projeto *fatorial 2^k* mostrará, entre outras métricas, a porcentagem do efeito do provedor, da concorrência e da combinação de ambos sobre os resultados obtidos nos *benchmarks*. O teste *t* pareado apresentará a significância da diferença encontrada entre as médias de latência dos resultados em diversos níveis de confiança, de 60% a 99,95%. Ao final da análise, o usuário pode desprovisionar o ambiente que foi implantado para a execução dos testes (se aplicável).

Outra ferramenta importante do *framework* Orama⁺⁺ é o preditor de tempo de execução. Essa ferramenta permite que o usuário escreva o código-fonte de uma função que pretende publicar em algum serviço de FaaS. A partir desse código são coletadas métricas Halstead, que então são enviadas para um preditor que estima o tempo de execução em cada provedor configurado na plataforma, tais como AWS Lambda, GCF, AZF e AFC.

³<https://hub.docker.com/u/oramaframework>

Essa ferramenta pode auxiliar o usuário na decisão sobre onde implantar a sua função, considerando tempo de execução mais vantajoso e, eventualmente, o custo previsto.

4.5 Casos de Uso Pré-configurados

O *framework* Orama⁺⁺ possui alguns casos de uso orientados a FaaS pré-configurados que podem ser implantados no AWS Lambda, no GCP, no AZF e no AFC. Uma vez implantados, os ambientes desses casos de uso podem passar por baterias de testes a partir de *benchmarks* também pré-configurados no *framework*.

Os casos de uso pré-configurados são capazes de implantar desde soluções simples, como uma calculadora, até APIs que aceitam verbos GET, POST e DELETE e consultam ou armazenam os dados em serviços adjacentes nos respectivos provedores de nuvem, tais como DBaaS e armazenamento por objetos. Nas próximas seções serão detalhados os casos de uso pré-configurados que integram o arcabouço do *framework* Orama⁺⁺.

4.5.1 Calculadora

O caso de uso de Calculadora representa o cenário mais elementar disponibilizado pelo *framework* Orama⁺⁺ e tem como principal objetivo servir como linha de base (*baseline*) para experimentos em ambientes FaaS. Por sua simplicidade estrutural e ausência de dependências externas, esse caso de uso permite isolar e avaliar aspectos fundamentais da execução de funções, tais como latência, tempo de inicialização (*cold start*), tempo de resposta, sobrecarga de invocação e variações de desempenho entre provedores de nuvem.

Nesse cenário, é implantada uma única função FaaS em cada provedor suportado, responsável por processar requisições HTTP contendo parâmetros via método GET. A função executa operações aritméticas básicas, tais como adição, subtração, multiplicação e divisão; sem interação com serviços externos, como bancos de dados ou sistemas de armazenamento. Essa característica torna o caso de uso particularmente adequado para experimentos controlados, nos quais se deseja minimizar interferências externas e focar exclusivamente no comportamento da infraestrutura FaaS.

Algoritmo 1: Calculadora Simples

Entrada: Evento HTTP

Saída: Resultado da operação ou mensagem(ns) de erro

```
1 Extrair  $a$ ,  $b$  e  $operacao$ ;
2 Inicializar  $erros \leftarrow []$ ;
3 Validação dos parâmetros;
4 Se  $a$  ausente então
5   | adicionar erro em  $erros$ ;
6 fim
7 Se  $b$  ausente então
8   | adicionar erro em  $erros$ ;
9 fim
10 Se  $operacao$  ausente então
11   | adicionar erro em  $erros$ ;
12 fim
13 senão se  $operacao$  inválida então
14   | adicionar erro em  $erros$ ;
15 fim
16 Se  $erros \neq []$  então
17   | Retornar resposta de erro contendo  $erros$ ;
18 fim
19 Converter  $a$  e  $b$  para valores numéricos;
20 Execução da operação;
21 Escolha  $operacao$  Caso  $adição$   $resultado \leftarrow a + b$ ;
22 Caso  $subtração$   $resultado \leftarrow a - b$ ;
23 Caso  $multiplicação$   $resultado \leftarrow a \times b$ ;
24 Caso  $divisão$  Se  $b \neq 0$  então
25   |  $resultado \leftarrow a/b$ ;
26 fim
27 senão
28   |  $resultado \leftarrow 0$ ;
29 fim
30 Retornar resposta contendo  $a$ ,  $b$ ,  $operacao$  e  $resultado$ ;
```

O Algoritmo 1 detalha o fluxo lógico implementado pela função. Inicialmente, os parâmetros de entrada (a , b e $operacao$) são extraídos do evento HTTP recebido. Em seguida, é realizada uma etapa explícita de validação, na qual são verificados tanto a presença quanto a validade dos parâmetros. Eventuais inconsistências são acumuladas em uma estrutura de erros, permitindo o retorno de mensagens informativas ao usuário, sem interromper prematuramente a execução.

Após a validação, os parâmetros numéricos são convertidos para tipos apropriados, garantindo a correta execução das operações. A lógica principal é então conduzida por uma estrutura de decisão que seleciona a operação aritmética a ser executada com base

no parâmetro informado. Destaca-se o tratamento da operação de divisão, que inclui uma verificação explícita para evitar divisão por zero, assegurando maior robustez da função mesmo em um cenário simplificado.

Por fim, a função retorna uma resposta estruturada contendo os valores de entrada e o resultado da operação, o que facilita a validação automatizada durante a execução dos *benchmarks*. Essa padronização de entrada e de saída é essencial para a integração com os mecanismos de geração de carga e coleta de métricas do Orama⁺⁺, permitindo a análise comparativa entre diferentes execuções, configurações e provedores de nuvem.

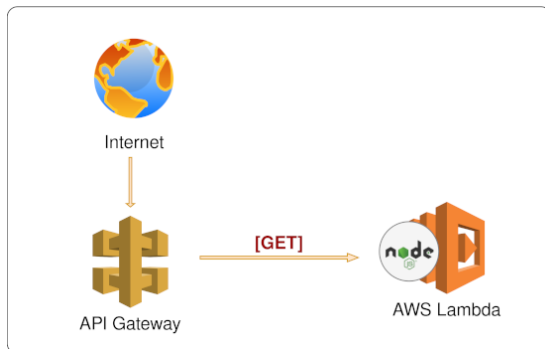


Figura 4.7: Calculadora no AWS Lambda.

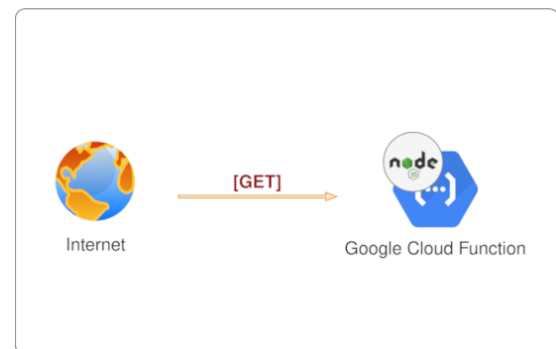


Figura 4.8: Calculadora no GCF.

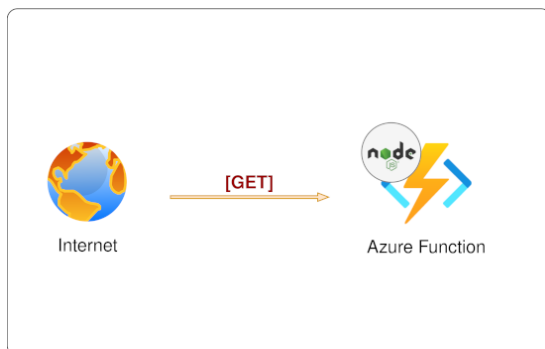


Figura 4.9: Calculadora no AZF.

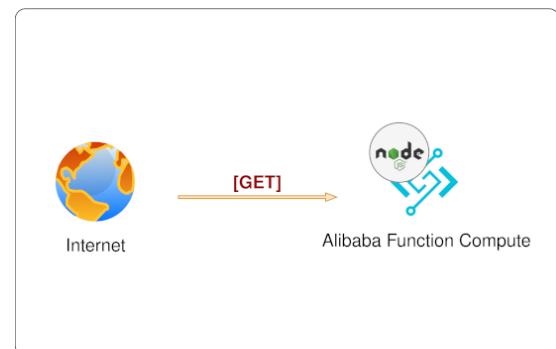


Figura 4.10: Calculadora no AFC.

Esse caso de uso, embora simples, desempenha um papel fundamental no contexto do Orama⁺⁺, pois estabelece um ponto de referência para a análise de desempenho em FaaS. A partir dele, é possível identificar o custo base de execução e a sobrecarga introduzida pela infraestrutura, servindo como base para comparação com casos de uso mais complexos que envolvem múltiplas funções e integrações com serviços externos. As Figuras 4.7, 4.8, 4.9 e 4.10 ilustram como esse caso de uso é provisionado nos provedores AWS Lambda, GCF, AZF e AFC, respectivamente.

4.5.2 API com Armazenamento em Objetos

O caso de uso de API com armazenamento em objetos representa um cenário mais próximo de aplicações reais baseadas em FaaS, incorporando não apenas o processamento de requisições HTTP, mas também a integração com serviços externos de armazenamento. Diferentemente do caso de uso da Calculadora, este cenário introduz dependências externas, permitindo avaliar o impacto de operações de I/O, latência de rede e interação com serviços gerenciados sobre o desempenho das funções.

Nesse contexto, o *framework* Orama⁺⁺ provisiona uma API composta por uma função FaaS responsável por tratar requisições HTTP utilizando os métodos GET, POST e DELETE. Essa função interage com serviços de armazenamento por objetos oferecidos pelos provedores de nuvem, como Amazon S3 [128], Google Cloud Storage [129], Azure Blob Storage [130] e Alibaba OSS [131]. Cada requisição resulta em operações de leitura, escrita ou remoção de objetos, permitindo simular um padrão típico de aplicações *serverless* orientadas a dados.

O fluxo lógico da API é dividido em três operações principais, detalhadas nos Algoritmos 2, 3 e 4. Em todos os casos, a função inicia com a extração e a validação dos parâmetros da requisição HTTP, garantindo a consistência dos dados antes da execução de qualquer operação no serviço de armazenamento.

O Algoritmo 2 descreve o tratamento de requisições POST, responsáveis pela criação de objetos. Após validar os parâmetros obrigatórios (como identificador e conteúdo), a função realiza a serialização dos dados e os persiste no serviço de armazenamento. Essa operação é, particularmente, relevante para análise de latência de escrita e *throughput*.

Algoritmo 2: API com Armazenamento em Objetos - Operação POST

Entrada: Evento HTTP

Saída: Confirmação de armazenamento ou erro

```
1 Extrair id e conteudo;
2 Inicializar erros ← [ ];
3 Se id ausente ou conteudo ausente então
4   | adicionar erro em erros;
5 senão
6   | Serializar conteudo;
7   | Armazenar objeto com chave id;
8 fim
9 Se erros ≠ [ ] então
10  | Retornar resposta de erro contendo erros;
11 senão
12  | Retornar confirmação de armazenamento;
13 fim
```

O Algoritmo 3 descreve o processamento de requisições GET, que são utilizadas para recuperar objetos previamente armazenados. A função valida o identificador fornecido e realiza a leitura no serviço de armazenamento. Esse fluxo permite avaliar métricas relacionadas à latência de leitura e consistência dos dados.

Algoritmo 3: API com Armazenamento em Objetos - Operação GET

Entrada: Evento HTTP

Saída: Objeto recuperado ou erro

```

1 Extrair id;
2 Inicializar erros  $\leftarrow$  [ ];
3 Se id ausente então
4   | adicionar erro em erros;
5 fim
6 Se erros  $\neq$  [ ] então
7   | Retornar resposta de erro contendo erros;
8 fim
9 Recuperar objeto com chave id;
10 Se objeto encontrado então
11   | Retornar conteúdo do objeto;
12 senão
13   | Retornar erro de objeto não encontrado;
14 fim

```

Por fim, o Algoritmo 4 apresenta o fluxo de remoção de objetos. Após validação do identificador, a função solicita a exclusão do objeto no serviço de armazenamento. Essa operação é relevante para análise de consistência eventual e latência de exclusão em diferentes provedores.

Algoritmo 4: API com Armazenamento em Objetos - Operação DELETE

Entrada: Evento HTTP

Saída: Confirmação de remoção ou erro

```

1 Extrair id;
2 Inicializar erros  $\leftarrow$  [ ];
3 Se id ausente então
4   | adicionar erro em erros;
5 fim
6 Se erros  $\neq$  [ ] então
7   | Retornar resposta de erro contendo erros;
8 fim
9 Remover objeto com chave id;
10 Retornar confirmação de remoção;

```

A utilização desse caso de uso no Orama⁺⁺ permite mensurar o impacto de fatores

como latência de rede, variabilidade de desempenho entre provedores, custos associados a operações de armazenamento e efeitos de escalabilidade sob diferentes cargas de trabalho.

Além disso, a padronização das operações REST e das respostas facilita a automação dos *benchmarks*, permitindo a geração de cargas sintéticas que simulam padrões de acesso típicos, como leitura intensiva, escrita intensiva ou cargas mistas. Dessa forma, esse caso de uso complementa o cenário da Calculadora, oferecendo uma visão mais abrangente do desempenho de aplicações *serverless*. As Figuras 4.11, 4.12, 4.13 e 4.14 ilustram a arquitetura de implantação desse caso de uso nos diferentes provedores de nuvem.

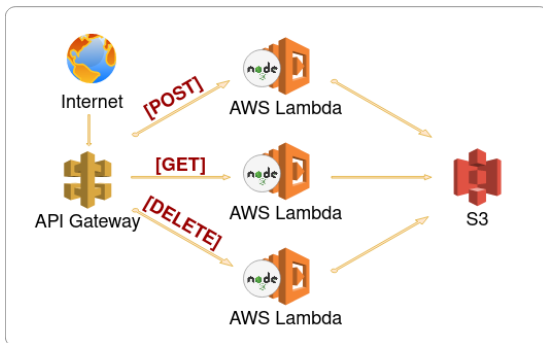


Figura 4.11: API → S3 no AWS Lambda.

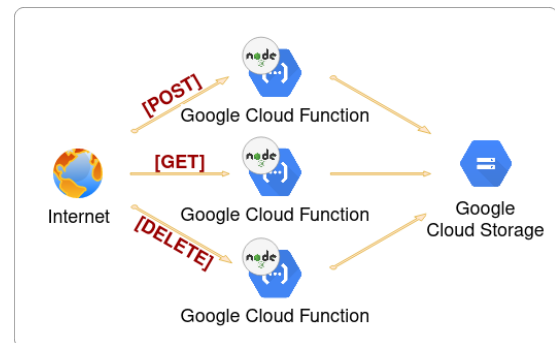


Figura 4.12: API → GCS no GCF.

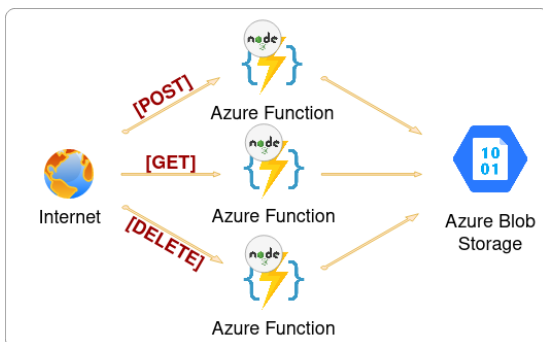


Figura 4.13: API → ABS no AZF.

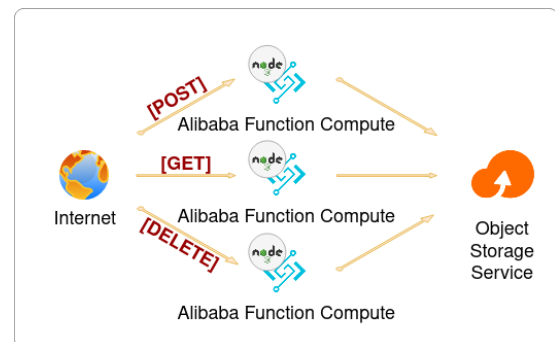


Figura 4.14: API → OSS no AFC.

4.5.3 API com Armazenamento em DBaaS

O caso de uso de API com Armazenamento em DBaaS representa um cenário mais avançado dentro do *framework* Orama⁺⁺, no qual múltiplas funções FaaS interagem com um serviço gerenciado de banco de dados. Esse cenário permite avaliar não apenas o desempenho individual das funções, mas também o comportamento de aplicações distribuídas que dependem de operações persistentes, consultas estruturadas e mecanismos de paginação.

O fluxo de inserção de dados é descrito pelo Algoritmo 5. Nesse processo, a função inicialmente valida a presença do corpo da requisição, garantindo que os dados necessários foram fornecidos. Em seguida, o conteúdo é convertido para um objeto manipulável e um

identificador único é gerado e associado ao registro. Por fim, o objeto é persistido no banco de dados. Essa operação é particularmente relevante para análise de latência de escrita, geração de identificadores e impacto de serialização de dados em ambientes FaaS.

O Algoritmo 6 descreve a operação de consulta, que introduz um elemento adicional importante, que é a segmentação de resultados por paginação. A função define parâmetros de início e fim da página, realiza a recuperação dos registros no banco de dados e converte os dados para o formato JSON, antes de retorná-los ao cliente. Esse fluxo permite avaliar o desempenho de consultas, o impacto da quantidade de dados retornados e o comportamento do sistema sob cargas de leitura intensiva.

Por sua vez, o Algoritmo 7 detalha o processo de remoção de registros. A função valida o identificador fornecido, garantindo sua consistência antes de realizar a operação. Após a conversão e verificação do valor, o registro correspondente é removido do banco de dados. Essa operação é relevante para análise de latência de exclusão, consistência e tratamento de requisições inválidas.

Algoritmo 5: API com DBaaS - Operação POST

Entrada: Evento HTTP

Saída: Confirmação de armazenamento ou mensagem de erro

- 1 **Se** *corpo da requisição ausente* **então**
- 2 **Retornar** erro de requisição inválida;
- 3 **fim**
- 4 Converter corpo da requisição em objeto;
- 5 Gerar identificador *id*;
- 6 Associar *id* ao objeto;
- 7 Armazenar objeto no banco de dados;
- 8 **Retornar** confirmação de sucesso;

Algoritmo 6: API com DBaaS - Operação GET

Entrada: Evento HTTP

Saída: Lista de registros

- 1 Definir parâmetros de segmentação (início e fim da página);
- 2 Recuperar registros do banco de dados;
- 3 Converter para JSON;
- 4 **Retornar** lista de registros;

No contexto do Orama⁺⁺, esse caso de uso é fundamental para avaliar cenários realistas de aplicações FaaS, nos quais operações de leitura e de escrita em banco de dados são frequentes e impactam diretamente o desempenho e o custo. A presença de múltiplas funções permite ainda analisar o comportamento individual de cada operação, bem como a variabilidade entre diferentes tipos de carga (inserção, consulta e remoção).

Algoritmo 7: API com DBaaS - Operação DELETE

Entrada: Evento HTTP

Saída: Confirmação de remoção ou mensagem de erro

- 1 **Se** *parâmetro id ausente* **então**
 - 2 **Retornar** erro de requisição inválida;
 - 3 **fim**
 - 4 Converter *id* para valor numérico;
 - 5 **Se** *id inválido* **então**
 - 6 **Retornar** erro de requisição inválida;
 - 7 **fim**
 - 8 Remover registro do banco de dados;
 - 9 **Retornar** confirmação de remoção;
-

As Figuras 4.15, 4.16 e 4.17 ilustram como esse cenário é provisionado nos diferentes provedores de nuvem, evidenciando a integração entre funções FaaS e serviços de banco de dados gerenciados. Na Figura 4.15, observa-se a integração entre o AWS Lambda e o serviço DynamoDB [132], caracterizado por um modelo *Not Only SQL* (NoSQL) altamente escalável e otimizado para baixa latência. De forma análoga, a Figura 4.16 apresenta a utilização do Firestore [33] no GCP, que oferece suporte a consultas estruturadas e sincronização eficiente de dados. Já na Figura 4.17 é ilustrada a integração com o CosmosDB [133] na Azure, um serviço multimodelo que fornece alta disponibilidade e baixa latência global.

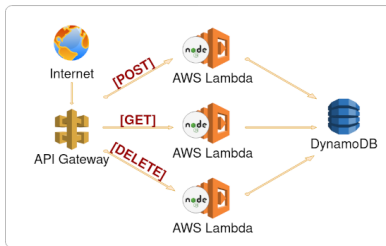


Figura 4.15: API → DynamoDB no AWS Lambda.

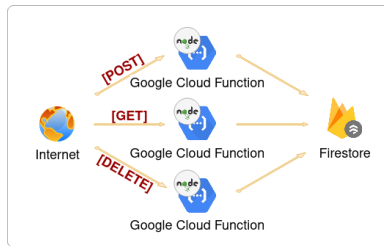


Figura 4.16: API → Firestore no GCF.

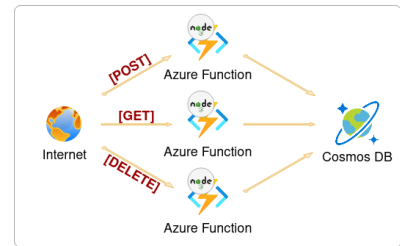


Figura 4.17: API → CosmosDB no AZF.

Esse caso de uso também permite investigar como diferentes estratégias internas dos provedores como particionamento de dados, indexação automática e replicação, influenciam métricas como latência, *throughput* e variabilidade de desempenho. Além disso, os artefatos de automação do Orama⁺⁺ são responsáveis por provisionar tanto as funções quanto os serviços de banco de dados, garantindo reprodutibilidade experimental e padronização dos ambientes. No provedor AFC não foi possível automatizar esse caso de uso devido a limitações da plataforma, o que também evidencia dificuldades práticas na adoção de arquiteturas *serverless* multi-nuvem.

Assim, embora esse caso de uso explore predominantemente operações orientadas a I/O e integração com serviços gerenciados, ele estabelece a base para a análise de cenários mais complexos. Nesse contexto, a seção a seguir introduz um caso de uso voltado ao alinhamento de sequências genéticas, que enfatiza cargas de trabalho intensivas em processamento (*CPU-bound*), permitindo uma avaliação complementar do comportamento dos provedores FaaS sob diferentes perfis de execução.

4.5.4 Alinhamento de Sequência Genética

Este caso de uso procura representar uma classe de aplicações com maior complexidade computacional e forte dependência de processamento intensivo, sendo particularmente relevante em domínios como Bioinformática, análise de dados científicos e aplicações de *High-Performance Computing* (HPC). Diferentemente dos casos anteriores, que enfatizam operações típicas de Sistemas Distribuídos (I/O, chamadas a serviços e persistência), o alinhamento de sequências genéticas introduz uma carga de trabalho predominantemente *CPU-bound*, permitindo avaliar o comportamento dos provedores FaaS sob cenários de maior consumo computacional.

O problema de alinhamento de sequências consiste em identificar similaridades entre duas cadeias de caracteres (tipicamente representando sequências de DNA, RNA ou proteínas), maximizando uma função de pontuação que considera correspondências, substituições e penalidades por lacunas. Esse tipo de processamento é amplamente utilizado em ferramentas como *Basic Local Alignment Search Tool* (BLAST)⁴ e em *pipelines* de análise genômica.

No contexto deste trabalho, esse caso de uso é implementado como uma função FaaS que recebe, via requisição, duas sequências genéticas e retorna o alinhamento ótimo entre elas, juntamente com sua pontuação. O Algoritmo 8 apresenta a lógica fundamental dessa função, na qual são inicialmente registradas as informações do estado e do tempo antes da execução do alinhamento. Em seguida, ocorre o alinhamento em si e o cálculo da pontuação. Por último, são coletados os dados do estado final após o alinhamento, incluindo o tempo. Essa abordagem permite avaliar métricas como tempo de execução, escalabilidade e custo em cenários nos quais o tempo de vCPU e o gerenciamento de memória são fatores críticos.

⁴BLAST: <https://blast.ncbi.nlm.nih.gov/Blast.cgi>

Algoritmo 8: Alinhamento de sequências genéticas

Entrada: Evento HTTP contendo sequências e parâmetros

Saída: Confirmação de execução

- 1 Extrair dados da requisição;
 - 2 Converter sequências para formato adequado;
 - 3 Registrar instante de início;
 - 4 Inicializar estrutura de resultado parcial;
 - 5 Armazenar estado inicial;
 - 6 Executar algoritmo de alinhamento;
 - 7 Calcular pontuação do alinhamento;
 - 8 Registrar instante de término;
 - 9 Calcular duração da execução;
 - 10 Atualizar estrutura de resultado com alinhamento e métricas;
 - 11 Armazenar resultado final;
 - 12 **Retornar** confirmação de execução;
-

Para a implementação do alinhamento foi adotado o algoritmo de Hirschberg [15], que é uma variação do algoritmo clássico de Needleman-Wunsch [53]. A principal justificativa para essa escolha reside na sua eficiência espacial. Enquanto o algoritmo de Needleman-Wunsch apresenta complexidade de tempo $O(mn)$ e complexidade de espaço também $O(mn)$, o algoritmo de Hirschberg mantém a mesma complexidade temporal, porém reduz significativamente o consumo de memória para $O(\min(m, n))$.

Essa característica é particularmente relevante no contexto de ambientes FaaS, nos quais os recursos são limitados e diretamente associados ao custo de execução. Em provedores como AWS Lambda, GCF e AZF, a alocação de memória impacta não apenas a capacidade de processamento, mas também o modelo de precificação. Dessa forma, o uso do algoritmo de Hirschberg permite executar alinhamentos de sequências maiores sem exceder os limites de memória impostos pelas plataformas, além de reduzir o custo associado à execução. Adicionalmente, o algoritmo de Hirschberg utiliza uma estratégia de divisão e conquista, decompondo o problema original em subproblemas menores, o que o torna mais adequado para cenários nos quais há restrições de memória, ainda que ao custo de uma maior complexidade de implementação quando comparado a abordagens mais diretas.

Portanto, a adoção do algoritmo de Hirschberg neste caso de uso permite ao *framework* Orama⁺⁺ explorar um cenário representativo de aplicações científicas e computacionalmente intensivas, fornecendo subsídios para análises mais abrangentes sobre o desempenho e custo de execuções FaaS em diferentes provedores de nuvem. A Figura 4.18 ilustra a arquitetura do caso de uso de alinhamento de sequência genética no contexto do *framework* Orama⁺⁺. Nessa configuração, a função FaaS recebe duas sequências genéticas como entrada, processa o alinhamento utilizando o algoritmo de Hirschberg e retorna o

resultado ao solicitante. Observa-se que, diferentemente dos casos de uso anteriores, não há integração com serviços externos de persistência ou armazenamento, caracterizando um fluxo predominantemente computacional. Essa arquitetura evidencia a execução isolada da função e destaca o impacto direto dos recursos de vCPU e memória no tempo de resposta, sendo adequada para análise de desempenho em cenários *CPU-bound*.

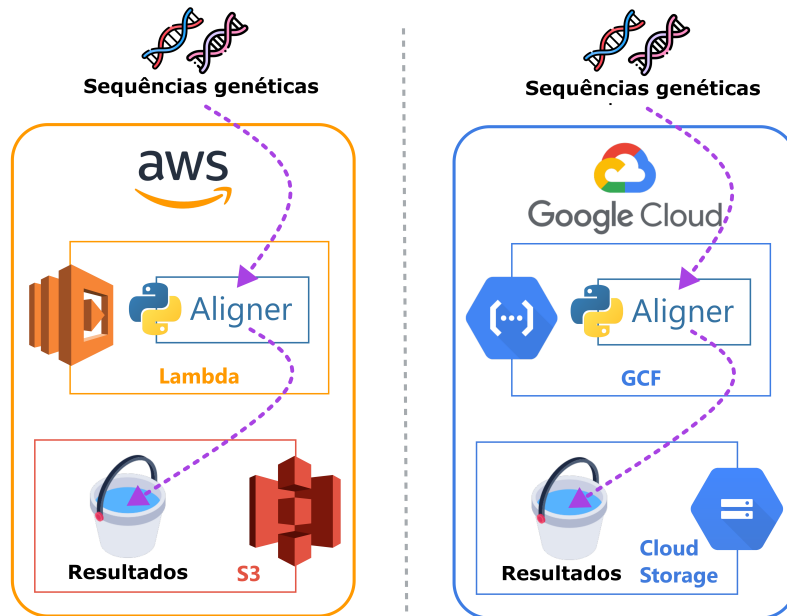


Figura 4.18: Alinhamento de sequência genética.

Além dos casos de uso pré-configurados, o *framework* permite a configuração de casos de uso implantados fora da plataforma. Neste caso, o usuário é responsável por implantar toda a infraestrutura necessária, inclusive, podendo utilizar outros modelos de nuvem diferentes de FaaS. Desde que o serviço seja acionável via API, será possível configurá-lo como um caso de uso no Orama⁺⁺, executar *benchmarks* sobre ele e aplicar as análises estatísticas que o *framework* disponibiliza.

4.5.5 Impacto no Esforço Experimental

A definição de casos de uso pré-configurados no *framework* Orama⁺⁺ não apenas padroniza os cenários experimentais, mas também reduz significativamente o esforço necessário para sua execução quando comparado ao processo manual descrito anteriormente.

Conforme discutido na Seção 2.2, a condução manual de experimentos em ambientes FaaS envolve uma série de atividades operacionais repetitivas, incluindo provisionamento de recursos, configuração de integrações, adaptação de código e execução de testes. Essas atividades, quando realizadas sem automação, resultam em um custo temporal elevado, especialmente, em cenários multi-nuvem.

Tabela 4.2: Estimativa de esforço manual por caso de uso e por provedor.

Caso de Uso	Tempo Estimado
Calculadora	8 – 12 horas
API + Armazenamento em Objetos	12 – 20 horas
API + DBaaS	15 – 28 horas
Alinhamento de Sequência Genética	20 – 40 horas

Considerando os casos de uso apresentados na Seção 4.5, é possível estimar o esforço necessário para sua implementação manual em diferentes provedores. A Tabela 4.2 sumaria esse custo com base na complexidade de cada cenário. Em um cenário com múltiplos provedores (por exemplo, AWS, GCP, Azure e Alibaba), o esforço total pode ultrapassar facilmente 80 a 160 horas, dependendo da complexidade do caso de uso. Além disso, esse processo tende a ser iterativo, exigindo reconfigurações e novas execuções experimentais, o que aumenta ainda mais o custo total. O *framework* Orama⁺⁺ automatiza as principais etapas do processo experimental, incluindo:

- Provisionamento de infraestrutura;
- Implantação das funções FaaS;
- Configuração de integrações com serviços gerenciados;
- Execução de *benchmarks*;
- Coleta de métricas.

Com isso, o esforço humano passa a se concentrar majoritariamente em atividades de maior valor analítico, como definição de cenários e interpretação dos resultados. A Tabela 4.3 apresenta uma estimativa do tempo necessário para execução dos mesmos casos de uso utilizando o *framework*. Observa-se que, com o uso do *framework*, o tempo necessário deixa de crescer linearmente com o número de provedores, uma vez que os artefatos de automação encapsulam as particularidades de cada plataforma.

Tabela 4.3: Estimativa de esforço com o uso do Orama⁺⁺ por caso de uso (multi-nuvem).

Caso de Uso	Tempo Estimado
Calculadora	1 – 2 horas
API + Armazenamento em Objetos	2 – 4 horas
API + DBaaS	3 – 6 horas
Alinhamento de Sequência Genética	4 – 8 horas

A comparação entre os dois cenários evidencia uma redução substancial no esforço necessário para condução dos experimentos. De forma geral, o uso do Orama⁺⁺ proporciona:

- Redução de aproximadamente 70% a 90% no tempo total de execução;
- Eliminação de tarefas operacionais repetitivas;
- Maior padronização e reprodutibilidade dos experimentos;
- Redução da variabilidade introduzida por configurações manuais.

Além da redução direta de tempo, um aspecto relevante é a diminuição do esforço cognitivo associado à configuração de múltiplos provedores, cada um com suas próprias interfaces, modelos de autenticação e APIs. Assim, os resultados apresentados indicam que o custo operacional da experimentação manual constitui um fator limitante para estudos comparativos em ambientes FaaS. Nesse contexto, o *framework* Orama⁺⁺ viabiliza a execução de experimentos mais amplos e sistemáticos, reduzindo significativamente o tempo necessário para obtenção de resultados. Dessa forma, o Orama⁺⁺ não apenas automatiza o processo experimental, mas também amplia o escopo de análises possíveis, permitindo a avaliação de múltiplos cenários e provedores com um custo operacional significativamente menor.

4.5.6 *Overhead* Experimental do *Framework*

Orama⁺⁺ introduz mecanismos adicionais de orquestração, coleta de métricas e coordenação experimental. Apesar disso, o impacto dessas operações sobre as medições realizadas mostrou-se reduzido quando comparado ao tempo total de execução dos casos de uso avaliados. Em particular, as atividades desempenhadas pelo *framework*, tais como disparo de requisições, coleta assíncrona de métricas e armazenamento de resultados, são executadas de forma desacoplada da lógica principal das funções FaaS, minimizando interferências diretas no caminho crítico de execução.

Além disso, a própria variabilidade inerente aos ambientes *serverless*, incluindo efeitos de rede, escalonamento interno dos provedores e eventos de *cold start*, tende a representar uma parcela significativamente maior da latência observada do que o custo adicional introduzido pelo Orama⁺⁺. Dessa forma, embora exista um *overhead* residual associado à instrumentação experimental, os resultados obtidos indicam que esse impacto não é suficientemente significativo para comprometer a validade estatística ou alterar as conclusões comparativas derivadas dos experimentos.

4.6 Análise Estatística

Os *benchmarks* realizados por meio do *framework* Orama⁺⁺ geram uma quantidade expressiva de dados experimentais, frequentemente sujeitos a variabilidade inerente a ambi-

entes distribuídos, como interferência de rede, variação de carga nos provedores e efeitos de inicialização (*cold start*). Embora a ferramenta ofereça mecanismos de visualização, como gráficos descritivos, a comparação direta entre *benchmarks* sem o suporte de métodos estatísticos pode conduzir a interpretações imprecisas ou enviesadas.

4.6.1 Projeto *Fatorial* 2^k

No contexto do *framework* Orama⁺⁺, o projeto *fatorial* 2^k considera dois fatores principais: (i) o provedor de nuvem e (ii) o nível de concorrência, caracterizando um experimento 2^2 . Essa modelagem permite quantificar: (a) o efeito isolado do provedor, (b) o efeito isolado da concorrência, e (c) o efeito da interação entre esses fatores. A análise da interação é particularmente relevante, pois o impacto da concorrência pode variar significativamente dependendo do provedor utilizado.

Por exemplo, considere dois *benchmarks*, sendo um executado no AWS Lambda e outro no GCF, ambos com níveis de concorrência 2, 4, 8 e 16. Nesse cenário, define-se o fator *provedor* com níveis AWS Lambda (inferior) e GCF (superior), e o fator *concorrência* com níveis 2 (inferior) e 16 (superior). A partir dessa configuração, o projeto *fatorial* 2^k permite estimar os efeitos principais e de interação, além de calcular o erro experimental, que representa a variabilidade não explicada pelos fatores considerados.

No Orama⁺⁺, a aplicação do projeto *fatorial* 2^k fornece ao usuário não apenas os resultados finais, mas também a memória de cálculo completa e visualizações que facilitam a interpretação dos efeitos. Para garantir a validade estatística da análise, é necessário que os *benchmarks* comparados sejam compatíveis, ou seja, utilizem os mesmos níveis de concorrência e número de repetições. Adicionalmente, o número de repetições é limitado a 30, pois os cálculos assumem a distribuição *t*-Student [134], adequada para amostras pequenas. Uma vez atendidos esses requisitos, o Orama⁺⁺ complementa automaticamente a análise com o teste *t* pareado.

4.6.2 Teste *t* Pareado

A escolha do teste *t* pareado é particularmente adequada ao contexto do Orama⁺⁺, pois os *benchmarks* comparados são executados sob condições equivalentes (mesmos níveis de concorrência e número de repetições), permitindo a comparação direta entre pares de observações. Isso reduz o impacto de variáveis externas não controladas e aumenta o poder estatístico da análise.

O Orama⁺⁺ realiza o teste em múltiplos níveis de confiança, variando de 60% até 99,95%, permitindo ao usuário avaliar o grau de evidência necessário para considerar uma diferença como significativa. Caso o resultado do teste não seja significativo para um

determinado nível de confiança, conclui-se que não há evidência estatística suficiente para afirmar que os *benchmarks* diferem entre si, sendo, portanto, considerados equivalentes do ponto de vista estatístico.

Essa abordagem é fundamental para evitar conclusões equivocadas, como por exemplo assumir superioridade de um provedor com base em diferenças marginais ou decorrentes de ruído experimental. Assim, o teste *t* pareado atua como um mecanismo de validação das conclusões extraídas a partir dos dados.

Em conjunto, o projeto *fatorial 2^k* e o teste *t* pareado permitem que o *framework* Orama⁺⁺ não apenas descreva os resultados dos *benchmarks*, mas também os qualifique estatisticamente, promovendo análises mais confiáveis e reproduzíveis. Entretanto, para a extração de *insights* mais avançados e capacidade preditiva, torna-se necessária a aplicação de modelos de IA sobre os dados gerados. Na Seção 4.7 são apresentados os detalhes do modelo de IA utilizado no Orama⁺⁺.

4.7 Modelo de Redes Neurais

O componente *Predictor* do *framework* Orama⁺⁺, detalhado na Seção 4.2.9, utiliza um modelo de Redes Neurais que foi treinado usando os dados históricos de *benchmarks* realizados no decorrer da evolução do *framework*. Nesta seção serão detalhados os processos de elaboração do *dataset* e o treinamento que resultou no modelo de Redes Neurais utilizado no *Predictor*.

4.7.1 Construção do *Dataset*

O treinamento de Redes Neurais que seja capaz de estimar o tempo de execução de uma determinada função em um provedor específico exige que diversas execuções de *benchmarks* tenham sido realizadas até que existam dados reais de tempo de execução suficientes para serem utilizados no processo de treinamento, e resultem em uma convergência satisfatória do modelo. Para isso, diversos experimentos, com configurações diferentes do *framework* Orama⁺⁺, foram realizados e seus resultados armazenados no repositório Orama⁺⁺ Results⁵. Esses experimentos serão detalhados no Capítulo 6.

Uma vez executados os experimentos e os resultados consolidados, obteve-se um conjunto com 332.190 registros de execuções. Entretanto, esses registros tinham apenas a informação do caso de uso utilizado na execução, mas nenhum detalhe sobre o código-fonte relacionado ao respectivo caso de uso. Dessa forma, foi preciso processar todos esses dados, relacionando cada execução com o respectivo código-fonte do caso de uso. Para

⁵<https://github.com/unb-faas/orama-results>

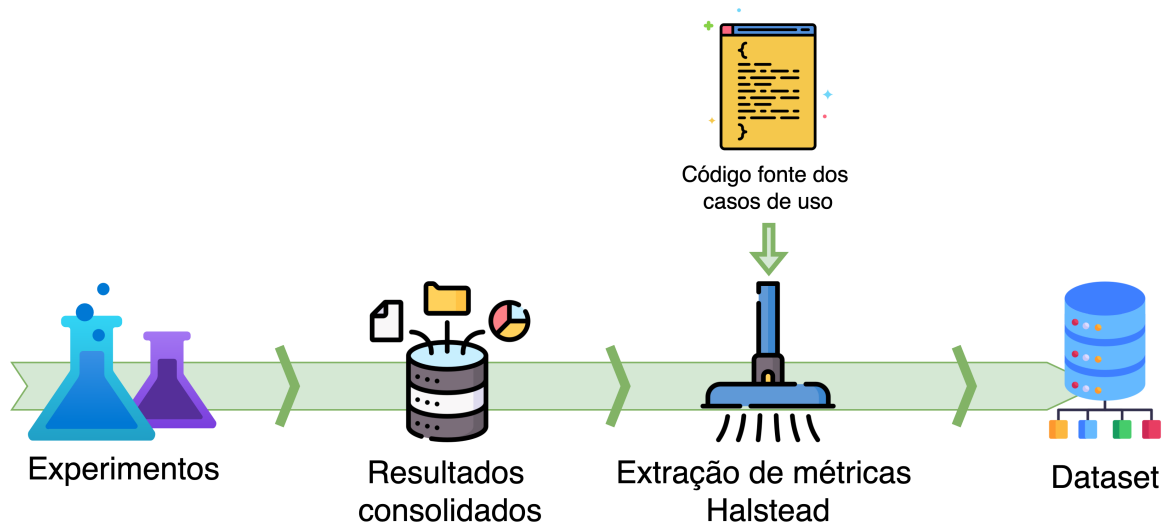


Figura 4.19: Passos para a construção do *dataset*.

isso, foram extraídas as métricas Halstead de cada caso de uso utilizado nos experimentos, e essas métricas foram incluídas nos registros de cada execução. Assim, a Figura 4.19 apresenta os passos seguidos na construção do *dataset* usado nesta tese para treinar o modelo de Redes Neurais implementado no Orama⁺⁺. Esse processamento resultou em um *dataset* composto pelas seguintes informações para cada execução:

- **timeStamp:** data e hora;
- **label:** descrição;
- **success:** indica resultado com sucesso ou falha;
- **concurrency:** nível de concorrência;
- **provider:** provedor;
- **usecase:** nome do caso de uso;
- **total_operands:** métrica Halstead;
- **distinct_operands:** métrica Halstead;
- **total_operators:** métrica Halstead;
- **distinct_operators:** métrica Halstead;
- **time:** métrica Halstead;
- **bugs:** métrica Halstead;
- **effort:** métrica Halstead;

- **volume**: métrica Halstead;
- **difficulty**: métrica Halstead;
- **vocabulary**: métrica Halstead;
- **length**: métrica Halstead;
- **latency**: tempo de execução apurado.

Vale ressaltar que os procedimentos mencionados nesta seção foram realizados por meio de *scripts* escritos em Python, que podem ser verificados no repositório do *framework*⁶.

4.7.2 Elaboração do Modelo

Com o *dataset* construído, a fase de elaboração do modelo de Redes Neurais se iniciou pelo pré-processamento dos dados. Na sequência, foi realizada uma modelagem inicial e depois uma otimização no modelo, para finalmente realizar o treinamento. Por fim, foram implementadas ferramentas para avaliação do modelo. A Figura 4.20 apresenta uma síntese das fases do treinamento que serão detalhadas nas próximas seções.

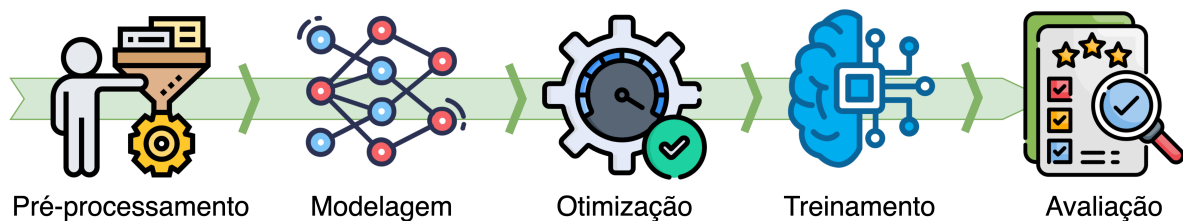


Figura 4.20: Passos da elaboração do modelo de Redes Neurais.

4.7.2.1 Pré-processamento

Os dados no *dataset* continham informações valiosas para o treinamento. Todavia, eles precisaram ser ajustados para um melhor entendimento por parte do modelo. Para isso, foram realizadas diversas transformações nos dados, tais como:

- Remoção de colunas desnecessárias, tais como *timeStamp*, *label* e *usecase*;
- Categorização da coluna *provider*;
- Remoção de linhas duplicadas;
- Tratamento de valores ausentes, por meio da identificação e remoção de execuções com essa característica;

⁶Disponível em: <https://github.com/unb-faas/orama>

- Normalização utilizando $z - score$, exceto para a coluna *latency*, que é o alvo da predição;
- Suavização dos dados removendo *spikes* utilizando *Weighted Moving Average* (WMA);
- Winsorização com limites inferiores e superiores em 15%;
- Identificação e tratamento de *outlier* usando o método IQR;
- Redução de escala usando PCA.

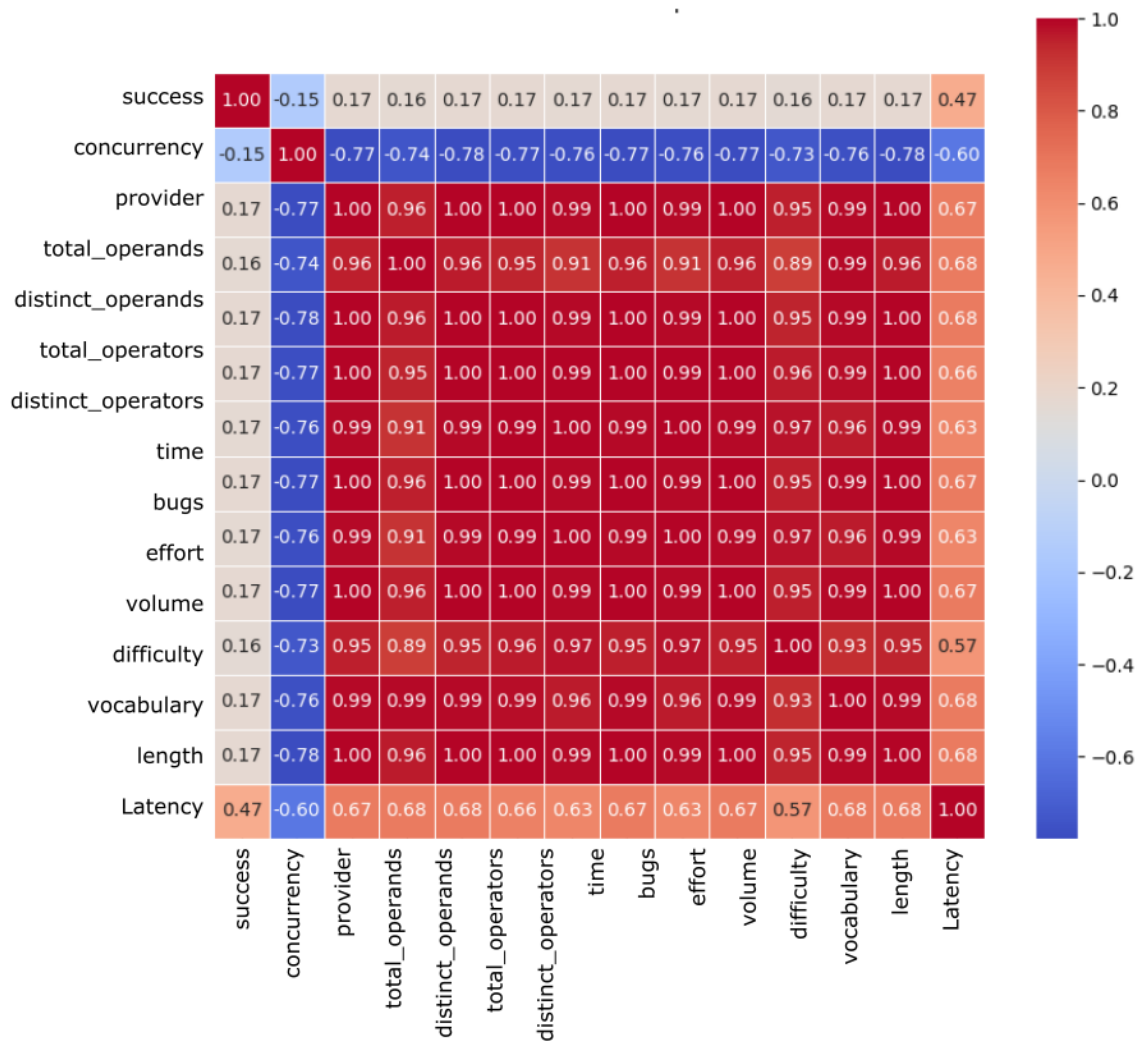


Figura 4.21: Correlação entre os parâmetros do *dataset* inicial.

Uma análise de correlação entre os parâmetros do *dataset* foi conduzida a partir da matriz de correlação que, então, deu origem ao mapa de calor mostrado na Figura 4.21. Nesse mapa é possível observar a existência de alta correlação entre as métricas Halstead. Dessa forma, os campos *total_operands*, *distinct_operands*, *total_operators*, *distinct_operators*,

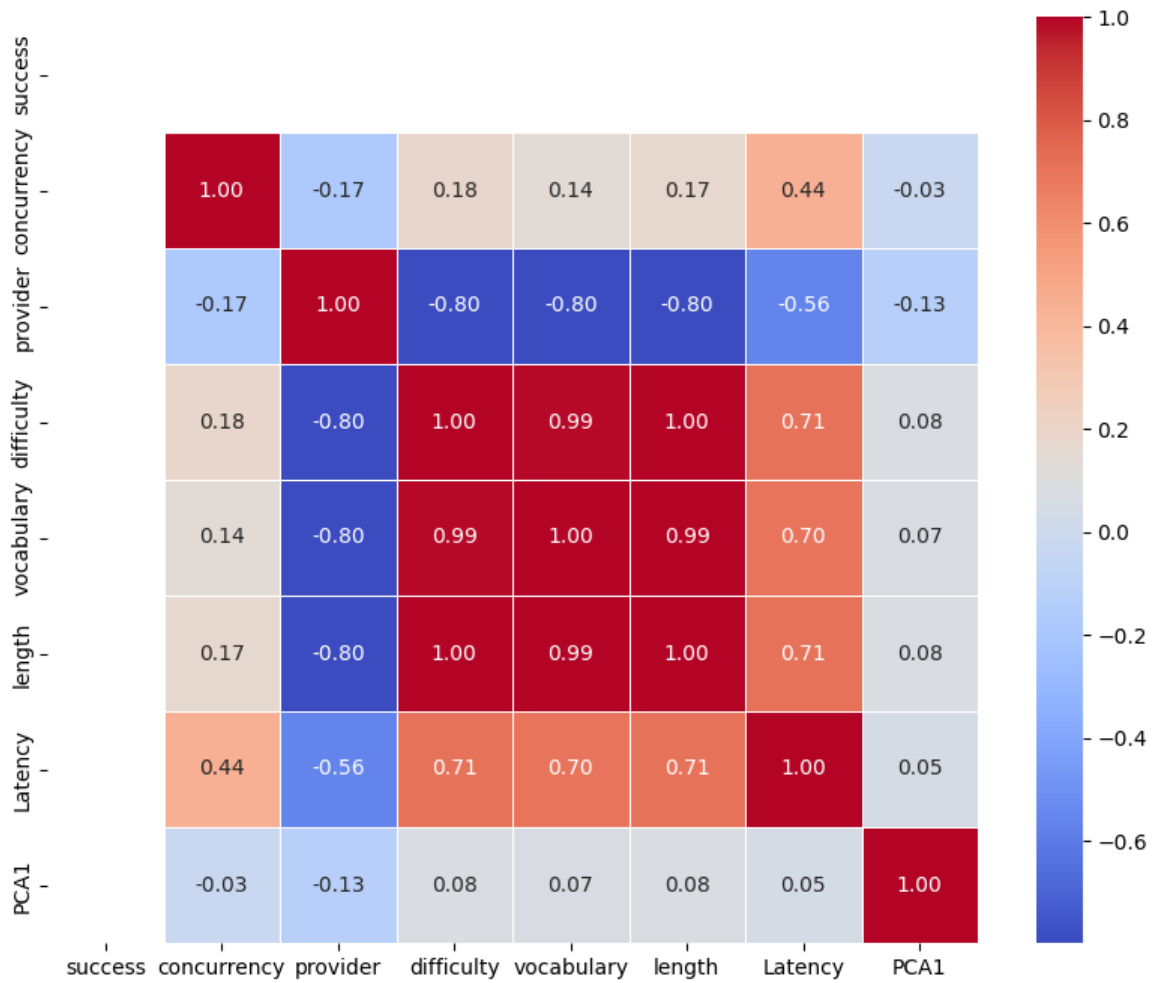


Figura 4.22: Correlação entre os parâmetros do *dataset* com PCA.

time, *bugs*, *effort* e *volume* foram substituídos por um único parâmetro PCA que os representa, reduzindo consideravelmente a dimensionalidade dos dados, conforme mostrado na Figura 4.22. Assim, após os procedimentos de pré-processamento dos dados, o modelo está pronto para ser treinado.

4.7.2.2 Modelagem

Considerando a quantidade de parâmetros do *dataset*, optou-se por utilizar um modelo de IA capaz de capturar relações complexas e não lineares entre as variáveis de entrada e o tempo de execução das funções. Para isso, foi adotada uma abordagem baseada em Redes Neurais artificiais, com suporte à avaliação de múltiplas arquiteturas, incluindo modelos do tipo *Dense*, *LSTM* e *Bidirectional LSTM*.

O processo de modelagem foi implementado de forma modular, contemplando as etapas de pré-processamento, divisão dos dados, otimização de hiperparâmetros, construção

do modelo, treinamento e avaliação. Inicialmente, o *dataset* passou por um *pipeline* de pré-processamento que incluiu remoção de atributos irrelevantes, tratamento de valores ausentes, eliminação de duplicatas e suavização de ruídos por meio de técnicas como WMA. Além disso, variáveis categóricas foram codificadas e os dados normalizados utilizando *z-score*, garantindo melhor estabilidade no treinamento.

Após o pré-processamento, os dados foram divididos em conjuntos de treinamento e teste. Em seguida, realizou-se o processo de *fine-tuning* dos hiperparâmetros. Esse processo consistiu em uma busca sistemática sobre o espaço de hiperparâmetros, incluindo taxa de aprendizado, número de neurônios nas camadas intermediárias, função de perda e uso de *dropout*. A busca foi conduzida por meio de múltiplas tentativas (*attempts*) e épocas reduzidas, permitindo explorar diferentes combinações de forma eficiente. As funções de perda avaliadas incluíram *Mean Squared Error* (MSE), *Mean Absolute Error* (MAE) e *huber*, enquanto a taxa de aprendizado foi ajustada em escala logarítmica.

Ao final do processo de otimização, os melhores hiperparâmetros encontrados foram utilizados na etapa de treinamento final. Neste trabalho, a arquitetura que apresentou melhor desempenho foi a *Bidirectional LSTM*, cujos parâmetros estão apresentados na Tabela 4.4.

Tabela 4.4: Parâmetros do modelo.

Parâmetro	Valor
Taxa de aprendizado	0.012410865746975944
Função de perda	mean_squared_error
Camadas intermediárias	1
Neurônios na camada intermediária	32
Camada de regularização (<i>dropout</i>)	não
Regularização L2	0.01
Função de ativação na camada intermediária	ReLU
Função de ativação na camada de saída	linear
Épocas	100

A construção do modelo foi realizada dinamicamente a partir dos hiperparâmetros definidos, permitindo a reutilização da mesma estrutura para diferentes arquiteturas. O treinamento foi então conduzido em lotes (*batch size* igual a 32), ao longo de múltiplas épocas, com avaliação contínua utilizando o conjunto de teste. Ao final, os modelos treinados foram avaliados individualmente e de forma consolidada, sendo armazenados juntamente com os artefatos necessários para inferência futura, como normalizadores e codificadores.

Os valores finais dos hiperparâmetros refletiram um compromisso entre capacidade de modelagem e generalização, necessário para capturar a relação indireta entre métricas

estáticas do código-fonte (como volume, esforço, operadores e operandos) e o tempo de execução em ambientes FaaS. A escolha de uma taxa de aprendizado moderada permite ajustes suficientemente sensíveis para explorar padrões complexos sem comprometer a estabilidade do treinamento, o que é essencial diante da variabilidade e do ruído presentes nesse tipo de dado.

A definição de uma única camada intermediária com 32 neurônios mostrou-se adequada para representar interações não lineares entre as métricas, evitando, ao mesmo tempo, um aumento excessivo da complexidade do modelo que poderia levar ao *overfitting*. A função de ativação ReLU contribui para destacar relações relevantes entre características do código, especialmente, em cenários nos quais pequenas variações estruturais podem gerar impactos significativos no desempenho. A ausência de *dropout*, combinada com a aplicação de regularização L2, indica que o modelo conseguiu controlar a magnitude dos pesos, sem necessidade de remoção estocástica de neurônios, preservando a capacidade de aprendizado das dependências presentes nos dados.

Por fim, a utilização de uma função de saída linear foi coerente com a natureza contínua do problema, permitindo que o modelo produzisse estimativas proporcionais às variações das métricas analisadas. Em conjunto, esses hiperparâmetros possibilitaram ao modelo aprender representações que correlacionaram a complexidade e a estrutura do código com seu comportamento em tempo de execução, mesmo diante da variabilidade inerente aos ambientes FaaS, tornando a predição mais robusta e consistente.

Essa abordagem permite não apenas a construção de modelos preditivos robustos, mas também a comparação sistemática entre diferentes arquiteturas e configurações, contribuindo para a seleção da melhor estratégia de predição de desempenho em ambientes FaaS.

4.7.2.3 Otimização

O processo de otimização de hiperparâmetros foi conduzido com o uso da biblioteca Hyperopt, sendo integrado ao fluxo de modelagem antes da etapa de treinamento final. Essa abordagem permitiu explorar de forma eficiente o espaço de busca de hiperparâmetros, por meio de estratégias de otimização baseadas em métodos probabilísticos. A cada iteração, o algoritmo avaliava diferentes combinações de parâmetros, como taxa de aprendizado, número de neurônios, funções de ativação e funções de perda, utilizando execuções de treinamento com número reduzido de épocas, de modo a equilibrar custo computacional e capacidade exploratória.

Ao final desse processo iterativo, o Hyperopt identificou a configuração de hiperparâmetros com melhor desempenho segundo as métricas definidas, permitindo selecionar a arquitetura mais adequada ao problema. Esses hiperparâmetros otimizados foram então

utilizados no treinamento completo do modelo, garantindo maior capacidade de generalização e melhor desempenho preditivo. Dessa forma, a etapa de otimização contribuiu diretamente para a robustez do modelo final, reduzindo a dependência de escolhas manuais e sistematizando a busca por configurações eficientes.

4.7.2.4 Treinamento

Para a execução do treinamento, o *dataset* foi dividido aleatoriamente em dados usados no treinamento em si e dados para validação. Dessa forma, 75% dos dados foram usados para treinamento e 25% para validação. Além disso, o modelo resultante do treinamento foi armazenado em um arquivo no formato HDF5, para depois ser transportado para o componente *Predictor* do projeto principal do *framework* Orama⁺⁺.

Antes de alcançar as definições de hiperparâmetros mostradas na Tabela 4.4, e os passos de pré-processamento elencados na Seção 4.7.2.1, o modelo passou por estados intermediários que precisaram ser avaliados, tanto para definir seu grau de qualidade quanto para identificar as necessidades de ajuste. Uma avaliação desse treinamento será detalhado na Seção 6.6.

Na Seção 4.8 serão apresentadas as principais tecnologias empregadas para a construção do *framework* Orama⁺⁺, desde as linguagens de programação usadas para implementar os componentes, como as soluções para orquestração de nuvem, disparo de requisições, entre outras.

4.8 Tecnologias Empregadas

A implementação do *framework* Orama⁺⁺ envolveu a integração de um conjunto heterogêneo de tecnologias voltadas ao provisionamento de infraestrutura, execução de *benchmarks*, análise estatística e predição de desempenho. Em vez de depender de uma única pilha tecnológica, o Orama⁺⁺ combina ferramentas consolidadas de desenvolvimento, orquestração e análise de dados, de modo a garantir flexibilidade, reprodutibilidade e escalabilidade experimental.

De forma geral, as tecnologias adotadas podem ser organizadas em quatro camadas principais: (i) desenvolvimento de aplicações e APIs, (ii) análise de dados e Aprendizado de Máquina, (iii) provisionamento e execução em nuvem, e (iv) geração de carga e coleta de métricas. Nas seções a seguir são apresentadas as principais ferramentas utilizadas em cada uma dessas camadas.

4.8.1 JavaScript

No contexto do Orama⁺⁺, o JavaScript foi utilizado principalmente na implementação das funções FaaS e das APIs dos casos de uso. Sua adoção se deve à ampla compatibilidade com provedores *serverless* e ao modelo assíncrono do Node.js [105], que favorece o tratamento eficiente de requisições concorrentes. Em relação à esse ecossistema, destacam-se as seguintes ferramentas:

- **Express** [106]: utilizado na definição de rotas e manipulação de requisições HTTP nos casos de uso baseados em API;
- **Knex.js** [110]: empregado como *query builder*, promovendo portabilidade entre diferentes sistemas de banco de dados;
- **Swagger** [107]: utilizado para documentação e teste das APIs, facilitando a padronização dos *endpoints*;
- **ESComplex** [121]: responsável pela extração de métricas estáticas de código, utilizadas como variáveis de entrada nos modelos preditivos;
- **React** e **Material UI** [103,104]: empregados na construção da interface do Orama⁺⁺, permitindo visualização e interação com os experimentos.

4.8.2 Python

Python foi utilizado no Orama⁺⁺ como linguagem principal para análise de dados e construção dos modelos preditivos. A sua adoção está diretamente relacionada ao ecossistema consolidado de bibliotecas para Ciência de Dados e Aprendizado de Máquina. As principais bibliotecas utilizadas incluem:

- **NumPy** e **Pandas** [124]: para manipulação e transformação de dados experimentais;
- **Scikit-learn** [125]: para modelagem estatística e algoritmos clássicos de Aprendizado de Máquina;
- **TensorFlow** [126]: utilizado na implementação dos modelos de *deep learning* empregados na predição;
- **Flask** [122]: empregado na disponibilização de serviços leves para integração entre os componentes do *framework*.

4.8.3 PostgreSQL

O PostgreSQL [112] foi utilizado como sistema de armazenamento relacional no Orama⁺⁺ principalmente para persistência de resultados experimentais e metadados. A sua escolha se deve à robustez, suporte a consultas complexas e capacidade de lidar com dados estruturados e semiestruturados (via JSON), características importantes para o armazenamento dos dados gerados pelos *benchmarks*.

Além disso, o PostgreSQL oferece vantagens em relação a outras opções por combinar conformidade com o padrão SQL, suporte a extensões e controle de concorrência eficiente, garantindo consistência e bom desempenho em execuções simultâneas. Em comparação a bancos NoSQL, mantém a flexibilidade com suporte a JSON, sem abrir mão das propriedades ACID, sendo mais adequado para cenários que exigem integridade e consultas analíticas mais complexas.

4.8.4 Terraform

O provisionamento dos ambientes em nuvem é realizado por meio do Terraform [113,135], adotado como ferramenta de Infraestrutura como Código (*Infrastructure-as-Code* (IaC), na sigla em inglês). Sua abordagem declarativa permite definir, versionar e reproduzir os ambientes experimentais de forma consistente, além de facilitar a orquestração em múltiplos provedores de nuvem.

Além disso, o Terraform se destaca em relação a outras ferramentas de *Infrastructure-as-Code* (IaC) por sua abordagem declarativa e ampla compatibilidade com múltiplos provedores de nuvem, permitindo a definição de ambientes de forma padronizada e reproduzível. Diferentemente de soluções mais restritas ou imperativas, facilita a manutenção, versionamento e portabilidade dos cenários experimentais.

4.8.5 Apache Kafka

O Apache Kafka [114] é utilizado como mecanismo de ingestão e transporte de dados experimentais. Sua arquitetura distribuída permite lidar com grandes volumes de dados gerados durante a execução dos *benchmarks*, garantindo escalabilidade e desacoplamento entre os componentes do *framework*.

Além disso, o Apache Kafka apresenta vantagens frente a outras soluções de mensageria por sua alta taxa de transferência, persistência de dados e capacidade de escalabilidade horizontal. Em comparação a sistemas tradicionais de filas, oferece maior robustez para lidar com fluxos contínuos de dados e garante melhor desacoplamento entre produtores e consumidores.

4.8.6 Apache JMeter

O Apache JMeter [115] é empregado na geração de carga e execução dos *benchmarks*. A ferramenta permite definir cenários controlados de teste, variando parâmetros como número de requisições, concorrência e padrões de acesso. Esses cenários são fundamentais para garantir reprodutibilidade experimental e coleta sistemática de métricas de desempenho.

Além disso, o Apache JMeter se diferencia de outras ferramentas de teste de carga por sua flexibilidade na definição de cenários e ampla capacidade de parametrização. Sua maturidade e suporte a diferentes protocolos permitem a criação de experimentos mais controlados e reprodutíveis, o que é essencial para a avaliação consistente de desempenho.

4.8.7 Docker

O uso de contêineres com Docker [127] garante portabilidade e isolamento dos componentes do Orama⁺⁺. Essa abordagem permite replicar ambientes de execução de forma consistente, independentemente da infraestrutura subjacente. O Docker Compose [127] é utilizado para orquestrar múltiplos serviços locais, simplificando a configuração e execução do ambiente experimental.

A integração dessas tecnologias resulta em uma infraestrutura capaz de automatizar, desde o provisionamento até a análise de desempenho de aplicações FaaS, oferecendo suporte à tomada de decisão sobre a escolha de provedores e configurações de execução. Trabalhos relacionados que abordam problemas semelhantes serão discutidos no Capítulo 5.

4.9 Implantação do *Framework* Orama⁺⁺

O processo de instalação do *framework* Orama⁺⁺ foi concebido de forma a simplificar a sua adoção, e permitir a rápida inicialização de um ambiente experimental completo para avaliação de cenários FaaS. A implantação é baseada em contêineres, utilizando *Docker* e *Docker Compose*, o que garante portabilidade, isolamento e reprodutibilidade do ambiente.

Em alto nível, o processo de instalação consiste na obtenção do código-fonte do projeto, na configuração de variáveis de ambiente e na inicialização dos serviços que compõem a arquitetura do *framework*. Essas variáveis incluem parâmetros de infraestrutura local (como portas e endereços de serviços internos), bem como credenciais de acesso aos provedores de nuvem suportados, tais como AWS, GCP, Microsoft Azure e Alibaba Cloud.

Uma vez configurado o ambiente, a execução dos serviços é realizada de forma automatizada por meio do *Docker Compose*, que instancia todos os componentes necessários do *framework*, incluindo módulos de orquestração, execução de *benchmarks*, coleta de métricas, persistência de dados e predição de desempenho. Após a inicialização, o sistema disponibiliza uma interface web que permite ao usuário interagir com os diferentes módulos, configurar experimentos e acompanhar sua execução.

Adicionalmente, o *framework* permite a extensão da capacidade de execução por meio da configuração de *workers* remotos, responsáveis pela execução distribuída de *benchmarks*. Essa abordagem possibilita a escalabilidade horizontal do processo experimental, sendo especialmente útil em cenários com alta carga de testes ou múltiplos experimentos concorrentes.

Por fim, destaca-se que o processo de instalação, bem como os requisitos específicos de configuração para cada provedor de nuvem, encontram-se detalhados na documentação oficial do projeto⁷. Essa documentação inclui instruções completas sobre dependências, permissões necessárias, estrutura de configuração e procedimentos de execução do *framework*. Um guia rápido pode ser consultado no Anexo I.

4.10 Evoluções do *Framework* Orama⁺⁺

A evolução do *framework* Orama⁺⁺, ao longo de suas diferentes versões, é apresentada na Tabela 4.5, na qual é possível notar um avanço progressivo tanto em termos de capacidades experimentais quanto de suporte à tomada de decisão.

Na versão 1.0 o *framework* estabeleceu a sua base metodológica, oferecendo suporte à execução de *benchmarks* estruturados e à análise estatística por meio de técnicas como projeto *fatorial 2^k* e teste *t* pareado. Apesar de garantir reprodutibilidade experimental, essa versão inicial apresentou limitações relacionadas à escalabilidade e à ausência de distribuição geográfica, restringindo a representatividade dos cenários avaliados.

A versão 2.0 introduziu um avanço significativo ao incorporar *workers* remotos coordenados via Apache Kafka, permitindo a execução distribuída dos experimentos. Essa evolução ampliou a capacidade de geração de carga, viabilizando testes com maior nível de concorrência e em diferentes regiões dos provedores de nuvem, aproximando os experimentos de condições reais de operação.

Na versão 3.0, o Orama⁺⁺ evoluiu de uma abordagem puramente experimental para uma abordagem híbrida, ao incorporar um modelo preditivo baseado em Redes Neurais. Esse modelo utiliza métricas extraídas do código-fonte para estimar o tempo de execução das funções, reduzindo a dependência exclusiva da execução de *benchmarks*. No entanto,

⁷Disponível em: <https://github.com/unb-faas/orama>

Tabela 4.5: Evolução do *Framework* Orama⁺⁺.

Versão	Principais Funcionalidades	Capacidades Experimentais	Limitações	Aprimoramentos
1.0	Execução de <i>benchmarks</i> estruturados; análise estatística com projeto <i>fatorial 2^k</i> e teste <i>t</i> pareado	Avaliação controlada de desempenho em ambientes FaaS com experimentos reprodutíveis	Execução restrita a um único ambiente; ausência de distribuição geográfica e baixa escalabilidade de carga	Estabelece a base experimental e metodológica do <i>framework</i>
2.0	Introdução de <i>workers</i> remotos coordenados via Apache Kafka	Execução distribuída de <i>benchmarks</i> ; suporte a alta concorrência e múltiplas regiões de nuvem	Maior complexidade operacional	Ampliação da escala experimental e maior representatividade de cenários reais
3.0	Incorporação de preditor de tempo de execução baseado em Redes Neurais	Estimativa de desempenho a partir de métricas extraídas do código-fonte	Modelo limitado ao conjunto inicial de casos de uso; ausência de predição de custo	Introdução da capacidade preditiva, reduzindo dependência exclusiva de execução empírica
4.0	Aprimoramento do modelo preditivo com novos casos de uso; inclusão de modelo de composição de custos por provedor	Predição conjunta de tempo de execução e custo em múltiplos provedores	Dependência da qualidade e diversidade dos dados de treinamento	Evolução para suporte à tomada de decisão, integrando desempenho e custo

a sua aplicação ainda é limitada ao conjunto inicial de casos de uso e não contempla a estimativa de custos.

Por fim, a versão 4.0 consolida o caráter decisório do *framework* ao expandir o modelo preditivo com novos casos de uso, e incorporar a estimativa de custos baseada nas configurações dos provedores de nuvem. Essa evolução permite a análise conjunta de desempenho e custo, tornando o Orama⁺⁺ uma ferramenta mais completa para apoiar decisões relacionadas à implantação de aplicações FaaS.

Em síntese, observa-se uma progressão clara do Orama⁺⁺ desde uma ferramenta voltada à execução e análise experimental (Versão 1.0), passando pela ampliação de escala (Versão 2.0) e introdução de predição (Versão 3.0), até alcançar um estágio mais avançado de suporte à decisão (Versão 4.0).

4.11 Limitações do *Framework* Orama⁺⁺

Apesar de fornecer uma infraestrutura robusta para avaliação de ambientes FaaS, o *framework* Orama⁺⁺ apresenta limitações inerentes ao seu escopo, às tecnologias adotadas e à heterogeneidade dos provedores de nuvem. A explicitação dessas limitações é fundamental para contextualizar os resultados obtidos e delimitar o alcance da proposta.

4.11.1 Dependência de APIs e Especificidades de Provedores

Uma das principais limitações do Orama⁺⁺ está relacionada à forte dependência das APIs e dos modelos operacionais específicos de cada provedor de nuvem. Embora o *framework*

busque padronizar a implantação e a execução dos casos de uso, diferenças significativas entre os provedores, tais como modelos de autenticação, nomenclatura de serviços, políticas de permissão e mecanismos de configuração, exigem adaptações específicas para cada plataforma.

Essa heterogeneidade dificulta a criação de uma abstração completamente uniforme, resultando na necessidade de código e artefatos de automação específicos por provedor, especialmente, no módulo de orquestração.

4.11.2 Dificuldades na Incorporação de Novos Provedores

A extensão do *framework* para suportar novos provedores de nuvem consiste em uma jornada significativa. A incorporação de um novo provedor não se limita à configuração de credenciais, mas envolve um conjunto de etapas complexas, dentre as quais se destacam:

- Implementação de novos artefatos de infraestrutura como código (por exemplo, *Terraform*), considerando os recursos específicos do provedor;
- Adaptação dos casos de uso para compatibilidade com os serviços equivalentes disponíveis (ou ausência destes);
- Integração com mecanismos de autenticação e autorização próprios do provedor;
- Ajustes no orquestrador para suportar novos fluxos de provisionamento e deprovisionamento;
- Tratamento de inconsistências em funcionalidades, tais como limites de execução, configuração de memória, suporte a linguagens e integração com serviços adjacentes.

Além disso, nem todos os provedores oferecem equivalentes diretos para serviços amplamente utilizados em outros ecossistemas. Um exemplo observado neste trabalho é a limitação na automação de cenários envolvendo DBaaS em determinados provedores, o que restringe a comparabilidade entre plataformas.

Outro fator relevante é a qualidade e a maturidade das ferramentas de automação disponíveis. Enquanto alguns provedores possuem amplo suporte em ferramentas como *Terraform*, outros apresentam suporte parcial ou menos estável, o que impacta diretamente a confiabilidade do processo de provisionamento automatizado.

4.11.3 Acoplamento entre Componentes

A arquitetura do Orama⁺⁺ é projetada de forma modular. Todavia, observa-se a existência de um certo grau de acoplamento entre componentes, especialmente entre o orquestrador, os casos de uso e os artefatos de infraestrutura. Esse acoplamento decorre da

necessidade de coordenar de forma consistente as diferentes etapas do ciclo experimental, como provisionamento, implantação, execução de *benchmarks* e coleta de métricas, em ambientes heterogêneos de múltiplos provedores de nuvem.

Na prática, os casos de uso dependem de convenções específicas definidas pelo *framework*, tais como parâmetros de entrada, estruturas de configuração e padrões de integração. De forma semelhante, os artefatos de infraestrutura como código frequentemente incorporam particularidades, tanto dos casos de uso quanto dos provedores, reduzindo o nível de abstração entre as camadas. Como consequência, modificações em um componente, como a introdução de novos parâmetros experimentais ou a adaptação de fluxos de execução, tendem a exigir ajustes em múltiplas partes do sistema, aumentando o esforço de manutenção.

Esse acoplamento também impacta a extensibilidade do *framework*, uma vez que a inclusão de novos tipos de experimentos ou serviços pode demandar alterações em componentes já existentes para garantir compatibilidade com o fluxo operacional. Ainda assim, parte desse comportamento é resultado de decisões de projeto que priorizam a automação e a integração entre módulos. Dessa forma, o acoplamento identificado não compromete o funcionamento do Orama⁺⁺, mas representa um fator relevante a ser considerado em sua evolução e manutenção.

4.11.4 Limitações na Cobertura de Cenários

Os casos de uso pré-configurados cobrem cenários representativos de aplicações FaaS, incluindo cargas *CPU-bound* e *I/O-bound*. No entanto, eles não esgotam a diversidade de aplicações reais, como *workflows* complexos, funções encadeadas (*function chaining*) ou integrações com serviços especializados (por exemplo, mensageria avançada ou *stream processing*).

Assim, embora o *framework* permita extensões, a inclusão de novos cenários pode demandar esforço adicional de desenvolvimento e adaptação. Esse esforço envolve não apenas a implementação das novas funções e integrações, mas também a definição de parâmetros experimentais consistentes, a garantia de reprodutibilidade dos testes e a adaptação dos mecanismos de coleta e análise de métricas.

4.11.5 Dependência de Configuração e Credenciais

O processo de utilização do Orama⁺⁺ requer a configuração adequada de credenciais e permissões nos provedores de nuvem. Essa etapa pode ser complexa, especialmente, em ambientes com políticas de segurança restritivas, e constitui um potencial ponto de falha na execução dos experimentos. Além disso, mudanças nas políticas de acesso ou nas APIs

dos provedores podem impactar diretamente o funcionamento do *framework*, exigindo manutenção contínua.

As limitações apresentadas evidenciam que, embora o Orama⁺⁺ reduza significativamente o esforço experimental, ele não elimina completamente a complexidade inerente à avaliação de ambientes multi-nuvem. Em particular, a dificuldade de incorporar novos provedores reforça a necessidade de mecanismos mais avançados de abstração e padronização entre plataformas FaaS.

Por outro lado, essas limitações também refletem desafios amplamente reconhecidos na literatura de Computação em Nuvem, especialmente, no contexto de interoperabilidade e portabilidade entre provedores. Dessa forma, o Orama⁺⁺ se posiciona como uma solução prática e extensível, ainda que sujeita às restrições impostas pelo ecossistema heterogêneo em que está inserido.

4.12 Considerações Finais

O *framework* Orama⁺⁺ consiste em uma plataforma para situações nas quais seja necessário escolher um provedor de nuvem para implantar uma aplicação FaaS. Por meio de um componente que emprega IA, o Orama⁺⁺ utiliza observações de desempenho do passado, registradas por *benchmarks* controlados pelo próprio *framework*, para estimar o tempo de execução de uma função, a partir do seu código-fonte.

Além de auxiliar diretamente na tomada de decisão, o *framework* contribui em processos de avaliação sistemática de ambientes de FaaS, fornecendo uma aplicação estruturada para execução de *benchmarks*, inclusive com o apoio de ferramentas de análise estatística diretamente aplicadas sobre os dados gerados pela ferramenta. Assim, para avaliar a solução proposta, diversos experimentos foram conduzidos. As metodologias utilizadas em cada um desses experimentos, assim como os testes realizados serão detalhadas no Capítulo 6.

Diante desse cenário, torna-se fundamental compreender como a abordagem proposta se posiciona em relação às soluções existentes na literatura. Assim, o Capítulo 5 apresenta uma análise dos trabalhos relacionados, destacando suas contribuições, limitações e as lacunas que motivaram o desenvolvimento do Orama⁺⁺.

Capítulo 5

Trabalhos Relacionados

Este capítulo apresenta uma análise dos trabalhos relacionados ao problema de apoio à tomada de decisão no provisionamento de aplicações em ambientes FaaS, com ênfase em abordagens de *benchmarking*, modelagem e predição de desempenho e custo. A Seção 5.1 apresenta a sistemática utilizada no levantamento bibliográfico. Na Seção 5.2 são analisados os trabalhos com foco em *benchmarking* em FaaS. Em seguida, na Seção 5.3 é apresentada a análise dos trabalhos com foco em predição em ambientes de FaaS. A Seção 5.4 faz uma análise comparativa entre os trabalhos, incluindo a solução proposta nesta tese. Por fim, a Seção 5.5 conclui este capítulo com as considerações finais.

5.1 Levantamento Bibliográfico

O levantamento bibliográfico foi conduzido de forma sistemática, seguindo princípios de reprodutibilidade e transparência, com o objetivo de identificar, selecionar e analisar trabalhos relevantes relacionados ao apoio à tomada de decisão em ambientes FaaS. A estratégia adotada contemplou a definição explícita das bases de dados, da *string* de busca, bem como dos critérios de inclusão e exclusão. A busca foi realizada nas seguintes bases digitais amplamente reconhecidas na área de Computação:

- IEEE Xplore;
- ACM Digital Library;
- Scopus;
- Web of Science;
- SpringerLink.

A seleção dessas bases se justifica pela sua abrangência, qualidade dos periódicos indexados e relevância para pesquisas em Computação em Nuvem, Sistemas Distribuídos

e Engenharia de Software. Para a recuperação dos trabalhos, foi definida uma *string* de busca estruturada, construída a partir de palavras-chave diretamente relacionadas ao escopo desta pesquisa. A *string* final utilizada foi:

```
("serverless" OR "Function-as-a-Service" OR "FaaS") AND ("performance prediction" OR "cost estimation" OR "resource provisioning" OR "performance modeling") AND ("multi-cloud" OR "multicloud" OR "multi-provider" OR "sky computing")
```

Essa *string* foi adaptada conforme a sintaxe específica de cada base, mantendo-se, entretanto, a mesma semântica. As buscas foram aplicadas sobre título, resumo e palavras-chave dos artigos. O processo de seleção dos trabalhos ocorreu em três etapas principais:

1. Triagem inicial: remoção de duplicatas e leitura de títulos e resumos para eliminação de trabalhos claramente fora do escopo;
2. Leitura completa: análise detalhada dos artigos selecionados na etapa anterior, avaliando sua contribuição, metodologia e aderência ao problema de pesquisa;
3. Seleção final: inclusão dos trabalhos que atendem integralmente aos critérios definidos.

Além disso, foram adotados os seguintes critérios de inclusão:

- Trabalhos que abordam explicitamente ambientes *serverless* ou FaaS;
- Propostas que envolvem *benchmarking*, modelagem ou predição de desempenho, custo ou comportamento;
- Estudos com validação experimental ou avaliação empírica;
- Publicações em periódicos ou conferências relevantes;
- Trabalhos publicados entre 2018 e 2026.

Como critérios de exclusão, foram considerados:

- Trabalhos que não tratam diretamente de FaaS ou *serverless*;
- Estudos puramente conceituais, sem validação experimental;
- Artigos com escopo restrito a tecnologias específicas sem possibilidade de generalização;
- Publicações redundantes ou versões preliminares de trabalhos, posteriormente, estendidos;
- Trabalhos com baixa relevância para o problema de tomada de decisão em provisionamento.

Adicionalmente, foi realizada uma busca complementar por encadeamento de referências, analisando as citações dos trabalhos selecionados, com o objetivo de identificar estudos relevantes não capturados diretamente pela *string* de busca. Esse procedimento contribuiu para ampliar a cobertura e reduzir possíveis vieses na seleção.

Como resultado desse processo, obteve-se um conjunto representativo de trabalhos que refletem as principais abordagens da literatura, organizados em duas vertentes principais: soluções de *benchmarking* para avaliação de plataformas FaaS e abordagens voltadas à modelagem e predição de desempenho, custo e comportamento de aplicações. Essa organização orienta a estrutura das seções subsequentes deste capítulo.

5.2 *Benchmarking* de Plataformas FaaS

Copik et al. [136] propõem o *Serverless Benchmark Suite* (Sebs), uma suíte de *benchmark* que avalia múltiplos provedores por meio de *workloads* representativos, coletando métricas como tempo de execução, uso de vCPU, memória e custo. A solução, embora abrangente, não oferece suporte à execução distribuída nem a análises estatísticas avançadas. Somu et al. [137] apresentam o *PanOpticon*, que permite avaliar configurações de recursos em diferentes plataformas, mas limita-se a experimentos específicos e a um conjunto restrito de cenários.

Grambow et al. [138] apresentam o *BeFaaS*, que propõe uma abordagem de *benchmarking* centrada em aplicações reais, aliada à execução federada entre múltiplos provedores de FaaS. O modelo consiste em um gerador de carga que injeta requisições em uma aplicação de *benchmark* composta por múltiplas funções interdependentes, representando um fluxo típico de aplicações distribuídas. Essa aplicação é então executada de forma paralela em diferentes plataformas FaaS, permitindo a comparação direta de comportamento entre provedores sob o mesmo padrão de carga. Os resultados obtidos são coletados e consolidados em um repositório central para posterior análise.

Apesar dessa abordagem mais próxima de cenários reais de uso, o *BeFaaS* não incorpora métodos estatísticos robustos para análise dos resultados, limitando a profundidade das conclusões obtidas. Por outro lado, trabalhos baseados em *microbenchmarks*, como o de Back e Andrikopoulos [139], concentram-se na avaliação isolada de aspectos específicos de desempenho e precificação em ambientes *serverless*.

No entanto, embora esse tipo de análise seja útil para caracterizações pontuais e comparações diretas entre provedores, ela abstrai fatores relevantes do ambiente real, como variabilidade geográfica, interferência entre execuções concorrentes e efeitos de escalonamento dinâmico. Dessa forma, tais estudos não capturam adequadamente o comporta-

mento sistêmico de aplicações mais complexas, limitando sua aplicabilidade em cenários práticos mais representativos.

Outros estudos ampliam a análise experimental em ambientes *serverless*. Kuhlenskamp et al. [140] propõem o método *Serverless Infrastructure Evaluation Method* (SIEM), voltado à avaliação sistemática do impacto de tarefas operacionais, como alterações de configuração e de código, em aplicações de *big data* executadas sobre plataformas FaaS. Diferentemente de abordagens baseadas apenas em medições de desempenho, o SIEM introduz uma estrutura metodológica orientada a objetivos de qualidade, considerando cenários reais de aplicação e definindo métricas específicas para capturar atributos como latência, disponibilidade e consistência de implantação.

Wen et al. [141] propõem uma abordagem abrangente para caracterizar plataformas FaaS, combinando análises qualitativas e quantitativas, apoiadas por *micro* e *macrobenchmarks*. A metodologia estrutura o estudo em etapas que incluem construção de taxonomia, avaliação de latência de inicialização, análise de eficiência de recursos e investigação de desempenho sob concorrência. Apesar da contribuição metodológica, as análises permanecem limitadas por um escopo experimental restrito e ausência de modelagem estatística estruturada. Essa ausência reduz a capacidade de extrapolação e predição, uma vez que os resultados são majoritariamente descritivos e baseados em medições pontuais. Dessa forma, embora a abordagem ofereça uma visão empírica relevante sobre o desempenho de provedores FaaS, ela não avança na construção de modelos analíticos ou preditivos que permitam apoiar decisões em cenários mais amplos ou dinâmicos.

Maissen et al. [142] propõem o *FaaS-dom*, um conjunto modular de *benchmarks* projetado para avaliar plataformas FaaS de forma sistemática e comparável. A abordagem busca ampliar a reprodutibilidade experimental ao oferecer suporte a múltiplas linguagens e provedores, com uma arquitetura que separa especificação, orquestração e abstração dos provedores. Embora verifique-se contribuições relevantes, o *FaaS-dom* apresenta limitações relacionadas à simplicidade das operações modeladas e à baixa integração com serviços nativos de nuvem, como armazenamento em objetos, bancos de dados gerenciados e sistemas de mensageria. Essa simplificação reduz a capacidade de representar cenários reais mais complexos e limita a análise de interações típicas de aplicações distribuídas modernas. Consequentemente, embora a arquitetura proposta favoreça a padronização e a reprodutibilidade, ela ainda carece de maior fidelidade em relação a cargas de trabalho do mundo real, o que restringe o alcance das conclusões obtidas.

No contexto de arquiteturas paralelas, Barcelona-Pons e García-López em [143] conduzem uma análise comparativa aprofundada de plataformas FaaS, com foco específico na capacidade de suportar computações altamente paralelas. A abordagem combina a construção de um *framework* comparativo com uma análise arquitetural detalhada de prove-

dores amplamente utilizados, complementada por experimentos empíricos que executam múltiplas invocações concorrentes para observar o comportamento real de paralelismo ao longo do tempo. Esse método permite visualizar a distribuição das execuções e identificar gargalos relacionados à alocação de recursos e ao escalonamento das funções.

Os resultados evidenciam que limitações estruturais dos provedores têm impacto direto no desempenho paralelo, destacando-se especialmente dois fatores centrais: as tecnologias de virtualização empregadas e as estratégias de escalonamento. Plataformas que utilizam virtualização mais leve e abordagens proativas de escalonamento tendem a apresentar melhor paralelismo, devido à maior elasticidade e à capacidade de provisionar recursos de forma mais ágil. Em contrapartida, abordagens mais conservadoras, como o empacotamento de múltiplas invocações em uma mesma instância, podem introduzir interferência de recursos e reduzir significativamente o grau de paralelismo efetivo.

Apesar da profundidade da análise arquitetural e experimental, a proposta não oferece suporte direto à experimentação configurável ou extensível por parte do usuário, limitando sua aplicabilidade como ferramenta de *benchmarking* reutilizável. O foco permanece na caracterização e compreensão do comportamento das plataformas avaliadas, sem disponibilizar mecanismos para definição flexível de cenários, parametrização de cargas de trabalho ou adaptação a diferentes contextos experimentais. Dessa forma, embora contribua para a identificação de fatores críticos que afetam o paralelismo em FaaS, a abordagem carece de generalização prática para estudos experimentais customizados.

A Tabela 5.1 apresenta um comparativo entre as principais características dos trabalhos da vertente de *benchmarking*. A análise das características permite observar que a maioria dos trabalhos oferece suporte a múltiplos provedores, mas há lacunas relevantes nas demais características. Poucos utilizam *workloads* reais ou execução distribuída, e abordagens com *microbenchmarks* permanecem limitadas a análises pontuais. Destaca-se também a ausência de métodos estatísticos estruturados na maioria das propostas, reduzindo a robustez dos resultados. Embora algumas soluções priorizem reprodutibilidade, há pouca integração entre realismo, escalabilidade e profundidade analítica, além de uma exploração ainda limitada do paralelismo.

A análise apresentada na Tabela 5.1 evidencia que as abordagens de *benchmarking* em FaaS concentram-se majoritariamente na avaliação empírica, com limitações quanto à generalização dos resultados, ausência de modelagem estatística e reduzida capacidade preditiva. Essas características indicam uma lacuna na literatura no que diz respeito ao suporte a análises mais abrangentes e orientadas à previsão. Nesse sentido, a próxima seção discute trabalhos que avançam nessa direção, explorando técnicas de modelagem e predição de desempenho, custo e comportamento de aplicações em ambientes FaaS.

Tabela 5.1: Comparação das características dos trabalhos de *benchmarking* em FaaS.

Trabalho	Multi-nuvem	Workloads reais	Microbench	Execução distribuída	Análise estatística	Reprodutibilidade	Foco em paralelismo
Sebs [136]	✓	✓	-	-	-	✓	-
PanOpticon [137]	✓	-	-	-	-	-	-
BeFaaS [138]	✓	✓	-	✓	-	-	-
Back e Andrikopoulos [139]	✓	-	✓	-	-	-	-
SIEM [140]	-	✓	-	-	-	✓	-
Wen et al. [141]	✓	✓	✓	-	-	-	-
FaaS-dom [142]	✓	-	-	-	-	✓	-
Barcelona-Pons e García-López [143]	✓	✓	-	✓	-	-	✓
Orama⁺⁺ (Este trabalho)	✓	✓	-	✓	✓	✓	✓

5.3 Modelagem e Predição de Desempenho, Custo e Comportamento de Aplicações

Na vertente relacionada à predição, Tomaras et al. [10] introduzem o *SLOPE*, um *framework* voltado à estimativa da alocação de recursos em ambientes FaaS, com o objetivo de atender restrições de desempenho e reduzir custos operacionais. A abordagem combina modelos de Redes Neurais com mecanismos de reutilização de conhecimento a partir de execuções anteriores, permitindo prever o número adequado de instâncias de funções para satisfazer taxas de invocação e *deadlines* definidos. Adicionalmente, o *framework* incorpora uma estratégia baseada em similaridade estrutural entre aplicações, utilizando *graph edit distance* para estimar o comportamento de aplicações mesmo na ausência de dados históricos.

Apesar de sua efetividade na redução de custos e na adaptação a cargas dinâmicas, a proposta concentra-se predominantemente em ambientes de execução baseados em contêineres e em plataformas específicas de orquestração, o que pode limitar sua generalização para outros modelos de FaaS. Além disso, embora avance em relação a trabalhos puramente descritivos ao incorporar técnicas preditivas, o *SLOPE* mantém forte dependência de características estruturais das aplicações e do ambiente subjacente, o que pode impactar sua aplicabilidade em cenários mais heterogêneos ou com diferentes abstrações de execução.

Horovitz et al. [8] propõem o *FaaSTest*, uma abordagem que combina técnicas de Aprendizado de Máquina com seleção dinâmica de provedores FaaS, visando minimizar

custo e latência de execução. O *framework* baseia-se na coleta e na análise de métricas históricas de desempenho para treinar modelos capazes de prever o comportamento das funções em diferentes plataformas, permitindo escolher, em tempo de execução, o provedor mais adequado para cada invocação. Essa estratégia explora a heterogeneidade entre provedores, buscando otimizar decisões de implantação com base em dados empíricos acumulados.

Apesar dos ganhos potenciais em eficiência, a proposta apresenta limitações importantes, especialmente, pela forte dependência de dados históricos para treinar os modelos. Isso pode comprometer sua eficácia em cenários com pouca informação prévia ou mudanças abruptas de comportamento. Além disso, o *FaaSTest* não considera características intrínsecas do código-fonte das funções, restringindo sua capacidade de generalização para novas aplicações ou padrões de execução distintos. Dessa forma, embora avance na seleção automatizada de provedores, sua aplicabilidade permanece condicionada à disponibilidade e representatividade dos dados coletados.

Dakkak et al. [144] introduzem o TrIMS (*Transparent and Isolated Model Sharing*), um sistema desenvolvido para otimizar a inferência de modelos de *Deep Learning* em ambientes FaaS. O foco central da solução é mitigar o problema do “cold start” e reduzir a latência de ponta a ponta, permitindo que múltiplos processos compartilhem instâncias de modelos carregados na memória de forma isolada. Ao evitar o carregamento redundante de pesos e tensores a cada nova requisição, o TrIMS alcança ganhos expressivos de desempenho, especialmente em fluxos de trabalho com demandas intermitentes.

Entretanto, apesar da eficiência na redução de latência e no uso de recursos, o TrIMS prioriza o isolamento e o compartilhamento transparente, não incorporando uma modelagem preditiva do tempo de execução. A solução foca na otimização da infraestrutura de carregamento e execução imediata, deixando uma lacuna em cenários que exigem previsões determinísticas ou estimativas de custo computacional antes do escalonamento das funções. Assim, o trabalho estabelece uma base sólida para a eficiência operacional, mas depende de heurísticas externas para o planejamento de capacidade a longo prazo.

Eismann et al. [9] introduzem o Sizeless, uma estratégia preditiva projetada para determinar configurações ótimas de memória em funções *serverless* utilizando apenas monitoramento passivo. A abordagem destaca-se por coletar dados de telemetria a partir de uma única configuração de implantação, eliminando a necessidade de testes exaustivos em múltiplos níveis de memória (*profiling* ativo). Ao explorar métricas de utilização de recursos como vCPU, memória e I/O, o método aplica modelos de regressão multialvo para estimar tanto o tempo de execução quanto o custo sob diferentes alocações, permitindo que desenvolvedores encontrem o equilíbrio ideal entre desempenho e gastos operacionais de forma eficiente.

Contudo, embora o Sizeless melhore a eficiência do dimensionamento, ele permanece fundamentalmente dependente de execuções previamente observadas e monitoramento em tempo de execução. A proposta foca estritamente no dimensionamento de recursos em vez de uma predição geral de tempo de execução para diferentes entradas de dados, e não considera características estáticas do código-fonte. Essa dependência de rastros históricos limita sua aplicabilidade no estágio “dia zero” da implantação. Em contraste, abordagens que incorporam características estruturais da aplicação podem operar com menos rastros e oferecer uma generalização superior entre diferentes tipos de cargas de trabalho.

Pandey e Kwon [145] propõem o FuncMem, um *framework* projetado para mitigar a latência de *cold start* em ambientes de computação *serverless* por meio de uma estratégia dupla: predição de memória necessária e execução adaptativa de tarefas. Diferente de abordagens que dependem apenas de dados históricos de infraestrutura, o FuncMem utiliza modelos de Aprendizado de Máquina para prever a quantidade ideal de memória para cada invocação de função. Essa antecipação permite que o sistema gerencie recursos de forma proativa, evitando tanto o desperdício por superestimativa quanto falhas de execução por falta de memória, otimizando o ciclo de vida das funções desde o seu acionamento.

Assim, embora o FuncMem apresente resultados superiores na redução da latência em comparação com o escalonador padrão do OpenWhisk [146], a abordagem exige uma infraestrutura de monitoramento constante para alimentar seus modelos preditivos. O foco do trabalho reside na otimização do uso de memória e no tempo de resposta. Mas, similar a outras soluções de monitoramento passivo, a eficácia do sistema é máxima em ambientes com cargas de trabalho recorrentes, onde o histórico de execução provê dados suficientes para a precisão do modelo. Ainda assim, a integração de um escalonador ciente da memória representa um avanço significativo na tentativa de tornar o *serverless* mais viável para aplicações sensíveis a atrasos.

Filippini et al. [11] propõem um *framework* de modelagem de desempenho baseado em Aprendizado de Máquina projetado especificamente para ambientes de *Edge* [147] FaaS. O trabalho aborda a dificuldade de prever o consumo de recursos em sistemas descentralizados, onde a escassez de hardware exige um gerenciamento rigoroso para evitar sobrecargas. A solução foca na predição de métricas críticas, como utilização de vCPU, memória e consumo de energia, tanto no nível da função individual quanto no nível do nó de computação, permitindo uma visão holística do estado do sistema antes do escalonamento de novas tarefas.

Apesar da robustez na predição de métricas de infraestrutura e energia, a abordagem é centrada em monitoramento passivo e telemetria de sistema, dependendo da disponibilidade de dados de execução em tempo real para manter a precisão dos modelos. O foco

principal reside na eficiência operacional e na sustentabilidade (consumo energético) do nó de borda, o que a torna complementar a métodos que utilizam análise estática de código. Ao unir a caracterização de funções com a previsão de carga, o trabalho oferece uma base sólida para sistemas de orquestração que buscam minimizar a latência e maximizar a vida útil das baterias em dispositivos de borda.

Pusztai e Nastic [148] introduzem o ChunkFunc, um *framework* orientado a objetivos de nível de serviço que aborda a configuração dinâmica de fluxos de trabalho *serverless*. O diferencial da proposta reside na consciência sobre o tamanho dos dados de entrada, utilizando um modelo de desempenho que ajusta a alocação de recursos e a granularidade das funções para minimizar custos sem violar os tempos de resposta acordados. Diferente de abordagens estáticas, o ChunkFunc emprega otimização bayesiana para realizar um *profiling* eficiente, identificando as configurações ideais de memória e vCPU que satisfaçam os requisitos de latência para diferentes volumes de dados.

Entretanto, apesar da eficácia na gestão de recursos e na otimização operacional orientada em nível de serviço, o ChunkFunc é predominantemente centrado em monitoramento e depende de instrumentação contínua e dados de *profiling* prévio. Embora o modelo considere o tamanho da entrada como variável, ele não incorpora características estáticas do código-fonte ou descritores semânticos mais profundos da aplicação como variáveis preditivas. Essa característica restringe sua capacidade de generalização para funções completamente novas ou para fases iniciais de implantação nas quais o histórico de execução ainda é inexistente, mantendo a dependência de uma fase de calibração baseada em execuções reais.

A Tabela 5.2 mostra que a maior parte dos trabalhos analisados nesta seção prioriza a predição de desempenho, frequentemente associada à estimativa de custo e ao uso de técnicas de Aprendizado de Máquina. Observa-se também uma forte dependência de dados históricos de infraestrutura, o que restringe a aplicação dessas abordagens em cenários sem execuções prévias. Apenas o FaaSTest considera explicitamente ambientes multi-nuvem, enquanto os demais concentram-se em otimizações específicas, como dimensionamento de memória ou redução de latência de inicialização. Além disso, o uso de características estruturais das aplicações é pouco explorado, aparecendo de forma mais evidente apenas no SLOPE. De modo geral, predominam abordagens baseadas em monitoramento, com limitações quanto à generalização para novos contextos e aplicações.

Com base nessas observações, torna-se necessário avançar para uma análise comparativa mais abrangente, que não apenas contraste as abordagens entre si, mas também as posicione em relação ao foco deste trabalho. A próxima seção aprofunda essa discussão, destacando convergências, lacunas e diferenciais, de modo a evidenciar como a proposta desenvolvida se insere e contribui no contexto das soluções existentes.

Tabela 5.2: Comparação entre abordagens de modelagem e de predição em ambientes FaaS.

Trabalho	Predição de Desempenho	Predição de Custo	Aprendizado de Máquina	Multi-nuvem	Foco em Memória	Mitigação de <i>Cold Start</i>	Depende de Histórico da Infraestrutura	Depende de Características Estruturais
SLOPE [10]	✓	✓	✓	-	-	-	-	✓
FaaSStest [8]	✓	✓	✓	✓	-	-	✓	-
TrIMS [144]	-	-	-	-	-	✓	-	-
Sizeless [9]	✓	✓	✓	-	✓	-	✓	-
FuncMem [145]	✓	-	✓	-	✓	✓	✓	-
MLFaaS [11]	✓	-	✓	-	✓	-	✓	-
ChunkFunc [148]	✓	✓	✓	-	✓	-	✓	-
Orama⁺⁺ (Este trabalho)	✓	✓	✓	✓	✓	✓	-	-

5.4 Análise Comparativa

A Tabela 5.3 consolida a análise dos trabalhos correlatos em ambas vertentes, estabelecendo um comparativo direto entre as soluções existentes e o *framework* Orama⁺⁺. Os critérios de comparação foram selecionados para evidenciar não apenas as funcionalidades operacionais, mas também o rigor metodológico e a capacidade preditiva de cada abordagem.

A análise da Tabela 5.3 revela uma dicotomia clara na literatura. De um lado, ferramentas de *benchmarking* (como Sebs e BeFaaS) oferecem excelente suporte multi-nuvem e execução experimental, mas carecem de inteligência preditiva para apoiar decisões em tempo de projeto. De outro, modelos baseados em Aprendizado de Máquina (como Sizeless e SLOPE) focam na otimização de recursos, porém apresentam alta dependência de telemetria histórica e monitoramento passivo, o que limita sua eficácia no cenário de “dia zero”, ou seja, quando uma nova função é implantada e não há dados prévios de execução.

Um diferencial crítico identificado é a ausência da análise de código-fonte nas propostas correlatas. A maioria das soluções de predição trata a função como uma “caixa preta”, baseando-se estritamente em métricas de infraestrutura. O Orama⁺⁺ rompe essa barreira ao extrair características estáticas do código, permitindo que o modelo compreenda a complexidade da tarefa antes mesmo da primeira execução. Além disso, a capacidade de

Tabela 5.3: Comparação entre os trabalhos relacionados e o *framework* Orama⁺⁺.

Trabalho	<i>Benchmark</i>	Predição	Multi-nuvem	Configurabilidade	Análise Estatística	Análise de Código	<i>Zero-Monitoring</i>	Foco Principal
Sebs [136]	✓	-	✓	-	-	-	-	Avaliação Geral
PanOpticon [137]	✓	-	✓	-	-	-	-	Alocação de Recursos
BeFaaS [138]	✓	-	✓	✓	-	-	-	Aplicações Reais
Microbench. [139]	✓	-	✓	-	-	-	-	Desempenho Isolado
SIEM [140]	✓	-	✓	-	✓	-	-	<i>Big Data</i>
Wen et al. [141]	✓	-	✓	-	-	-	-	Caracterização FaaS
FaaS-dom [142]	✓	-	✓	-	-	-	-	Modularidade
Barcelona-Pons et al. [143]	✓	-	✓	-	-	-	-	Paralelismo
SLOPE [10]	-	✓	-	-	-	-	-	Provisionamento
FaaSTest [8]	-	✓	✓	-	-	-	-	Otimização de Custo
TrIMS [144]	-	-	-	-	-	-	-	Gestão de Memória
FuncMem [145]	-	✓	-	-	-	-	-	Redução de <i>Cold Start</i>
ML-FaaS [11]	-	✓	-	-	✓	-	-	<i>Edge Computing</i>
Sizeless [9]	-	✓	-	-	-	-	-	Dimensionamento
ChunkFunc [148]	✓	✓	-	✓	-	-	-	<i>Workflows</i> e SLO
Orama⁺⁺ (Este trabalho)	✓	✓	✓	✓	✓	✓	✓	Predição e Tomada de Decisão

predição sem monitoramento (*zero-monitoring*) posiciona o Orama⁺⁺ como uma solução proativa, capaz de estimar desempenho e custo em múltiplos provedores sem a necessidade de *profiling* exaustivo em cada ambiente.

Quanto ao rigor metodológico, observa-se que poucos trabalhos incorporam análise estatística robusta. Enquanto a maioria se limita a médias descritivas, o Orama⁺⁺ utiliza projeto *fatorial* 2^k e testes de hipótese (*t*-Student pareado), garantindo que as comparações entre provedores e as predições de desempenho possuam significância estatística. Essa abordagem unificada preenche a lacuna entre a experimentação empírica e a modelagem analítica, oferecendo um suporte à tomada de decisão que é, simultaneamente, baseado em dados e ciente das características intrínsecas da aplicação.

Conclui-se que, embora o estado da arte tenha avançado em métricas de isolamento e eficiência de custos, subsiste um *gap* na integração de características de carga, semântica de código e suporte multi-nuvem configurável. O *framework* Orama⁺⁺ endereça essas limitações, consolidando-se como uma ferramenta integral para o ciclo de vida de aplicações FaaS.

5.5 Considerações Finais

A revisão bibliográfica apresentada neste capítulo possibilitou identificar tendências e avanços no suporte à tomada de decisão em ambientes FaaS, evidenciando um ecossistema de soluções que evoluiu de ferramentas puramente descritivas para modelos preditivos complexos. Observou-se que as abordagens de *benchmarking* (como *Sebs* [136] e *BeFaaS* [138]) estabeleceram bases sólidas para a comparação de provedores, enquanto as propostas baseadas em Aprendizado de Máquina (como *SLOPE* [10] e *Sizeless* [9]) avançaram na otimização de custos e recursos.

Entretanto, foram identificadas lacunas na literatura, conforme abordado na Seção 5.4. O *framework* Orama⁺⁺ foi concebido para endereçar essas lacunas, integrando análise estática, suporte multi-nuvem nativa e um motor de predição ciente da carga de trabalho. Para demonstrar a eficácia dessa abordagem e validar suas contribuições, o próximo capítulo apresenta uma avaliação experimental detalhada do *framework*.

Capítulo 6

Avaliação do *Framework* Orama⁺⁺

Com o intuito de avaliar a utilidade do *framework* Orama⁺⁺, bem como o comportamento das suas funcionalidades, foram conduzidos seis experimentos voltados à avaliação dos casos de uso, pré-configurados em diferentes formatos de implantação do *framework* Orama⁺⁺. Para isso, a Seção 6.1 apresenta o Experimento 1, no qual AWS e GCP são avaliados em seis casos de uso do *framework* Orama⁺⁺ implantado localmente e em duas nuvens diferentes. Na Seção 6.2 é apresentado o Experimento 2, no qual AWS, GCP e Azure são avaliados em nove casos de uso do Orama⁺⁺ em uma implantação local.

Na Seção 6.3 é apresentado o Experimento 3, no qual AWS, GCP, Azure e Alibaba Cloud são avaliados em um único caso de uso do *framework*, alternando implantações de *workers* distribuídos e concentrados geograficamente. Na Seção 6.4 é apresentado o Experimento 4, no qual AWS, GCP, Azure e Alibaba Cloud são avaliados em um único caso de uso, o mais simples, porém com implantação do Orama⁺⁺ em cinco regiões diferentes no globo terrestre. Na Seção 6.5 é apresentado o Experimento 5, que é uma variação do Experimento 4, porém usando um caso de uso mais complexo.

A Seção 6.6 apresenta uma avaliação do modelo de Redes Neurais que foi treinado usando as informações obtidas nos experimentos de 1 a 5. Na Seção 6.7 é apresentado o Experimento 6, que amplia a variabilidade e a complexidade dos dados de *benchmark* por meio de um caso de uso científico (alinhamento de sequências genéticas). Na Seção 6.9 são apresentadas predições de tempo de execução e custo em diferentes cenários. Por fim, a Seção 6.10 sintetiza os resultados, enfatizando o aspecto evolutivo do Orama⁺⁺, e na Seção 6.11 são apresentadas as considerações finais deste capítulo.

6.1 Experimento 1

No Experimento 1, os provedores AWS e GCP foram avaliados em seis casos de uso do *framework* Orama⁺⁺ implantado localmente e em duas nuvens diferentes. Para isso,

foram definidos três cenários de teste, conforme mostrado na Figura 6.1. No Cenário 1, o *framework* Orama⁺⁺ foi instalado em uma máquina física local com o motor Docker instalado, e seis casos de uso foram implantados nas respectivas nuvens, ou seja, três na AWS e três na GCP. Os tipos de casos de uso utilizados neste experimento foram: Calculadora, API com armazenamento em objetos, e API com armazenamento em bancos de dados como serviço.

No Cenário 2, uma máquina virtual foi instanciada no provedor GCP que recebeu a instalação do motor Docker e, em seguida, a instalação do *framework* Orama⁺⁺. Assim como no Cenário 1, os seis casos de uso foram implantados nas respectivas nuvens, mas no Cenário 2 a distância entre o *framework* e os casos de uso da GCP é muito menor, pois ambos estão rodando dentro do mesmo provedor.

No Cenário 3, uma máquina virtual foi instanciada no provedor AWS, que recebeu a instalação do motor Docker e, em seguida, a instalação do *framework* Orama⁺⁺. Assim como no Cenário 2, no Cenário 3 há uma maior proximidade entre o *framework* e os casos de uso relacionados à AWS, uma vez que ambos estão no mesmo provedor.

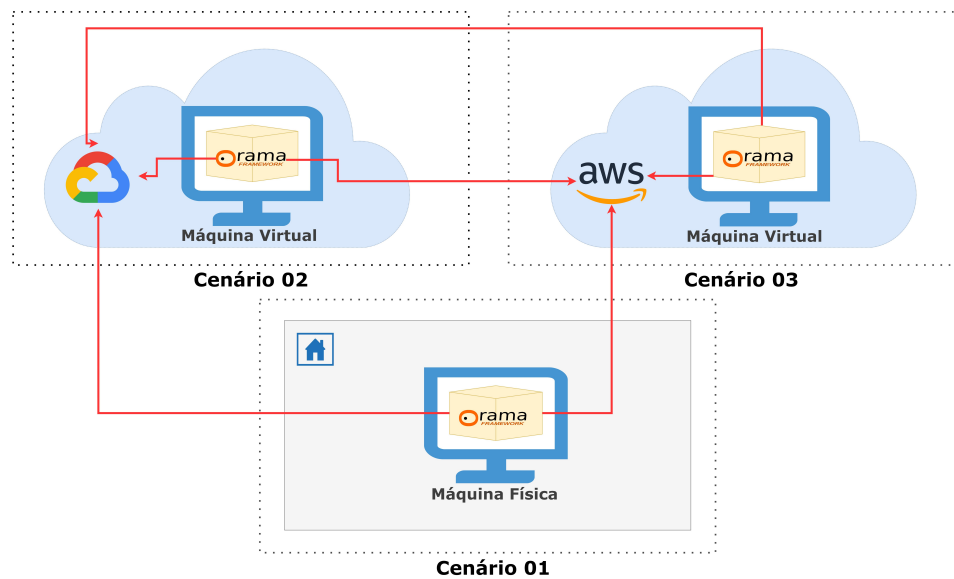


Figura 6.1: Experimento 1 - arquitetura dos testes.

É importante destacar que o componente de orquestração do Orama⁺⁺ utiliza nomes aleatórios para nomear os recursos na nuvem e, por isso, não há conflito de nomes, mesmo

Tabela 6.1: Experimento 1 - parâmetros dos cenários.

	Local	RAM	vCPU	Região	SO	Flavor
1	Local	16GB	8	Brasília/Br	Ubuntu 20.04	N/A
2	AWS	16GB	8	us-east-1	Ubuntu 20.04	c5.2xlarge
3	GCP	16GB	8	us-central1	Ubuntu 20.04	Custom

que os três cenários estejam ativos ao mesmo tempo. A Tabela 6.1 apresenta a configuração da máquina física utilizada no teste do Cenário 1, bem como as configurações das instâncias criadas para os testes dos Cenários 2 e 3.

Após a instalação do *framework* e a implementação dos casos de uso em cada provedor, baterias idênticas de testes foram executadas para todos os casos de uso em todos os cenários. A bateria de testes consiste em submeter o ambiente a dois níveis de carga concorrente. Um primeiro nível com 10 requisições concorrentes, e outro nível com 100 requisições concorrentes, sendo que cada teste foi repetido 25 vezes. A partir dessas baterias de testes foi possível visualizar os resultados do projeto *fatorial* 2^k e do teste *t* pareado, conforme analisado na próxima seção.

6.1.1 Análise dos Resultados do Experimento 1

Os resultados apresentados nesta seção, incluindo as imagens, foram publicados em [149] (Anexo XIII). A Figura 6.2 apresenta os resultados do caso de uso da calculadora, evidenciando a variabilidade dos tempos de latência ao longo das execuções, característica inerente a ambientes distribuídos e dependentes de rede. Ainda assim, é possível observar diferenças consistentes entre os provedores, tanto em termos de desempenho médio quanto de estabilidade. Isso sugere que diferenças nas estratégias dos provedores afetam o desempenho das aplicações que operam sobre seus serviços. Na Figura 6.2, embora possam ser observadas variações nos tempos de execução (latência) em cada rodada, é a latência média (apresentada acima de cada gráfico) que permite comparar os resultados entre os provedores. Neste caso, quanto menor é o tempo médio, melhor tende a ser a experiência do usuário.

De forma complementar, os resultados indicam a existência de um *trade-off* entre desempenho e confiabilidade, uma vez que determinados provedores apresentaram menor latência média, porém com maiores taxas de falha. Ou seja, ainda que a estratégia de um provedor priorize a redução do tempo médio de execução, é preciso observar a ocorrência de falhas, já que as características da aplicação podem não tolerar esse comportamento, sendo, dessa forma, mais adequado implantar a aplicação em um provedor mais estável, mesmo que ofereça tempos médios de execução superiores.

Nos cenários que envolvem integrações com serviços externos, como armazenamento em objetos (Figura 6.3) e DBaaS (Figura 6.4), os resultados evidenciam ainda mais a influência da infraestrutura subjacente, especialmente da camada de rede e da proximidade entre os serviços. Observa-se que a localização da execução do experimento (local, dentro da nuvem do provedor ou em nuvem distinta) impacta diretamente os tempos de resposta, reforçando a importância de considerar aspectos de implantação ao tomar decisões de provisionamento.

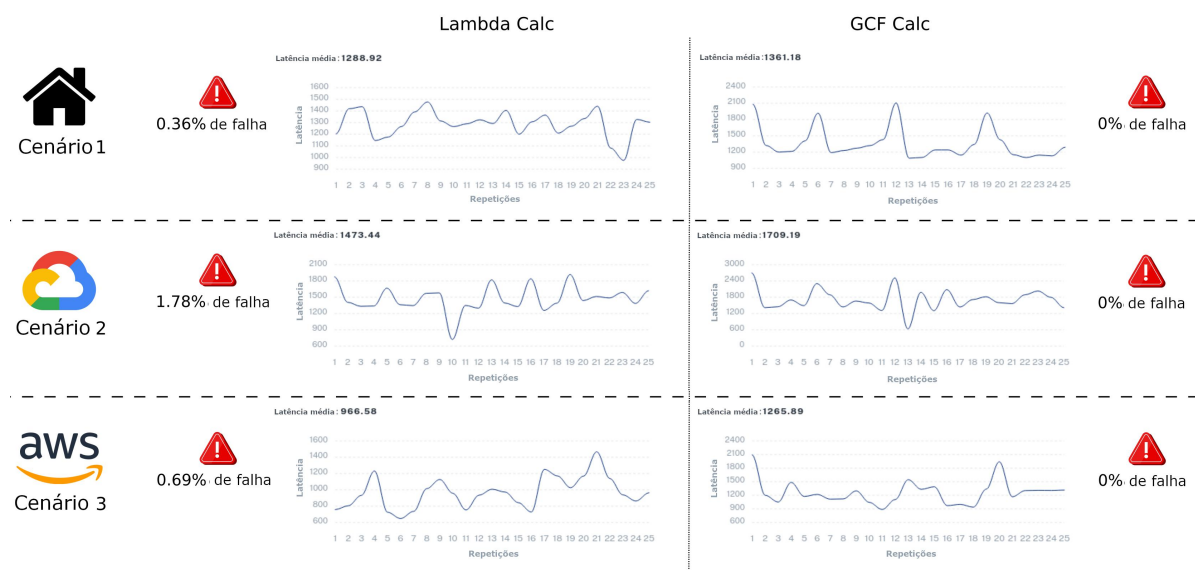


Figura 6.2: Experimento 1 - resultado do caso de uso Calculadora.

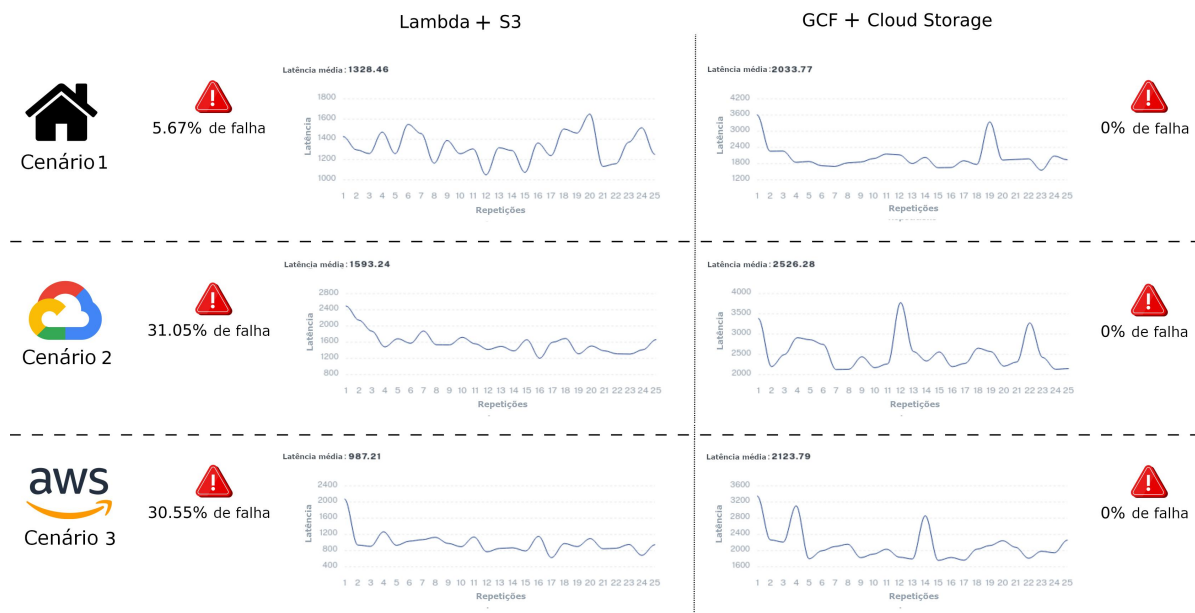


Figura 6.3: Experimento 1 - resultado do caso de uso API com armazenamento em objetos.

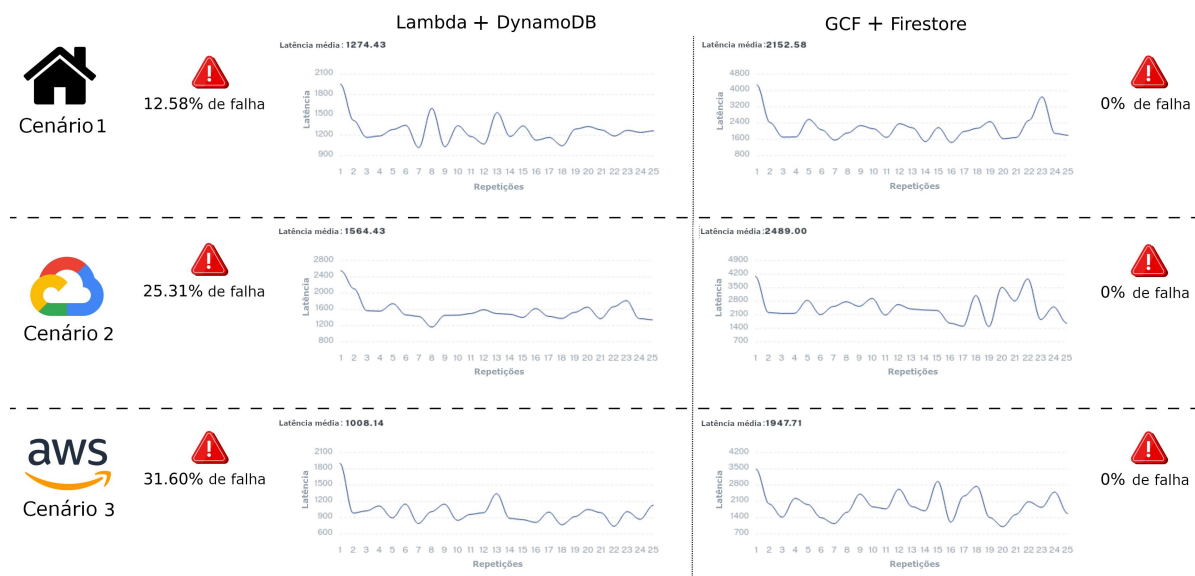


Figura 6.4: Experimento 1 - resultado do caso de uso API com DBaaS.

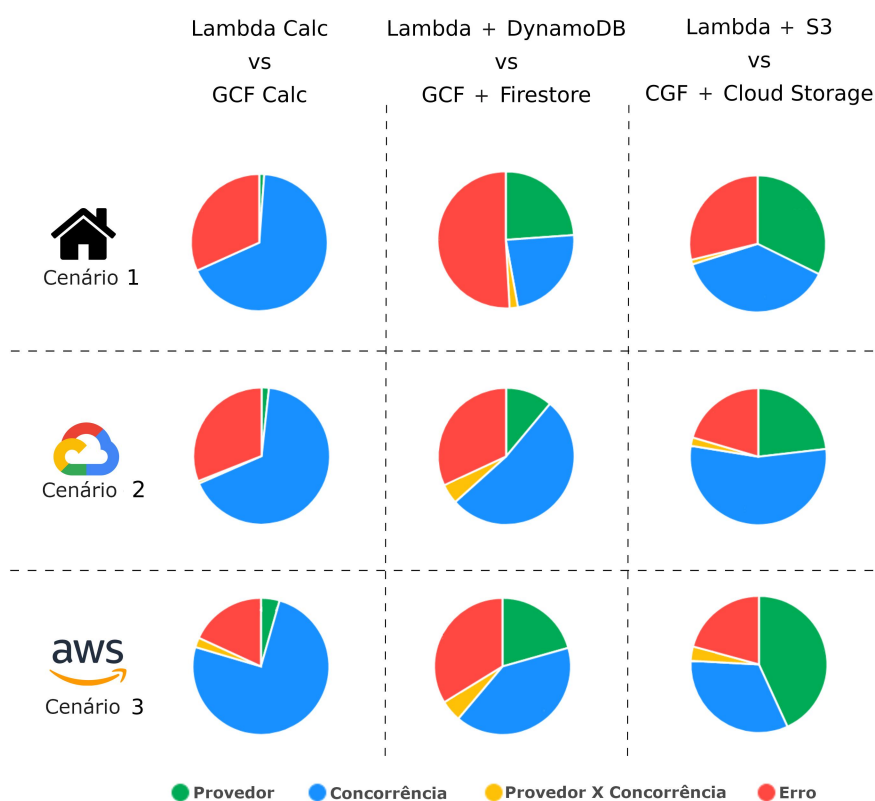


Figura 6.5: Experimento 1 - resultado do projeto *fatorial 2^k*.

A análise por meio de projeto *fatorial* 2^k , apresentada na Figura 6.5, permitiu identificar os principais fatores que influenciam o desempenho observado. De maneira geral, os resultados indicam que o nível de concorrência exerce maior impacto sobre o desempenho das aplicações do que o próprio provedor de nuvem, embora a influência relativa desses fatores varie conforme o cenário e o tipo de aplicação. Adicionalmente, foi possível observar a presença de efeitos associados a fatores não controlados, como variações da rede, especialmente em cenários externos aos provedores.

Tabela 6.2: Resultados dos testes t pareados no Experimento 1.

Caso de uso	Cenário								
	1 - Local			2 - GCP			3 - AWS		
	Dif.	Teste t	Conf.	Dif.	Teste t	Conf.	Dif.	Teste t	Conf.
Calculadora	72.26	Significante	80%	235.75	Significante	97.50%	299.32	Significante	99.95%
Armazenamento em Objetos	705.31	Significante	99.95%	933.04	Significante	99.95%	1136.58	Significante	99.95%
DBaaS	878.15	Significante	99.95%	924.57	Significante	99.95%	939.57	Significante	99.95%

A Tabela 6.2 apresenta os resultados dos testes t pareados realizados entre pares de casos de uso do mesmo tipo, nos quais foi possível identificar diferenças entre as latências médias. Observa-se que, em todos os testes t , a diferença foi considerada significativa. A maioria das diferenças conseguiu significância no teste t com nível de confiança de 99.95%. Apenas nos casos de uso relacionados à calculadora, nos Cenários 1 e 2, o grau de confiança com que a diferença foi aprovada no teste t foi de 80% e 97.50%, respectivamente.

De forma geral, os resultados do Experimento 1 demonstraram o valor do *framework* Orama⁺⁺ ao viabilizar a execução automatizada de experimentos, a coleta sistemática de métricas e a aplicação de técnicas estatísticas para análise comparativa. Essa abordagem permitiu identificar não apenas diferenças de desempenho entre provedores, mas também compreender os fatores que mais impactam os resultados, oferecendo suporte mais robusto à tomada de decisão no provisionamento de aplicações em ambientes FaaS.

6.2 Experimento 2

No Experimento 2 o *framework* Orama⁺⁺ foi instalado em uma máquina física localizada em Brasília/Brasil. Este computador possui 16GB de memória RAM, processador Intel Core i5 e sistema operacional Linux Mint 20. Após a instalação do *framework*, nove casos de uso foram implementados, bem como a configuração das credenciais dos provedores envolvidos nos casos de uso, ou seja, AWS, GCP e Azure. Os tipos de casos de uso utilizados neste experimento foram: Calculadora, API com armazenamento em objetos e API com armazenamento em bancos de dados como serviço.

Todos os nove casos de uso foram implantados em seus respectivos provedores. Na AWS foi utilizada a região *US-East*, na GCP a região *US-central1* e na Azure a região *West US*. Uma vez provisionados, cada caso de uso foi submetido a baterias de testes utilizando níveis de concorrência de crescimento exponencial, iniciando com 1 até 512 requisições concorrentes ao respectivo serviço. Assim, foram realizados testes sem concorrência (apenas uma requisição), 2, 4, 8, 16, 32, 64, 128, 256 e 512 ativações concorrentes ao serviço.

Essas baterias de testes foram executadas e controladas pelo *framework* Orama⁺⁺ a partir de um único comando para iniciar os testes em cada caso de uso, por meio da interface gráfica fornecida pelo *framework*. Antes de cada bateria de testes, o *framework* executa uma requisição prévia para que os provedores preparem efetivamente a infraestrutura para a execução das baterias. Essa primeira execução sofre com o efeito de *cold start* e seu tempo de execução é medido pelo *framework* Orama⁺⁺, permitindo uma comparação entre os provedores.

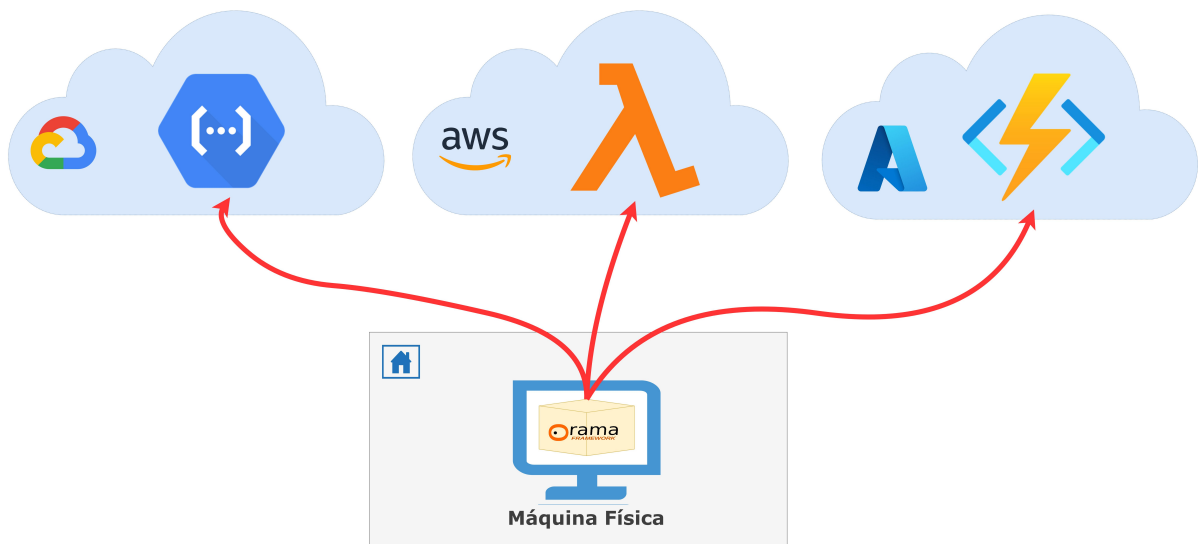


Figura 6.6: Experimento 2 - arquitetura dos testes.

A Figura 6.6 ilustra a arquitetura utilizada neste Experimento 2. Nessa figura, é possível observar que a instalação do *framework* na máquina física aciona os serviços FaaS dos provedores envolvidos no experimento. Cada bateria de testes foi configurada para ser repetida 10 vezes e, assim, o *framework* Orama⁺⁺ conduz o processo de teste utilizando esse parâmetro. Os testes comparativos entre os *benchmarks* realizados em cada caso de uso consideram as médias de latência obtidas em todas as repetições de cada nível de concorrência. O *framework* também calcula eventuais falhas nas requisições, permitindo uma análise comparativa dessas ocorrências de erro.

Após a execução das baterias de testes, nove projetos fatoriais foram criados no *framework* Orama⁺⁺ comparando os menores e os maiores níveis de concorrência, ou seja, 1 e 512, e as combinações entre os provedores, a fim de analisar a influência dos fatores “Provedor” e “Concorrência” nos testes realizados. A diferença de latência verificada em cada projeto *fatorial* 2^k teve sua significância estatística analisada utilizando a funcionalidade de teste *t* pareado oferecida pelo *framework* Orama⁺⁺. A partir dessas baterias de testes foi possível visualizar os resultados, conforme analisado na próxima seção.

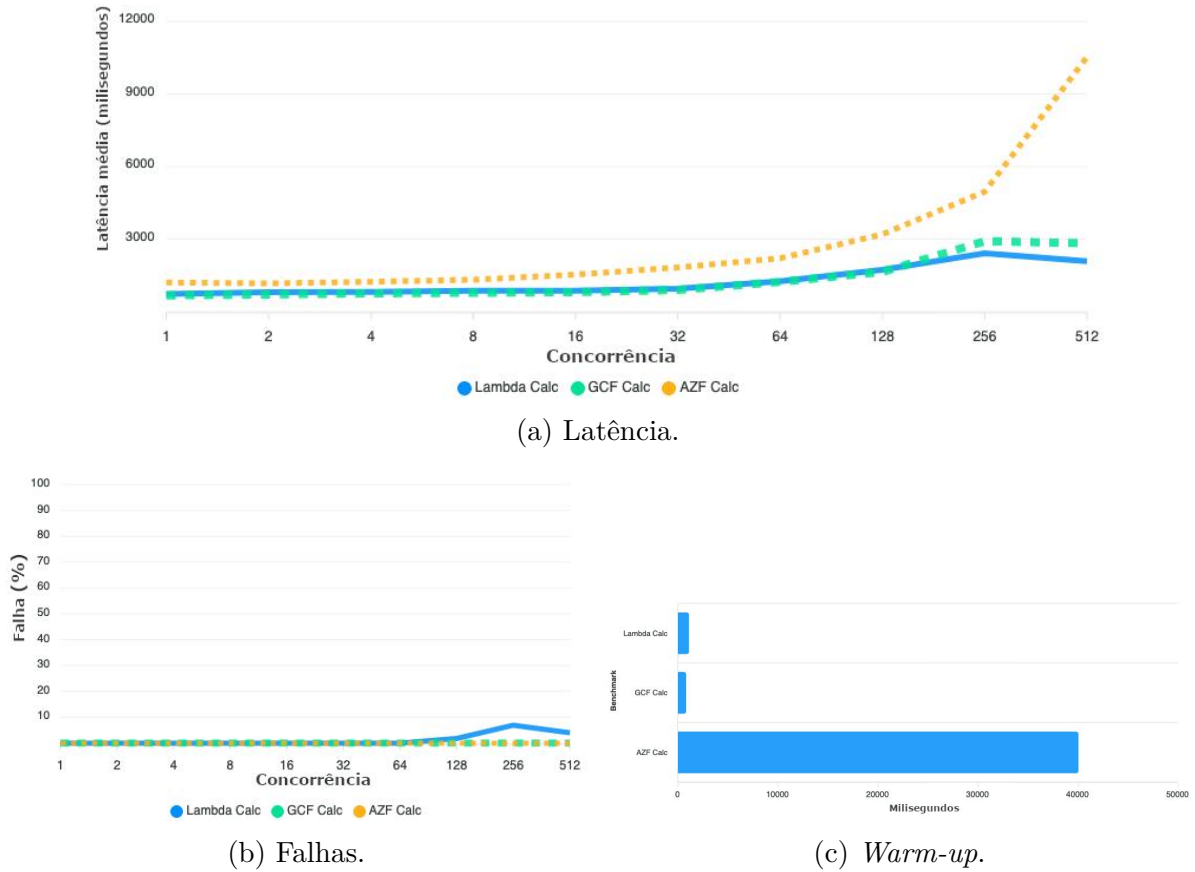


Figura 6.7: Experimento 2 - resultados do caso de uso Calculadora.

6.2.1 Análise dos Resultados do Experimento 2

Os resultados apresentados nesta seção, incluindo as imagens, foram publicados em [150] (Anexo XVIII). Os resultados do caso de uso da calculadora são apresentados na Figura 6.7. Em termos de latência (Figura 6.7a), observa-se que os provedores apresentam comportamento semelhante em baixos níveis de concorrência, mantendo estabilidade até determinado limiar. Contudo, à medida que a concorrência aumenta, surgem diferenças mais expressivas, evidenciando limitações na capacidade de escalonamento de alguns

provedores. Enquanto os provedores Lambda e GCF mantêm tempos médios de execução próximo e mais baixos, mesmo perante níveis mais elevados de concorrência, o AZF apresenta aumento no tempo médio de execução à medida que a concorrência aumenta. É possível observar uma leve queda no tempo médio de execução após o nível 256 de concorrência nos Lambda e no GCF, sugerindo que sob esse nível de concorrência ambos provedores implementam escalada nas suas infraestruturas. Já no AZF, como o tempo médio de execução cresce continuamente, infere-se que o provedor não implementou escalada no ambiente durante a bateria de testes.

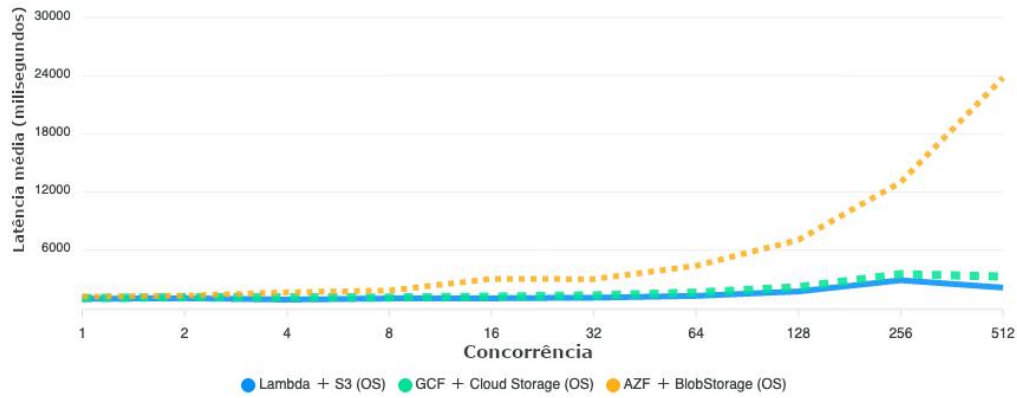
A análise das falhas (Figura 6.7b) mostra que o melhor desempenho em latência nem sempre está associado à maior confiabilidade, especialmente em níveis elevados de carga. É possível observar que o Lambda apresentou falhas após o nível 32 de concorrência, demonstrando que o provedor teve dificuldades para processar as requisições em níveis mais elevados de demanda. Mais uma vez, após o nível 256 houve uma queda nas falhas, evidenciando que o reforço na infraestrutura (promovido pela escalada) surtiu efeito também na taxa de falhas.

Os resultados de *warm-up* (Figura 6.7c) evidenciam diferenças significativas no tempo de inicialização, refletindo o impacto do fenômeno de *cold start*. Como pode ser observado, o provedor AZF apresentou um tempo de inicialização bem superior aos demais provedores, evidenciando que a estratégia de carregamento desse provedor difere consideravelmente dos demais provedores sob análise.

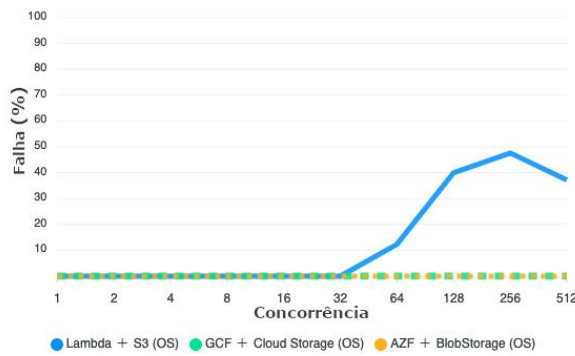
Tabela 6.3: Experimento 2 - resultados do teste *t* pareado para o caso de uso Calculadora.

Aspecto	AWS x Azure	AWS x GCP	GCP x Azure
Diferença	5173.65	506.55	4667.10
Desvio Padrão	496.24	253.29	520.37
Graus de Liberdade	8.56	13.70	10.39
99.95%	(2749.60 , 7597.69)	(-547.82 , 1560.92)	(2313.26 , 7020.94)
Significativo	Sim	Não	Sim
95.00%	(4258.66 , 6088.63)	(59.75 , 953.35)	(3727.56 , 5606.63)
Significativo	Sim	Sim	Sim

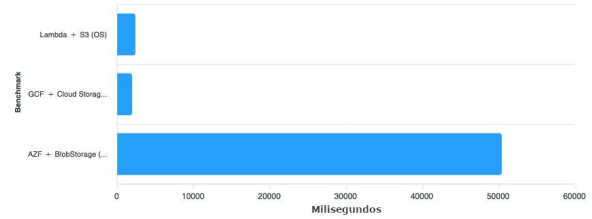
Para o caso de uso com armazenamento em objetos (Figura 6.8), os resultados reforçam tendências semelhantes. A latência (Figura 6.8a) cresce com o aumento da concorrência, porém de forma desigual entre os provedores. Assim como observado no caso de uso da calculadora, após o nível 256 de concorrência os provedores Lambda e GCF escalam suas infraestruturas, o que afeta o tempo médio de execução. Por outro lado, o AZF não aparentou implementar escala durante os testes. A análise das falhas (Figura 6.8b) evidencia maior robustez em alguns provedores, enquanto outros apresentam degradação



(a) Latência.



(b) Falhas.



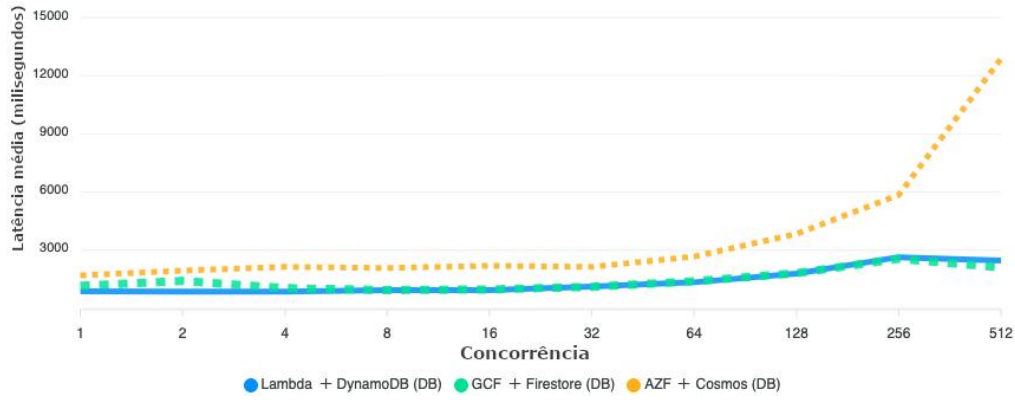
(c) Warm-up.

Figura 6.8: Experimento 2 - resultados do caso de uso API com armazenamento em objetos.

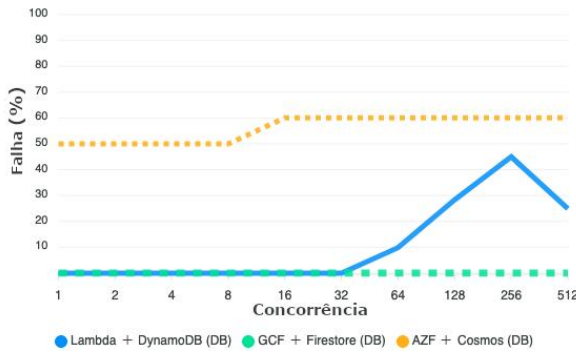
progressiva sob carga. Os tempos de *warm-up* (Figura 6.8c) novamente indicam diferenças estruturais relevantes entre as plataformas.

Nos casos de uso com DBaaS (Figura 6.9), observa-se maior variabilidade nos resultados. A latência (Figura 6.9a) reflete a complexidade adicional da integração com serviços externos, enquanto as taxas de falha (Figura 6.9b) evidenciam gargalos em cenários de alta concorrência nos provedores Lambda e AZF. Mais uma vez todos os provedores apresentaram aumento progressivo dos tempos médios de execução, com percepção de escalada no Lambda e no GCF, enquanto no AZF não se verificou tal comportamento. Os tempos de *warm-up* (Figura 6.9c) seguem o padrão observado anteriormente, reforçando diferenças entre provedores na inicialização dos ambientes, em especial no AZF, que apresenta um tempo de inicialização consideravelmente mais elevado.

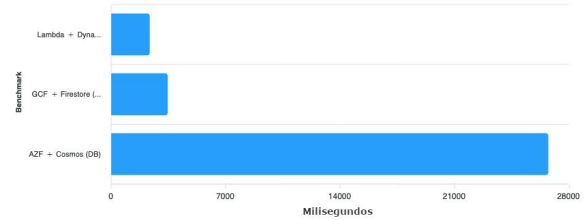
A análise por meio de projeto *fatorial* 2^k (Figura 6.10) evidencia que o fator *concorrência* exerce maior influência sobre os resultados do que o fator *provedor* na maioria dos cenários. Esse resultado reforça a importância da capacidade de escalonamento e do gerenciamento de carga em ambientes FaaS. Adicionalmente, a presença de erro estatístico



(a) Latência.



(b) Falhas.



(c) Warm-up.

Figura 6.9: Experimento 2 - resultados do caso de uso API com armazenamento em DBaaS.

em alguns casos indica a influência de fatores não controlados, como variações de rede e características internas das plataformas.

Complementarmente, foram realizados testes t pareados para avaliar a significância das diferenças entre os provedores. A Tabela 6.3 apresenta os resultados para o caso de uso de calculadora. Os resultados indicam que as diferenças entre AWS e Azure, bem como entre GCP e Azure, são estatisticamente significativas em altos níveis de confiança. Entre AWS e GCP a significância depende do nível de confiança considerado, refletindo a proximidade de desempenho entre esses dois provedores.

De maneira geral, os resultados do Experimento 2 evidenciam que não existe um provedor universalmente superior em todos os cenários. O desempenho observado depende do tipo de aplicação, do nível de concorrência e de fatores como inicialização e confiabilidade.

Por fim, os achados deste experimento evidenciam o potencial do *framework* Orama⁺⁺ como instrumento sistemático de investigação em ambientes FaaS. Ao integrar automação na execução dos cenários, padronização na coleta de dados e mecanismos analíticos avançados, o Orama⁺⁺ viabiliza uma análise mais aprofundada e confiável do comportamento

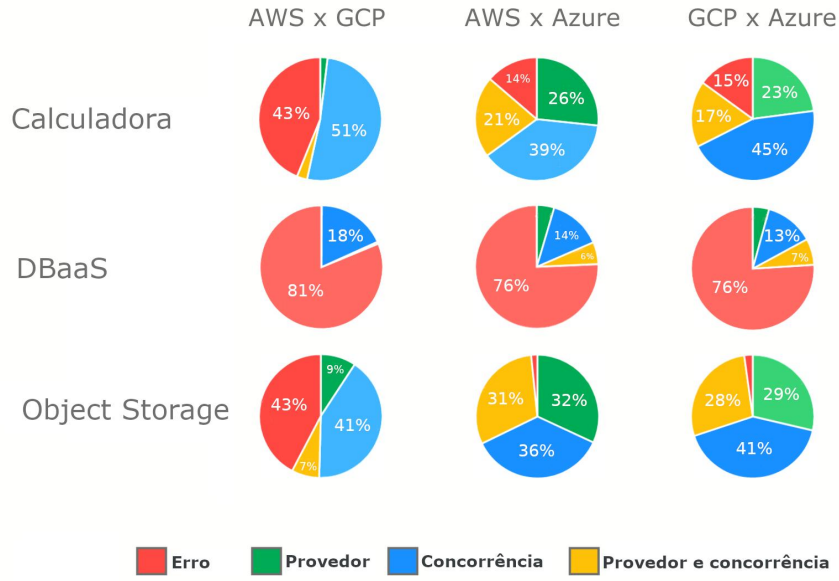


Figura 6.10: Experimento 2 - resultados do projeto *fatorial 2^k*.

dos provedores. Dessa forma, torna-se possível não apenas identificar diferenças de desempenho, mas também interpretar as causas subjacentes a essas variações, contribuindo de maneira mais fundamentada para o planejamento e a otimização do provisionamento em nuvem. Entretanto, os experimentos 1 e 2 foram realizados usando a versão do *framework* que era restrita à execução em apenas um *worker*, o que limitava a quantidade de requisições concorrentes, bem como cenários mais complexos, nos quais os *workers* fossem geograficamente distribuídos. A partir da próxima seção serão descritos os experimentos usando a arquitetura distribuída do Orama⁺⁺.

6.3 Experimento 3

O objetivo do Experimento 3 foi analisar o comportamento da arquitetura distribuída do *framework* Orama⁺⁺. Para isso, foi selecionado um caso de uso que tinha como objetivo garantir a execução correta do FaaS nos provedores, sem integração com outros serviços, como DBaaS ou armazenamento em objetos, que poderiam introduzir características específicas desses serviços, e desviar o foco da arquitetura propriamente dita. Portanto, foi escolhido o caso de uso que implementa uma calculadora matemática simples em cada um dos quatro principais provedores de FaaS suportados pelo *framework* Orama⁺⁺. Os provedores avaliados foram: AWS, GCP, Azure e Alibaba Cloud.

A arquitetura distribuída do *framework* Orama⁺⁺ permite que *workers* sejam implantados em diferentes regiões. Assim, neste experimento foram elaborados dois cenários.

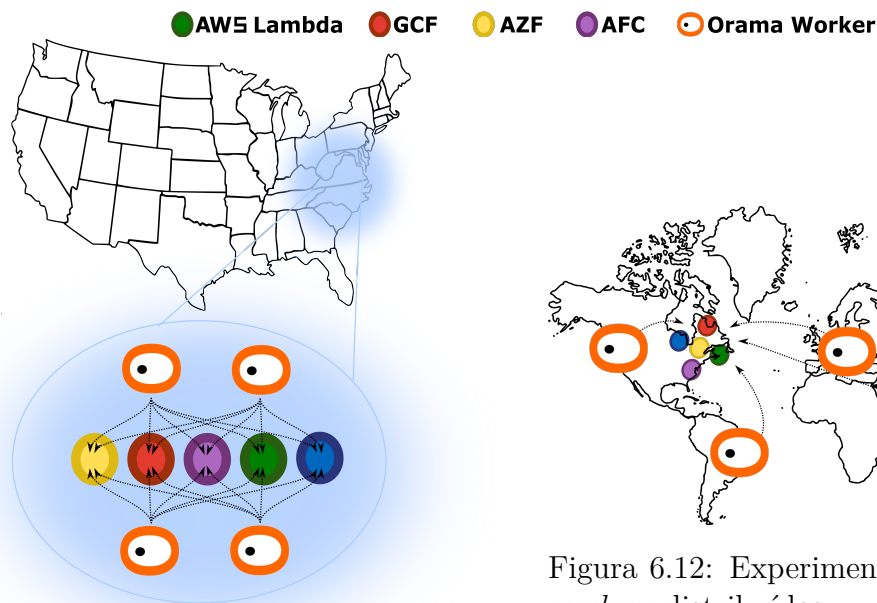


Figura 6.11: Experimento 3 - cenário com *workers* concentrados.

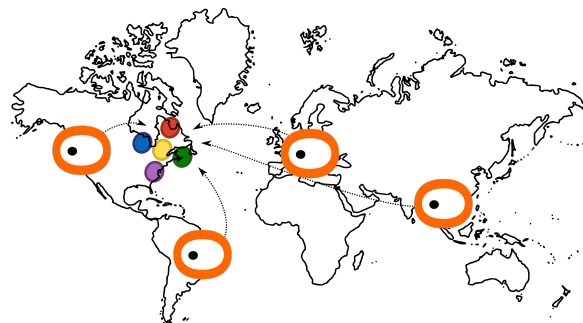


Figura 6.12: Experimento 3 - cenário com *workers* distribuídos.

No primeiro cenário, mostrado na Figura 6.11, todos os *workers* foram implantados na mesma região onde os ambientes FaaS alvo também foram provisionados (cenário de *workers* concentrados). No segundo cenário, os *workers* foram distribuídos em regiões geograficamente distantes, conforme mostrado na Figura 6.12. Com esses dois cenários, esperava-se demonstrar a influência da latência no FaaS, no entanto, seu comportamento deveria ser equivalente e demonstrar a mesma estratégia aplicada pelos provedores.

Ambos os cenários foram submetidos a cargas concorrentes de requisições, variando de apenas uma requisição simultânea até 2048 requisições, com crescimento exponencial. Assim, os serviços FaaS de cada provedor foram submetidos individualmente a cargas de 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 e 2048. Vale ressaltar que os testes realizados com a arquitetura *standalone* do *framework* Orama⁺⁺ foram limitados a um máximo de 512 requisições simultâneas, devido às limitações na abertura de conexões com os serviços FaaS impostas pela instância na qual o *framework* Orama⁺⁺ foi instalado. Como 2048 foi o nível máximo deste experimento, foi definida uma quantidade de quatro *Benchmark Worker*s, de forma que cada um fosse responsável por realizar no máximo 512 requisições concorrentes. Além disso, cada bateria de testes com diferentes níveis de concorrência foi repetida 10 vezes para estabelecer uma base estatística de dados para análise posterior.

No primeiro cenário (*workers* concentrados), o *framework* Orama⁺⁺ foi implantado em uma instância do *Compute Engine* da GCP na região *US-East*, com 16GB de RAM e 4 vCPUs, e cada um dos quatro *workers* também foi implantado na mesma região da

GCP. Os servidores tinham 4GB de RAM e 2 vCPUs cada. Para o segundo cenário, a instalação principal do *framework* Orama⁺⁺ foi mantida na mesma instância na região *US-East* da GCP utilizada no primeiro cenário. No entanto, os *workers* foram distribuídos em outras regiões da GCP, tais como *asia-northeast1* (Tóquio, Japão), *US-West* (*The Dalles*, Oregon, Estados Unidos), *europa-west2* (Londres, Inglaterra) e *southamerica-east1* (Osasco, Brasil). Todas as instâncias da GCP utilizadas neste experimento executaram o sistema operacional Debian 11. Os serviços FaaS que foram alvo deste experimento, em ambos os cenários, foram configurados para serem implantados nas respectivas regiões do leste americano de cada provedor, a saber: AWS Lambda, AZF, GCF e AFC.

Ao final da execução das repetições de ambos os cenários de teste, foram criados projetos fatoriais e testes *t* pareados para analisar os efeitos da diferença entre os provedores e a diferença entre o menor nível de concorrência (apenas uma requisição) e o nível mais alto, com 2048 requisições concorrentes. A análise dos resultados obtidos neste experimento será abordada na próxima seção.

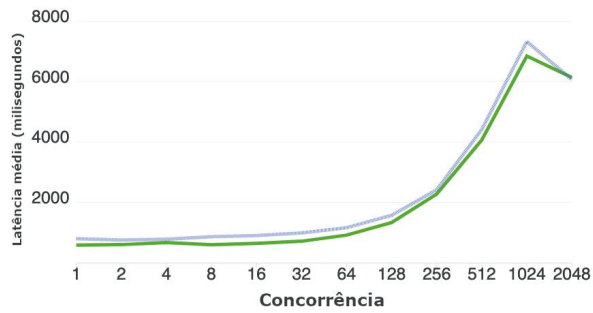
6.3.1 Análise dos Resultados do Experimento 3

Os resultados apresentados nesta seção, incluindo as imagens, foram publicados em [151] (Anexo XIX). A Figura 6.13 apresenta os resultados de latência e taxa de falhas para os quatro provedores analisados. No caso do AWS Lambda (Figuras 6.13a e 6.13b), observa-se que a latência cresce progressivamente com o aumento da concorrência, apresentando uma leve redução nos níveis mais elevados, o que sugere a atuação de mecanismos de escalonamento automático. Entretanto, esse comportamento é acompanhado pelo surgimento de falhas a partir de níveis intermediários de concorrência, indicando limitações na sustentação de cargas elevadas.

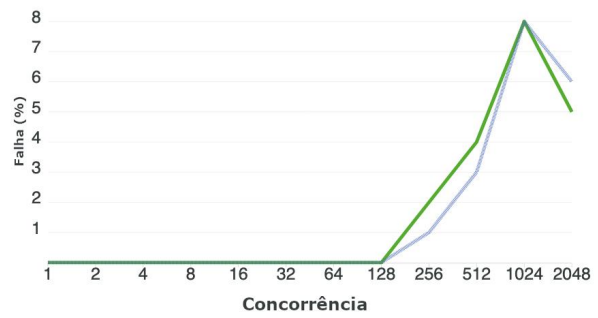
Resultados semelhantes são observados para o provedor GCP (Figuras 6.13c e 6.13d), cujo comportamento de latência acompanha o crescimento da concorrência até determinado ponto, seguido por uma leve redução, também associada à adaptação da infraestrutura. No entanto, diferentemente do AWS, as falhas surgem apenas em níveis mais elevados de concorrência, embora com maior intensidade, sugerindo diferenças na estratégia de gerenciamento de carga entre os provedores.

No caso do Azure Functions (Figuras 6.13e e 6.13f), observa-se um comportamento distinto, caracterizado por crescimento contínuo da latência sem evidências claras de adaptação da infraestrutura ao aumento da demanda. Apesar disso, as taxas de falha permanecem relativamente baixas ao longo dos testes, indicando uma estratégia mais conservadora, com priorização da estabilidade em detrimento do desempenho.

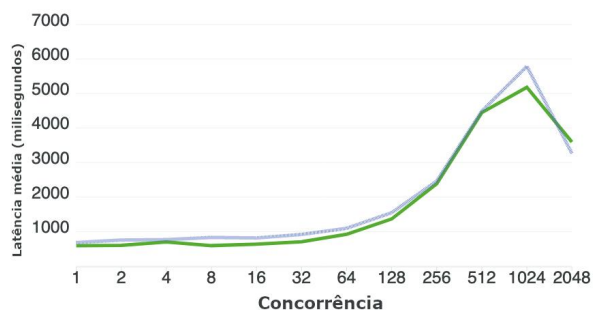
Por outro lado, o provedor Alibaba Cloud Functions (Figura 6.13g) apresentou os menores tempos médios de execução entre todos os cenários analisados, além de não



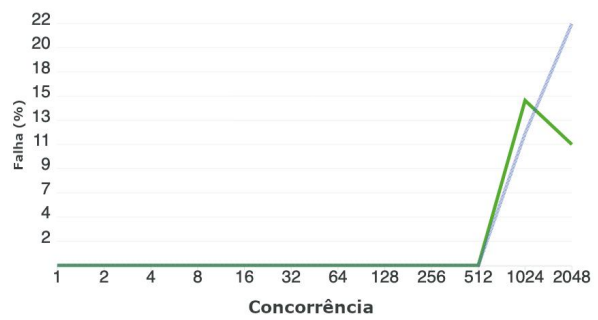
(a) AWS Lambda - Latência.



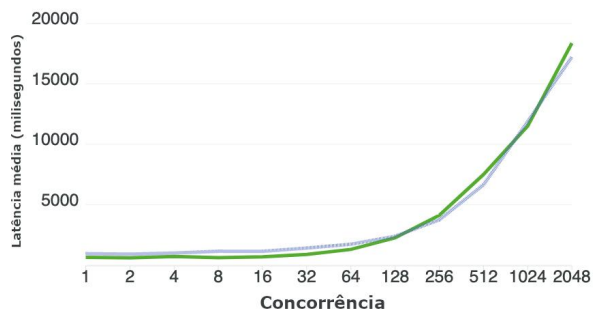
(b) AWS Lambda - Falhas.



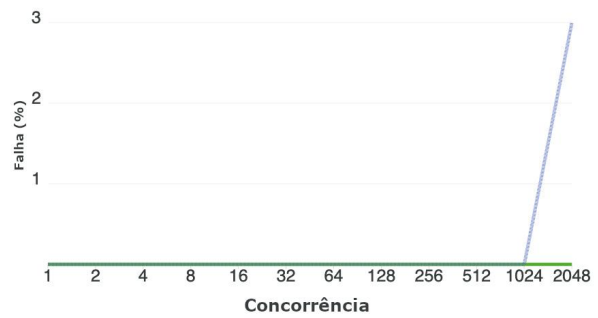
(c) GCP - Latência.



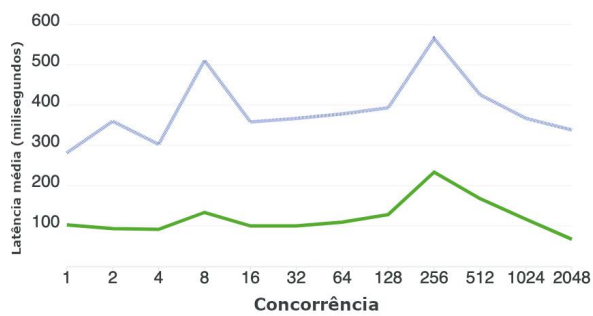
(d) GCP - Falhas.



(e) AZF - Latência.



(f) AZF - Falhas.



(g) AFC - Latência.

● Workers concentrados ● Workers distribuídos

Figura 6.13: Experimento 3 - resultados do caso de uso Calculadora.

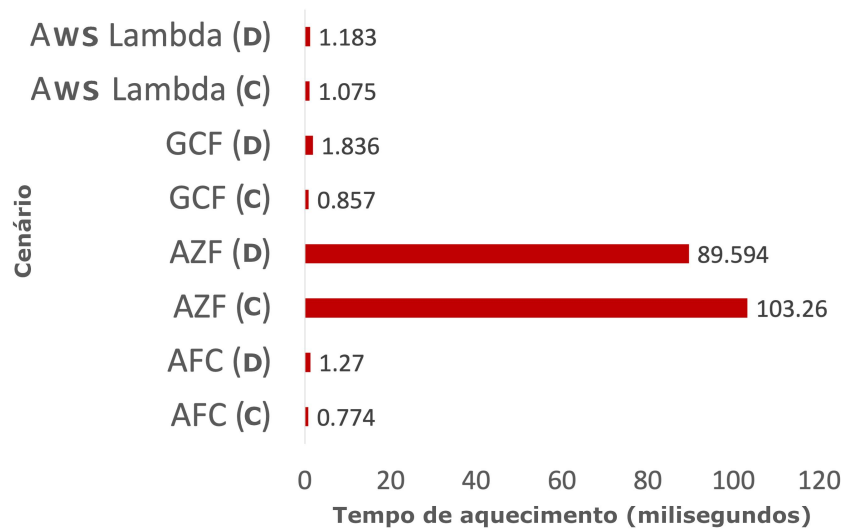


Figura 6.14: Experimento 3 - requisições de *warm-up*.

registrar falhas durante os experimentos. Esse comportamento sugere uma estratégia mais eficiente de escalonamento, capaz de manter desempenho consistente mesmo sob altas cargas.

De forma geral, observa-se que os cenários com *workers* distribuídos apresentaram latências superiores aos cenários concentrados, o que era esperado devido ao aumento da latência de rede. No entanto, o comportamento global das curvas se manteve semelhante entre os dois cenários, indicando que a arquitetura distribuída do Orama preserva a consistência dos resultados, mesmo quando executada em ambientes geograficamente dispersos.

Outro aspecto relevante analisado neste experimento se refere ao fenômeno de *cold start*. A Figura 6.14 apresenta os tempos de *warm-up* para todos os provedores. Observa-se uma diferença significativa no comportamento do Azure Functions, que apresentou tempos de inicialização consideravelmente superiores aos demais provedores. Enquanto AWS, GCP e Alibaba Cloud registraram tempos próximos de um segundo, o Azure apresentou tempos muito mais elevados, o que pode impactar negativamente aplicações sensíveis à latência inicial.

Adicionalmente, a análise por meio de projeto *fatorial 2^k* permitiu identificar a influência relativa dos fatores “provedor” e “nível de concorrência” nos resultados. De forma geral, verificou-se que o fator concorrência exerceu maior impacto em parte das comparações, enquanto o fator provedor se destacou principalmente nos cenários envolvendo o Alibaba Cloud, corroborando seu desempenho diferenciado observado nas métricas de latência e falha.

Os resultados dos testes *t* pareados, apresentados na Tabela 6.4, indicam que todas as

diferenças entre os provedores são estatisticamente significativas com nível de confiança de 99,95%. Esse resultado reforça que as variações observadas não são decorrentes de flutuações aleatórias, mas sim de diferenças reais entre as plataformas avaliadas.

Tabela 6.4: Experimento 3 - resultados do teste t pareado entre provedores.

Comparação	Diferença (ms)	Desvio Padrão	Confiança (%)
AWS x GCP	1.317,03	281,82	99,95
AWS x AZF	7.792,25	928,90	99,95
AWS x AFC	5.454,88	259,53	99,95
GCP x AZF	9.109,28	898,69	99,95
GCP x AFC	4.137,84	110,14	99,95
AZF x AFC	13.247,13	891,95	99,95

De maneira geral, os resultados deste experimento evidenciam que a arquitetura distribuída do Orama⁺⁺ é capaz de conduzir experimentos de larga escala com consistência, mesmo em cenários com distribuição geográfica dos geradores de carga. Além disso, os resultados obtidos permitem identificar diferenças relevantes entre provedores, bem como compreender os impactos da concorrência e da latência de rede no desempenho de aplicações FaaS. Esses achados reforçam o papel do Orama⁺⁺ como uma ferramenta robusta para análise experimental e suporte à tomada de decisão em ambientes de Computação em Nuvem.

6.4 Experimento 4

O foco do Experimento 4 foi entender o comportamento do serviço FaaS de forma isolada, o caso de uso escolhido foi a Calculadora, pois evita a interferência de serviços subjacentes, como armazenamento de objetos e banco de dados. Para isso, foram selecionadas cinco zonas de disponibilidade em cada um dos quatro provedores disponíveis no *framework* Orama⁺⁺, conforme mostrado na Tabela 6.5.

A seleção das regiões foi realizada de forma a permitir uma comparação justa em relação à posição dos serviços e ao local onde o *framework* Orama⁺⁺ foi instalado para a realização dos testes (Brasil). As cinco regiões do mundo onde todos os quatro provedores possuem zonas de disponibilidade são: Leste dos EUA, Oeste dos EUA, Europa, região da Ásia-Pacífico e Austrália, conforme mostrado na Figura 6.15. Embora as zonas de disponibilidade não estejam exatamente na mesma localização, verificou-se que sua proximidade é suficiente para realizar comparações de desempenho entre os provedores.

O *framework* Orama⁺⁺ foi instalado em uma máquina localizada em Brasília (Brasil), equipada com um processador Intel Core i5 de 8ª geração, 16GB de memória RAM,

Tabela 6.5: Experimentos 4 e 5 - regiões dos provedores envolvidos.

Macro região	Micro região	Provedor	Região
USA (<i>West</i>)	Virginia	AWS	us-west-1
	Virginia	Azure	westus
	Virginia	GCP	us-west2
	Virginia	Alibaba	us-west-1
USA (<i>East</i>)	California	AWS	us-east-1
	California	Azure	eastus
	California	GCP	us-east4
	California	Alibaba	us-east-1
<i>Europe</i>	Frankfurt	AWS	eu-central-1
	France	Azure	francecentral
	Frankfurt	GCP	europe-west3
	Frankfurt	Alibaba	eu-central-1
Asia	Hong Kong	AWS	ap-east-1
	Hong Kong	Azure	eastasia
	Hong Kong	GCP	asia-east2
	Hong Kong	Alibaba	cn-hongkong
Austrália	Sydney	AWS	ap-southeast-2
	Victoria	Azure	australiasoutheast
	Sydney	GCP	australia-southeast1
	Sydney	Alibaba	ap-southeast-2

sistema operacional Linux Mint 20.04, e uma conexão de Internet compartilhada de aproximadamente 500 Mbps. Foram definidos níveis de concorrência com crescimento exponencial, variando de 1 até 512 requisições simultâneas, ou seja, 1, 2, 4, 8, 16, 32, 64, 128, 256 e 512 ativações simultâneas em cada bateria de testes.

As baterias foram repetidas 10 vezes para formar uma média e permitir a análise estatística. Ao final das baterias de testes foram utilizadas as funcionalidades do próprio *framework* Orama⁺⁺ para comparação direta entre os *benchmarks*, e análise estatística com projeto *fatorial 2^k* e teste *t* pareado. O projeto *fatorial 2^k* envolveu dois provedores e dois níveis de concorrência (o menor e o maior), permitindo analisar o impacto de ambos os fatores (provedor e concorrência) nos resultados. No teste *t* pareado, foi analisada a diferença entre o tempo médio de execução para determinar se a diferença é estatisticamente relevante. A partir dessas baterias de testes, foi possível visualizar os resultados, conforme analisado na próxima seção.

6.4.1 Análise dos Resultados do Experimento 4

Os resultados apresentados nesta seção, incluindo as imagens, foram publicados em [152] (Anexo XX). Inicialmente, a Figura 6.16 apresenta os tempos de *warm-up*, que correspondem à primeira requisição realizada antes da execução das baterias de testes e permitem analisar o *cold start*. Observa-se que o provedor AZF apresentou tempos significativamente superiores aos demais, indicando uma estratégia de inicialização mais custosa. Em

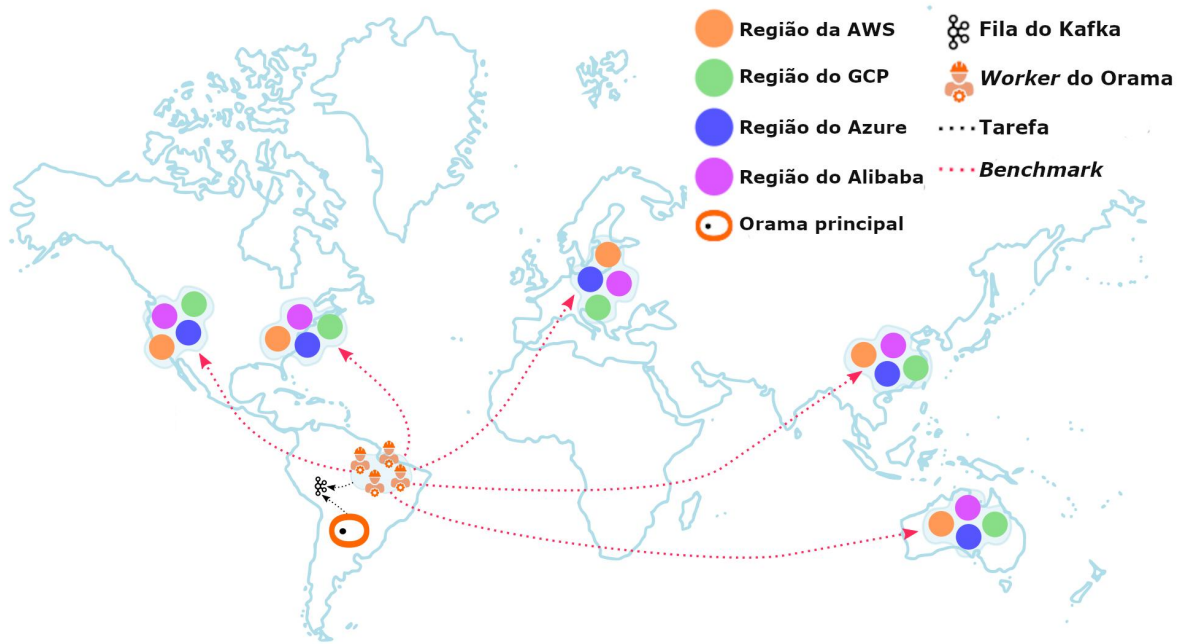


Figura 6.15: Experimentos 4 e 5 - arquitetura dos testes.

contraste, AWS Lambda, GCF e AFC apresentaram tempos de inicialização mais homogêneos e substancialmente inferiores. Esse comportamento evidencia que aplicações sensíveis à latência inicial podem ser impactadas negativamente quando implantadas em ambientes com alto custo de *cold start*, especialmente, em cenários com períodos de inatividade.

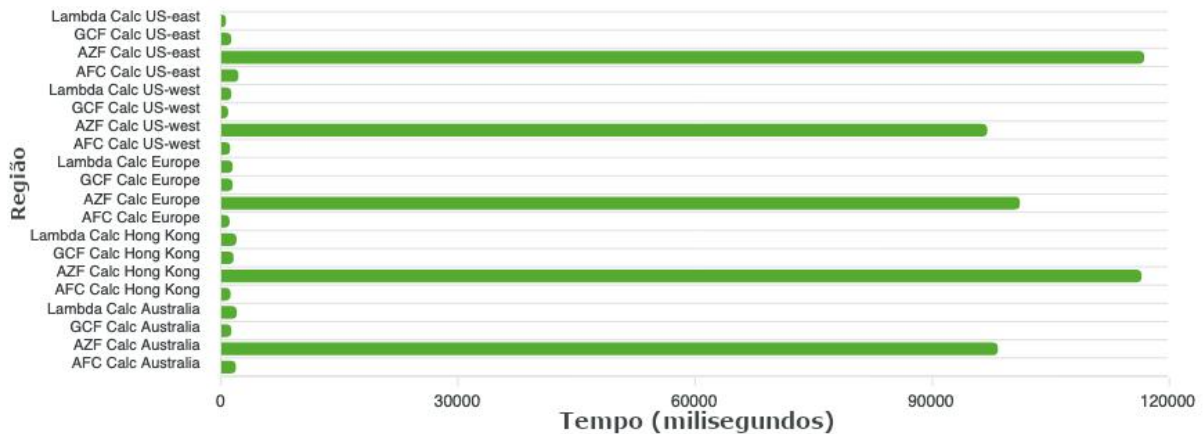
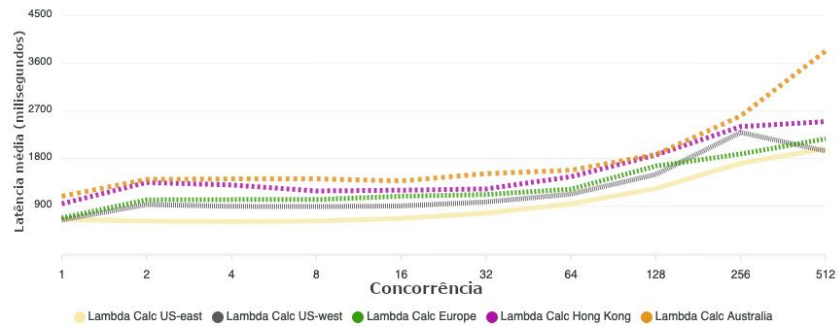
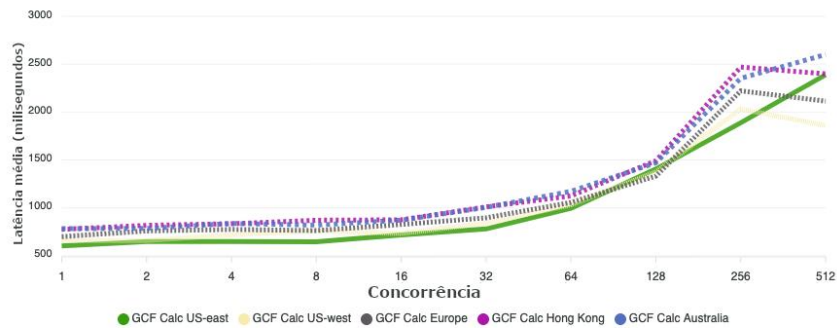


Figura 6.16: Experimento 4 - resultados das requisições de *warm-up*.

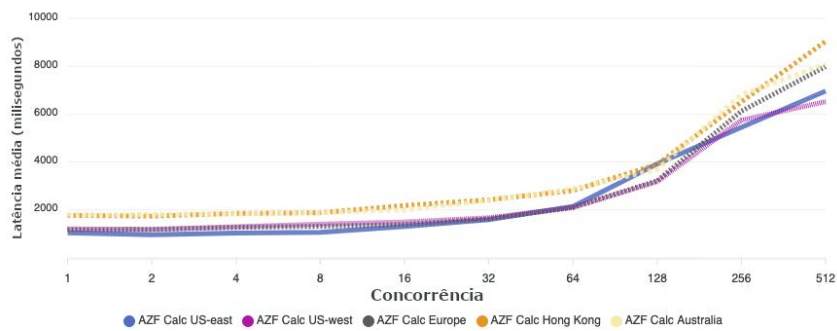
A análise dos resultados agrupados por provedor é apresentada na Figura 6.17. No caso do AWS Lambda (Figura 6.17a), observa-se que, embora existam diferenças nos níveis absolutos de latência entre regiões, o comportamento geral permanece consistente. Em todos os casos, o aumento do nível de concorrência resulta no crescimento do tempo



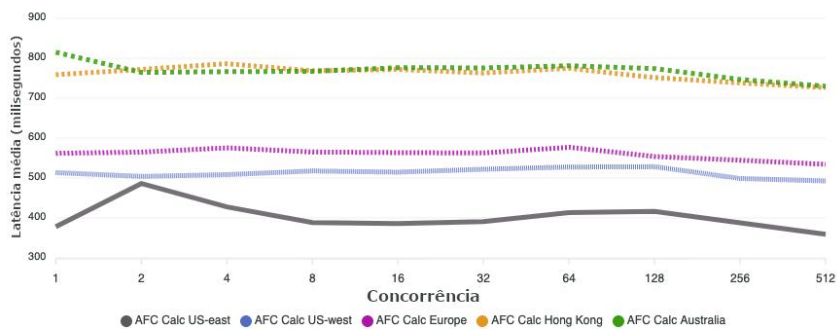
(a) AWS Lambda.



(b) GCF.



(c) AZF.



(d) AFC.

Figura 6.17: Experimento 4 - resultados do caso de uso Calculadora em cinco regiões dos provedores.

médio de execução, com indícios pontuais de redução em níveis mais elevados, sugerindo possíveis mecanismos de escalabilidade dinâmica e reforço de infraestrutura.

Resultados semelhantes são observados para o GCF (Figura 6.17b), cujo comportamento também apresenta crescimento da latência com o aumento da concorrência, seguido por reduções em determinados pontos. Esse padrão reforça a hipótese de que provedores adotam estratégias reativas para adaptação à carga, promovendo ajustes na infraestrutura conforme a demanda aumenta. Ainda assim, diferenças entre regiões indicam que tais mecanismos não são uniformemente aplicados ou possuem níveis distintos de maturidade.

Por outro lado, os resultados do AZF (Figura 6.17c) evidenciam um comportamento distinto. Em todas as regiões analisadas, verifica-se um crescimento contínuo do tempo médio de execução à medida que a concorrência aumenta, sem indícios claros de estabilização ou redução. Esse padrão sugere ausência ou menor efetividade de mecanismos de escalabilidade dinâmica capazes de mitigar o impacto do aumento de carga, o que corrobora os resultados observados em experimentos anteriores.

Em contraste, o AFC (Figura 6.17d) apresenta um comportamento significativamente diferente dos demais provedores. Os tempos médios de execução mantêm-se relativamente estáveis mesmo com o aumento da concorrência, com pequenas variações e eventuais reduções. Esse comportamento indica a adoção de estratégias mais robustas de provisionamento ou gerenciamento de recursos, capazes de absorver variações de carga sem impacto significativo no desempenho. Além disso, tal estabilidade sugere a existência de mecanismos proativos ou de superdimensionamento de recursos. Como os testes foram realizados usando regimes de contratação semelhantes com cada provedor, não houve qualquer benefício específico disponibilizado pelo provedor que pudesse justificar um superdimensionamento da infraestrutura. Dessa forma, é provável que o provedor adote essa abordagem a fim de se destacar no mercado por meio de um desempenho superior.

Ao se analisar os resultados sob a perspectiva das regiões, observa-se que, independentemente da localização geográfica, o padrão relativo entre provedores se mantém. O AFC apresenta os menores tempos médios de execução, seguido por AWS Lambda e GCF com desempenhos semelhantes, enquanto o AZF apresenta os maiores valores. Essa consistência indica que as diferenças entre provedores são mais determinantes para o desempenho do que as variações regionais, embora estas também influenciem os valores absolutos observados.

Adicionalmente, foram aplicadas técnicas de análise estatística, incluindo projeto *fatorial* 2^k e testes *t* pareados, com o objetivo de avaliar a significância das diferenças observadas. De forma geral, os resultados indicaram que o fator concorrência exerce forte influência sobre o desempenho, especialmente nas comparações entre AWS Lambda e GCF. Em diversos cenários, o fator provedor apresentou menor impacto relativo, refor-

gando a predominância da carga de trabalho como elemento determinante. Ainda assim, os testes t apontaram que, mesmo quando próximas, as diferenças entre os provedores são estatisticamente relevantes, embora com níveis de confiança variáveis entre as regiões.

Em síntese, os resultados deste experimento evidenciam que, embora exista variabilidade entre regiões, o comportamento estrutural dos provedores permanece consistente. Além disso, destacam-se diferenças significativas nas estratégias de gerenciamento de recursos adotadas por cada plataforma, o que impacta diretamente o desempenho observado. Esses achados reforçam a importância de considerar tanto o provedor quanto o perfil de carga, e a sensibilidade à latência na tomada de decisão sobre o provisionamento de aplicações em ambientes FaaS.

6.5 Experimento 5

O Experimento 5 compartilha a mesma arquitetura usada no Experimento 4, porém usando outro caso de uso. As soluções que envolvem o uso de FaaS em colaboração com serviços de banco de dados são escolhas comuns para resolver problemas do mundo real. Assim, neste experimento foi utilizado um tipo de caso de uso que provisiona os serviços FaaS integrados a soluções de DBaaS, como DynamoDB na AWS, Firestore no GCP, CosmosDB no Azure e TableStore na Alibaba Cloud. Assim como no Experimento 4, neste experimento foram selecionadas cinco zonas de disponibilidade em cada um dos quatro provedores disponíveis no *framework* Orama⁺⁺, conforme mostrado na Tabela 6.5.

Com o propósito de submeter os ambientes FaaS a diferentes níveis de concorrência, foram definidos 13 cenários de teste com graus de concorrência iniciando em 1 e chegando a 4096, com crescimento exponencial, ou seja, cenários com 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 e 4096 requisições concorrentes ao FaaS. Cada bateria de testes foi configurada no *framework* Orama⁺⁺ para ser repetida 10 vezes, a fim de construir uma média dos tempos de execução.

Considerando os níveis mais altos de concorrência (2048 e 4096), decidiu-se implementar o *framework* Orama⁺⁺ no modo *master-worker*, pois no nível mais alto, cada *worker* seria responsável por acionar o FaaS com 1024 requisições concorrentes. Portanto, o *framework* Orama⁺⁺ foi implantado em uma instância do GCP *Compute Engine* na região de São Paulo/Brasil. O nó *master* possuía 4 vCPUs e 16GB de RAM (e2-standard-4), enquanto cada um dos 4 *workers* tinha 2 vCPUs e 4GB de RAM (e2-medium). Todas as instâncias utilizavam o sistema operacional Debian 11.

Conforme pode ser visto na Figura 6.15, uma vez que o *framework* Orama⁺⁺ foi implantado em uma região da América do Sul, as requisições para acionar o FaaS partiram de lá e viajaram pela rede até alcançar as respectivas regiões onde os serviços-alvo estavam

alocados. Era esperado que houvesse diferença de latência entre as regiões, já que suas posições não são idênticas. No entanto, como o foco deste trabalho é analisar as diferenças nas implementações entre regiões, considerou-se suficiente apenas alocar regiões próximas entre os provedores para mitigar os impactos da diferença de latência.

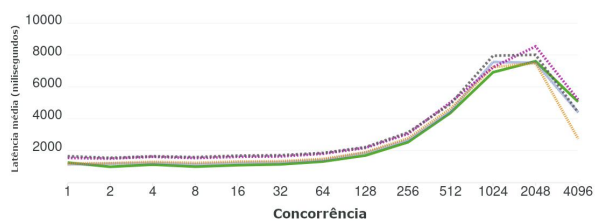
Uma vez implementado o *framework*, foram solicitadas as implantações dos respectivos ambientes do caso de uso de FaaS com DBaaS para cada região de cada provedor, totalizando 20 provisionamentos. Orama⁺⁺ foi configurado para realizar uma requisição de aquecimento aos serviços e separá-la das demais requisições, permitindo uma análise de *cold start*. Cada ambiente provisionado foi submetido a repetições das baterias de teste para que os resultados pudessem ser estatisticamente relevantes, conforme analisado na próxima seção.

6.5.1 Análise dos Resultados do Experimento 5

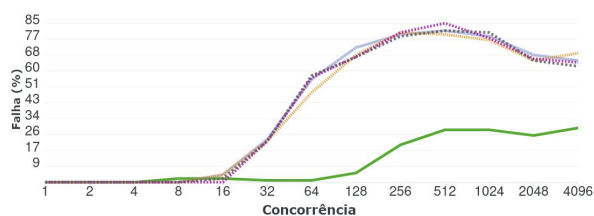
Os resultados apresentados nesta seção, incluindo as imagens, foram publicados em [153] (Anexo XXII). A Figura 6.18 apresenta os tempos médios de execução e as taxas de falha para os quatro provedores avaliados. De forma geral, observa-se que os serviços AWS Lambda (Figura 6.18a) e GCF (Figura 6.18c) apresentam comportamentos semelhantes, com crescimento progressivo do tempo médio de execução à medida que o nível de concorrência aumenta. Em ambos os casos há indícios de redução nos tempos médios em níveis mais elevados de concorrência, o que sugere a atuação de mecanismos de elasticidade automática característicos do modelo FaaS. Além disso, as diferentes regiões de cada provedor apresentam curvas com comportamento equivalente, diferindo principalmente nos valores absolutos, influenciados por fatores como latência de rede.

Por outro lado, o comportamento observado no AZF (Figura 6.18e) difere significativamente dos demais provedores. Os tempos médios de execução apresentam valores substancialmente mais elevados, atingindo ordens de grandeza superiores, além de exibirem crescimento contínuo com o aumento da concorrência. Não há evidências claras de estabilização ou redução dos tempos, o que indica limitações na adaptação dinâmica da infraestrutura frente a aumentos de carga. Adicionalmente, observa-se uma alta variabilidade nos resultados, sugerindo sensibilidade a efeitos cumulativos decorrentes da execução sequencial das cargas de teste.

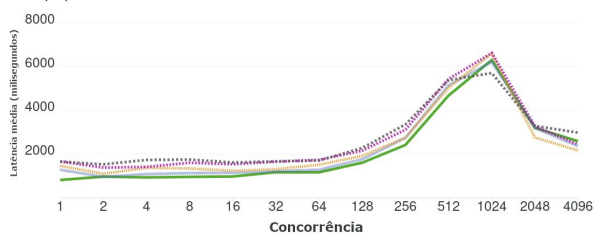
Em contraste, o AFC (Figura 6.18g) apresenta os menores tempos médios de execução entre os provedores analisados, além de um comportamento consideravelmente mais estável. Mesmo em níveis elevados de concorrência, a variação dos tempos médios é limitada, o que indica a adoção de estratégias eficientes de gerenciamento de recursos. Embora existam diferenças entre regiões, estas são relativamente pequenas e podem ser atribuídas principalmente à latência.



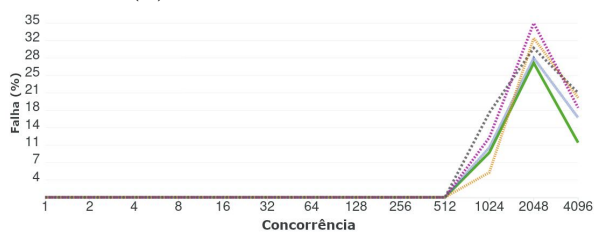
(a) AWS Lambda - Tempos de Execução.



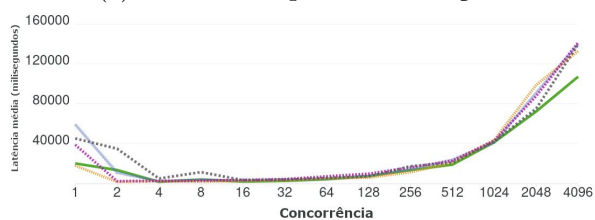
(b) AWS Lambda - Falhas.



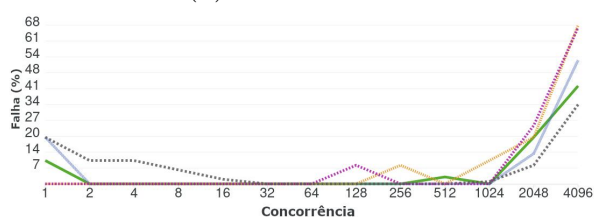
(c) GCF - Tempos de Execução.



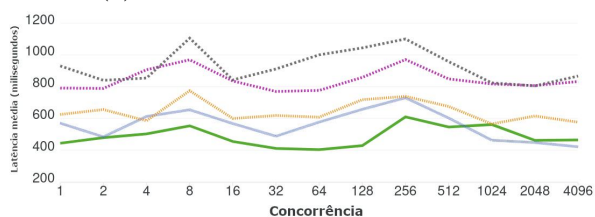
(d) GCF - Falhas.



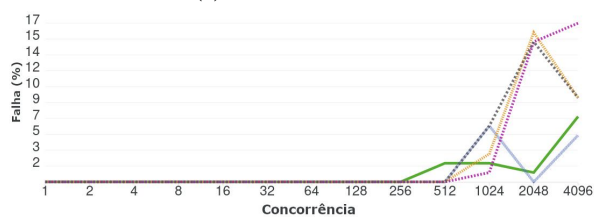
(e) AZF - Tempos de Execução.



(f) AZF - Falhas.



(g) AFC - Tempos de Execução.



(h) AFC - Falhas.

● US-East ● US-West ● Europe ● Asia ● Oceania

Figura 6.18: Experimento 5 - resultados do caso de uso API com DBaaS em cinco regiões dos provedores.

Em relação às taxas de falha, observa-se que o AWS Lambda (Figura 6.18b) apresenta falhas a partir de níveis intermediários de concorrência, com crescimento progressivo até níveis mais elevados, seguido por pequenas reduções associadas a possíveis intervenções na infraestrutura. O GCF (Figura 6.18d) apresenta início tardio das falhas, porém com crescimento mais acentuado em níveis elevados de concorrência. Já o AZF (Figura 6.18f) apresenta falhas em praticamente todos os níveis de concorrência, com taxas elevadas que acompanham o aumento da carga, reforçando as limitações observadas nos tempos de execução.

Por sua vez, o AFC (Figura 6.18h) apresenta as menores taxas de falha entre os provedores analisados, mesmo sob altas cargas. Ainda que ocorram falhas em níveis mais elevados de concorrência, estas se mantêm em patamares significativamente inferiores aos demais provedores, indicando maior robustez na execução das requisições.

A análise sob a perspectiva das regiões evidencia que, embora existam variações nos valores absolutos de desempenho, o comportamento relativo entre os provedores se mantém consistente. Em todas as regiões, o AFC apresenta os melhores resultados, seguido por AWS Lambda e GCF com desempenhos intermediários e semelhantes, enquanto o AZF apresenta os piores resultados. Esse padrão reforça que as diferenças entre provedores são mais determinantes do que as variações regionais.

Adicionalmente, foram empregadas técnicas de análise estatística, incluindo projeto *fatorial* 2^k e testes *t* pareados, com o objetivo de avaliar a significância das diferenças observadas. De forma geral, os resultados indicaram que o fator concorrência exerce influência predominante sobre o desempenho, especialmente nas comparações entre AWS Lambda e GCF, cujos comportamentos são bastante próximos. Também foram identificados indícios da influência de fatores adicionais, tal como latência de rede, especialmente, em cenários multi-região.

Os testes *t*, por sua vez, indicaram que as diferenças entre os provedores são estatisticamente relevantes, embora com níveis de confiança distintos entre as regiões analisadas. Esse resultado reforça que, mesmo quando os desempenhos são próximos, como no caso de AWS Lambda e GCF, as diferenças observadas não são aleatórias e devem ser consideradas no processo de tomada de decisão.

Em síntese, os resultados deste experimento evidenciam que a integração com serviços externos, como DBaaS, acentua as diferenças entre os provedores, especialmente em termos de latência e robustez sob carga. Além disso, destacam a importância de considerar tanto o comportamento multi-região quanto a natureza da aplicação ao selecionar o provedor mais adequado em ambientes FaaS.

Os experimentos descritos neste capítulo auxiliaram tanto na compreensão do comportamento dos provedores que participam de casos de uso implantados automaticamente

pelo *framework* Orama⁺⁺, quanto na validação das próprias funcionalidades do *framework*, especialmente, aquelas relacionadas a *benchmarking* e análise estatística. Entretanto, em um processo decisório para implantação de ambientes FaaS é muito dispendioso o esforço para implantação de ambientes e execução de *benchmarks*. Uma vez que os referidos experimentos geraram informações valiosas quanto ao comportamento dos provedores em diferentes cenários, esses dados puderam ser consolidados e utilizados no treinamento do modelo de Redes Neurais. Isso permitiu a incorporação, na proposta, de uma ferramenta que permite ao usuário escrever uma função diretamente na plataforma e visualizar estimativas de tempo de execução em diferentes provedores, assim como estimativas de custo, sem a necessidade de implantar nenhum ambiente. Essa solução torna o processo decisório sobre em qual provedor implantar uma determinada função mais objetivo, fundamentado e barato. Na próxima seção, será apresentada uma avaliação do modelo de Redes Neurais que foi treinado usando os dados dos Experimentos de 1 a 5.

6.6 Avaliação do Modelo de Redes Neurais Treinado Usando Informações dos Experimentos de 1 a 5

Para avaliar a qualidade do modelo foi utilizada uma análise gráfica a partir das principais métricas relacionadas ao processo de predição, tais como RMSE, MSE, MAPE e R^2 . As análises a seguir se referem ao último cálculo dessas métricas, que foram recalculadas e analisadas a cada execução de uma nova rodada de treinamento.

O *Boxplot* do RMSE, apresentado na Figura 6.19, mostra que a maioria dos erros está entre 0,1 e 0,25, com mediana próxima de 0,15. No entanto, há muitos *outliers* acima de 0,4, indicando que o modelo tem dificuldades com certos casos, resultando em erros elevados. A distribuição é assimétrica positiva, sugerindo que, embora o desempenho seja estável na maioria dos casos, alguns apresentam desvios significativos.

O *Boxplot* do MSE, apresentado na Figura 6.20, mostra uma distribuição altamente assimétrica, com mediana próxima de zero e quartis bem concentrados, indicando que a maioria dos erros é baixa. No entanto, a presença de muitos *outliers* na região superior sugere que, embora o modelo tenha um bom desempenho na maioria dos casos, ele ocasionalmente gera erros muito altos, possivelmente devido a *overfitting* ou dados ruidosos.

O *Boxplot* do MAPE, apresentado na Figura 6.21, indica uma distribuição concentrada, com mediana próxima de 2%, e 50% dos erros variando entre 1,5% e 3%, sem *outliers* visíveis. Os valores mínimo e máximo ficam entre 1% e 5%, sugerindo baixa dispersão e leve assimetria para valores mais altos. No geral, o modelo apresenta boa

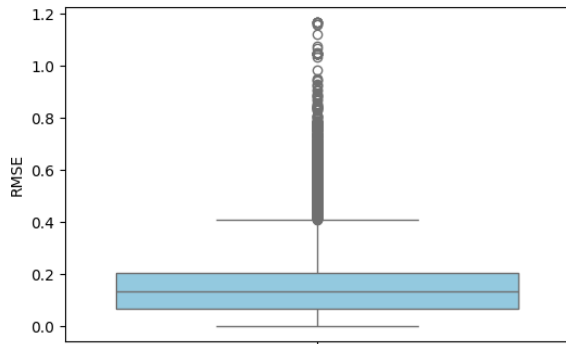


Figura 6.19: *Boxplot* do RMSE.

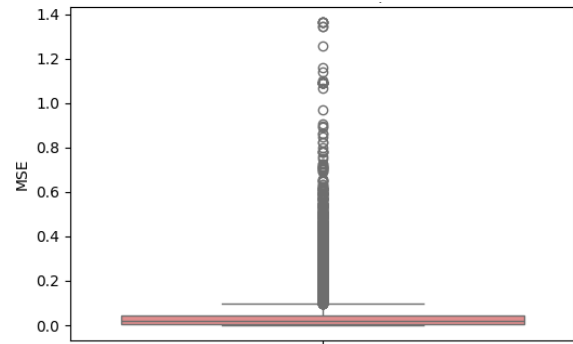


Figura 6.20: *Boxplot* do MSE.

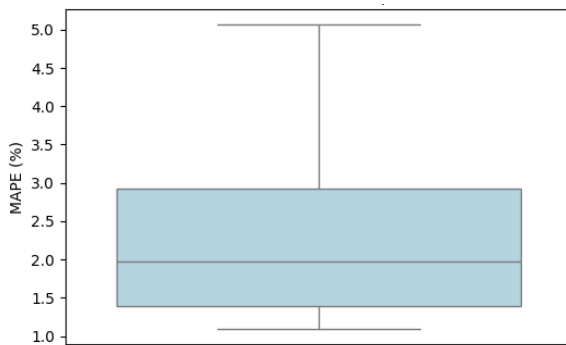


Figura 6.21: *Boxplot* do MAPE.

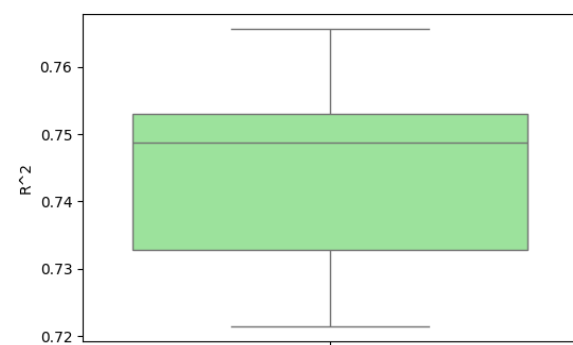


Figura 6.22: *Boxplot* do R^2 .

precisão, com a maioria dos erros abaixo de 3%, o que é positivo para aplicações que exigem previsões estáveis.

O *Boxplot* do R^2 , apresentado na Figura 6.22, indica uma distribuição estável entre aproximadamente 0,72 e 0,76, com mediana em torno de 0,75. A baixa variabilidade sugere um desempenho consistente do modelo, sem *outliers* que apontem falhas significativas. Como os valores estão relativamente altos, o modelo demonstra boa capacidade preditiva, explicando bem a variabilidade dos dados. Outras análises gráficas adotadas na qualificação do modelo foram a avaliação da função de perda no decorrer do treinamento, e o gráfico do comparativo entre observações e previsões do lote de validação do *dataset*.

O gráfico apresentado na Figura 6.23 mostra a evolução da função de perda ao longo de 50 épocas para os conjuntos de treinamento e validação. Inicialmente, ambas as perdas começam elevadas e rapidamente diminuem nas primeiras épocas, indicando que o modelo aprende rapidamente no início. Após essa queda inicial, a perda de treinamento estabiliza em torno de 0,05, com uma tendência suave e sem grandes oscilações. Já a perda de validação apresenta flutuações consideráveis, sugerindo maior variabilidade.

O gráfico mostrado na Figura 6.24 compara as observações reais (azul) com as previsões do modelo (laranja) ao longo de diferentes execuções. Em geral, há uma correlação entre as séries, pois a linha de predição segue o padrão das observações, mas há momentos

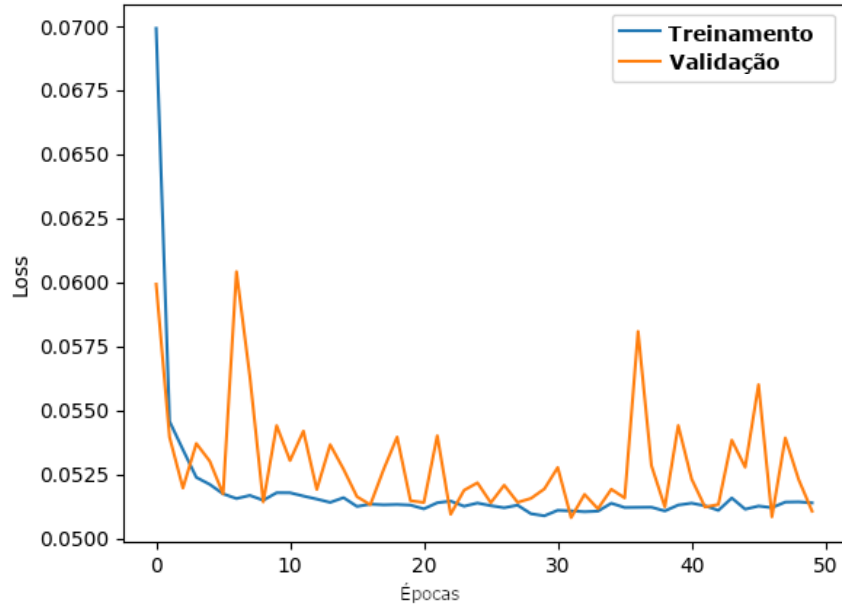


Figura 6.23: Função perda (*Loss*) no decorrer do treinamento.

em que ocorrem desvios. Os picos e vales das predições nem sempre coincidem perfeitamente com os das observações, sugerindo que o modelo captura tendências gerais, mas pode ter dificuldades em prever variações extremas ou oscilações abruptas. Além disso, em algumas regiões, o modelo subestima ou superestima valores, o que pode indicar limitações na capacidade preditiva ou ruídos nos dados.

Como já foi mencionado, todas essas métricas citadas nesta seção foram recalculadas a cada rodada do processo de treinamento e reanalisadas. Para auxiliar na definição de novos valores para os hiperparâmetros foram utilizadas técnicas de otimização, descritas na Seção 4.7.2.3. Diante dos resultados apresentados nesta seção, o modelo foi considerado apto para ser incorporado ao componente *Predictor* do *framework* Orama⁺⁺, atuando na estimativa de tempo de execução de funções FaaS em diferentes provedores.

Apesar dos resultados satisfatórios obtidos a partir dos dados dos Experimentos 1 a 5, observou-se que o modelo ainda apresentava limitações na captura de comportamentos mais complexos, especialmente aqueles influenciados por características dinâmicas de entrada e cenários de execução mais realistas. Nesse contexto, tornou-se necessária a ampliação do *dataset* com dados provenientes de aplicações mais representativas e com maior variabilidade computacional. Assim, a próxima seção apresenta o Experimento 6, no qual é introduzido um caso de uso baseado em alinhamento de sequências genéticas, permitindo incorporar ao processo de treinamento informações mais ricas relacionadas à escalabilidade da entrada e aos efeitos de concorrência, com o objetivo de aprimorar a capacidade preditiva do modelo.

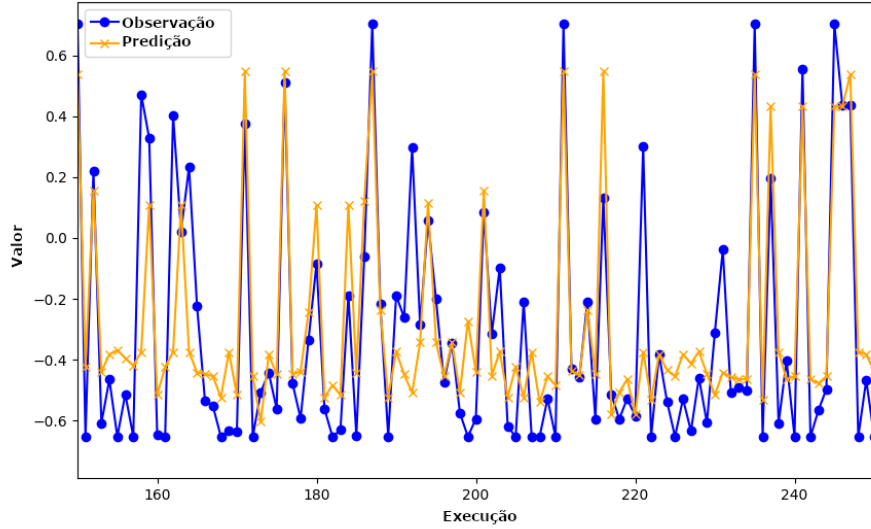


Figura 6.24: Comparativo entre observações e previsões.

6.7 Experimento 6

O Experimento 6 foi concebido com o objetivo de ampliar o *dataset* utilizado no treinamento do modelo preditivo de tempo de execução do *framework* Orama⁺⁺, incorporando dados provenientes de um caso de uso mais complexo e sensível às características de entrada. Diferentemente dos experimentos anteriores, que exploram predominantemente métricas extraídas do código-fonte, este experimento introduz variabilidade associada ao tamanho da entrada e ao nível de concorrência, permitindo capturar comportamentos mais próximos de cenários reais de execução em ambientes FaaS.

O experimento foi estruturado a partir de um caso de uso de alinhamento de sequências genéticas, previamente descrito na Seção 4.5.4, no qual duas funções equivalentes foram implementadas e implantadas em diferentes provedores de nuvem: AWS Lambda e GCF. Ambas as funções foram configuradas com 2 GB de memória e ambiente de execução Python 3.11, sendo alocadas, respectivamente, em regiões distintas dos provedores. Antes da coleta efetiva dos dados, foi realizada uma fase de aquecimento (*warm-up*) para reduzir o impacto de *cold starts* e garantir maior estabilidade nas medições.

A modelagem experimental foi definida com base na combinação de dois fatores principais: escala de entrada e nível de concorrência. Para representar diferentes níveis de complexidade computacional, foram definidos três perfis de entrada: baixo, médio e alto. Cada perfil é composto por pares de sequências biológicas previamente selecionadas, cujos tamanhos crescentes impactam diretamente o tempo de execução da função. A Tabela 6.6 apresenta as sequências utilizadas em cada escala.

Para cada escala de entrada foram definidos seis níveis de concorrência: 1, 2, 4, 8, 16 e 32 invocações paralelas. Essa variação permite observar o comportamento da função sob

Tabela 6.6: Sequências genéticas utilizadas no Experimento 6, categorizadas por escala de entrada.

Escala de Entrada	ID da Sequência	Tamanho (aa)
Baixa	NC_045512.2 [154]	117
	YP_009725255 [155]	38
Média	NC_021546 [156]	966
	NM_001142364 [157]	1136
Alta	OK084631 [158]	4114
	OK084572 [158]	4123

diferentes condições de carga, refletindo cenários típicos de elasticidade em plataformas *serverless*. Cada combinação de escala de entrada e nível de concorrência foi executada de forma independente em ambos os provedores, totalizando seis configurações de *benchmark* por provedor.

A fim de garantir a robustez estatística dos dados coletados, cada configuração experimental foi repetida 30 vezes. Ao final, foi obtido um conjunto abrangente de medições de tempo de execução, contemplando tanto a variabilidade introduzida pelo tamanho da entrada quanto os efeitos decorrentes da concorrência e da infraestrutura subjacente dos provedores. Os dados gerados neste experimento foram posteriormente integrados ao *dataset* original do Orama⁺⁺, sendo utilizados no processo de re-treinamento do modelo de Redes Neurais responsável pela predição de tempo de execução. Dessa forma, o modelo passa a considerar não apenas características estáticas do código-fonte, mas também aspectos dinâmicos relacionados à entrada, ampliando sua capacidade de generalização e precisão em cenários mais complexos.

6.8 Avaliação do Modelo de Redes Neurais Retreinado Após o Experimento 6

Esta seção apresenta a avaliação do modelo preditivo do Orama⁺⁺ após o processo de retreino utilizando o caso de uso de alinhamento de sequências genéticas. A análise contemplou o comportamento de convergência dos modelos durante o treinamento, bem como a qualidade das predições obtidas, a partir de métricas clássicas de regressão e da comparação direta entre valores observados e estimados.

A Figura 6.25 apresenta as curvas da função de perda (*loss*) ao longo das épocas de treinamento para as arquiteturas Dense, LSTM e BLSTM. De forma geral, observa-se que todos os modelos apresentam uma redução acentuada da perda nas épocas iniciais,

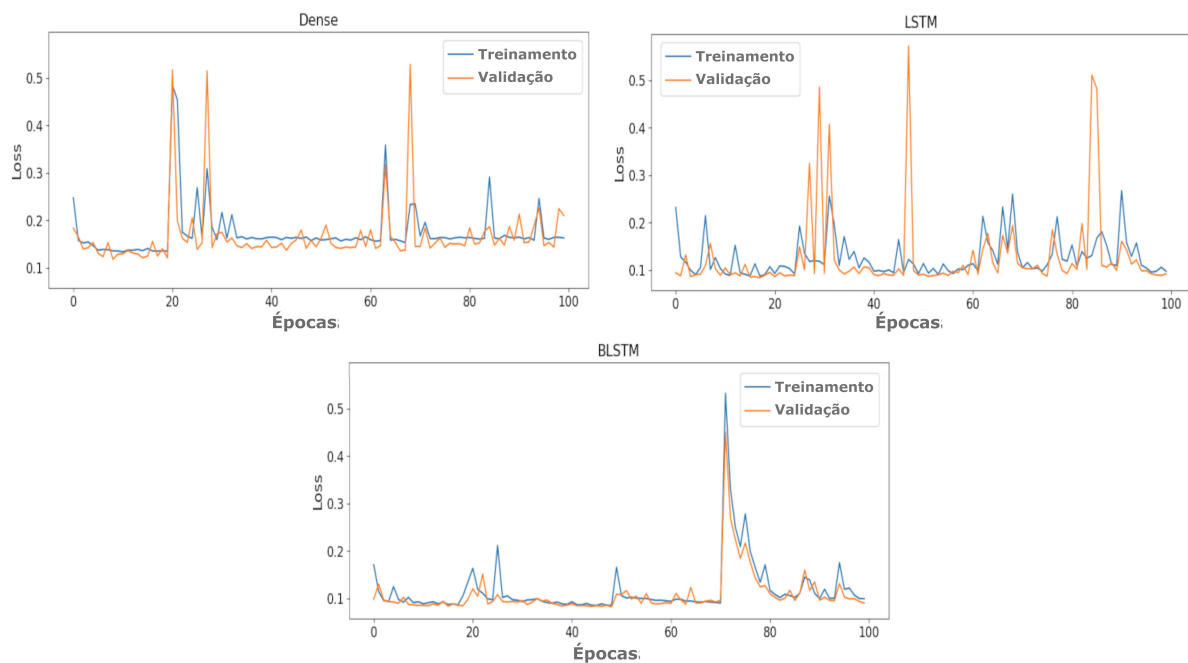


Figura 6.25: Resultados da métrica *Loss* nos modelos Dense, LSTM e BLSTM.

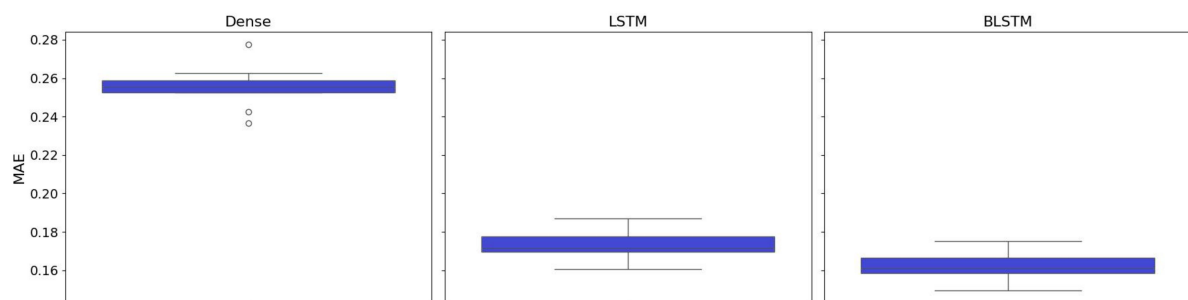


Figura 6.26: Resultados da métrica MAE nos modelos Dense, LSTM e BLSTM.

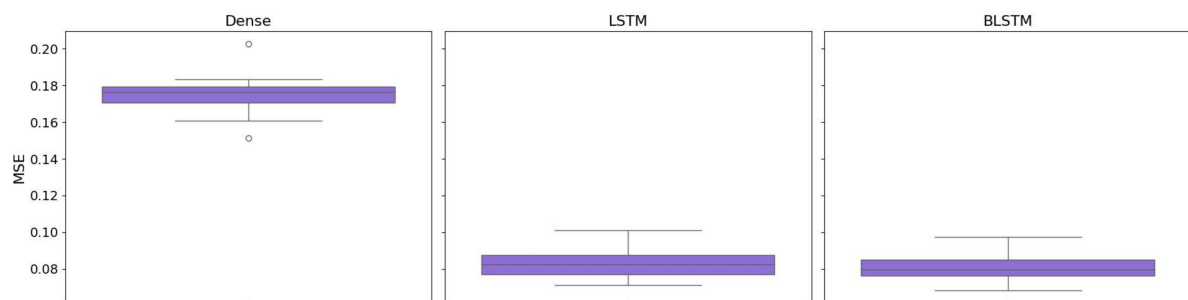


Figura 6.27: Resultados da métrica MSE nos modelos Dense, LSTM e BLSTM.

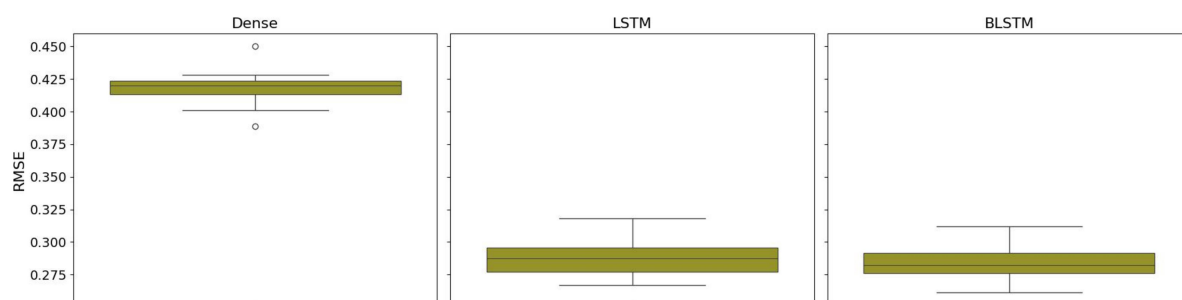


Figura 6.28: Resultados da métrica RMSE nos modelos Dense, LSTM e BLSTM.

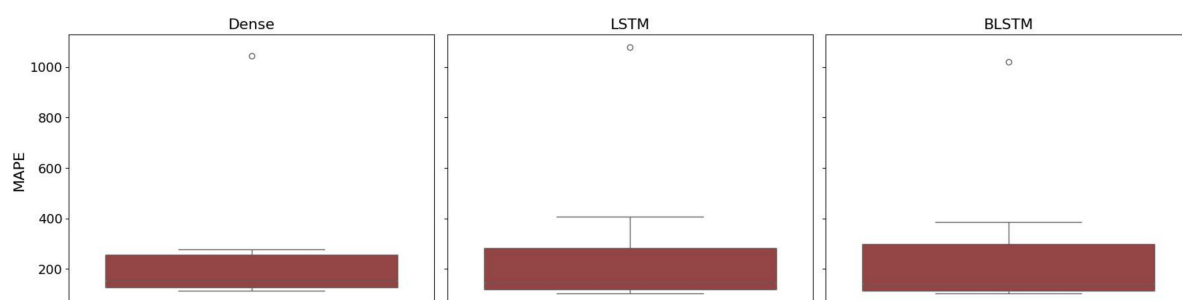


Figura 6.29: Resultados da métrica MAPE nos modelos Dense, LSTM e BLSTM.

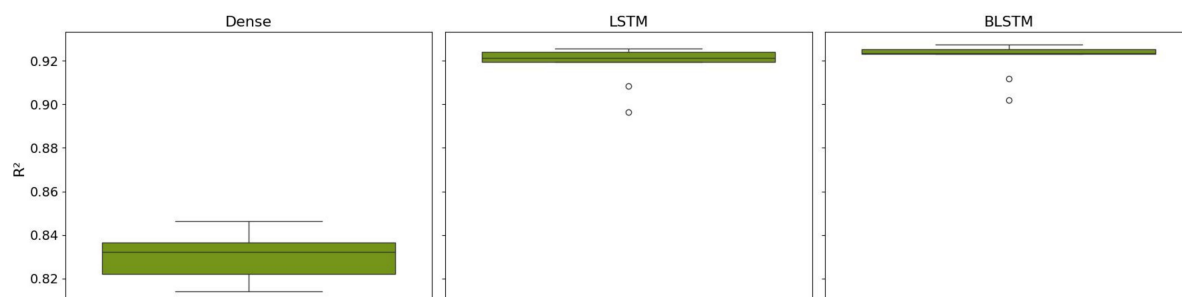


Figura 6.30: Resultados da métrica R^2 nos modelos Dense, LSTM e BLSTM.

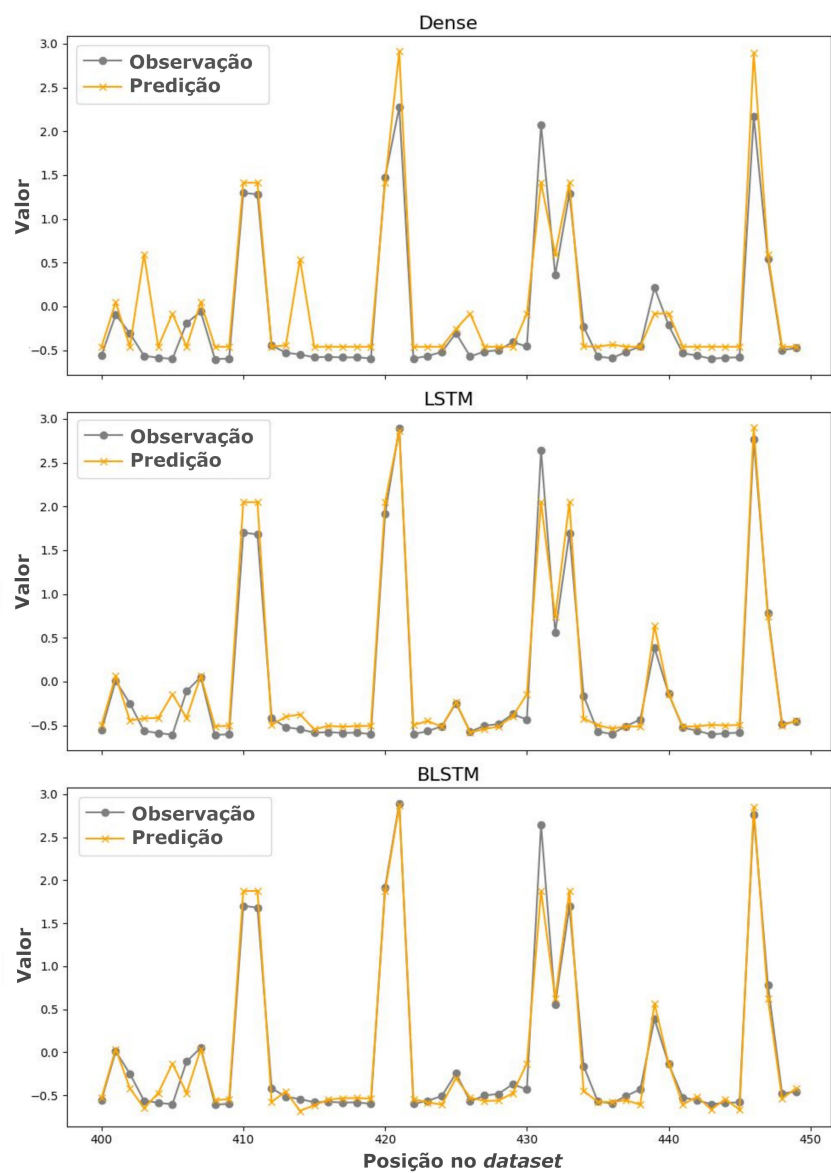


Figura 6.31: Comparativo entre predições e observações.

seguida de estabilização em níveis mais baixos, indicando comportamento de convergência adequado.

Embora sejam observadas pequenas oscilações ao longo do treinamento, essas variações ocorrem de maneira semelhante tanto nos conjuntos de treino quanto de validação, o que sugere que tais flutuações estão associadas à variabilidade intrínseca dos dados e não a problemas de sobreajuste. Entre os modelos, o Dense apresenta uma convergência mais suave, enquanto o LSTM exibe maior variabilidade. Já o BLSTM se destaca por apresentar curvas mais estáveis e alinhadas, indicando maior robustez durante o processo de aprendizado.

As Figuras 6.26 a 6.30 apresentam os resultados das métricas de regressão para os três modelos avaliados. A Figura 6.26 mostra os resultados da métrica MAE. Observa-se que o modelo BLSTM apresenta a menor mediana de erro absoluto, além de uma distribuição mais concentrada, indicando maior precisão e estabilidade em suas predições. Os modelos Dense e LSTM apresentam erros mais elevados e maior dispersão.

Resultados semelhantes são observados na Figura 6.27, referente à métrica MSE. O BLSTM novamente apresenta os menores valores, demonstrando menor sensibilidade a erros mais elevados. O LSTM apresenta desempenho intermediário, enquanto o Dense mantém os maiores valores de erro.

Na análise da RMSE, apresentada na Figura 6.28, o comportamento se mantém consistente: o BLSTM apresenta os menores valores e menor variabilidade, indicando melhor capacidade de generalização. As diferenças entre LSTM e Dense são menos pronunciadas, mas ambos permanecem inferiores ao BLSTM.

A Figura 6.29 apresenta os resultados da métrica MAPE. Observa-se maior variabilidade em todos os modelos, o que é esperado devido à sensibilidade dessa métrica a valores de baixa magnitude. Ainda assim, o BLSTM mantém a menor mediana de erro percentual, confirmando sua superioridade.

Por fim, a Figura 6.30 apresenta os resultados do coeficiente de determinação (R^2). O modelo BLSTM apresenta valores concentrados próximos ao limite superior da métrica, indicando elevada capacidade explicativa. O LSTM também apresenta desempenho elevado, porém ligeiramente inferior, enquanto o Dense apresenta desempenho significativamente mais baixo, evidenciando menor capacidade de modelar a variabilidade dos dados.

A Figura 6.31 apresenta a comparação direta entre os tempos de execução observados e os valores preditos pelos modelos. O modelo Dense apresenta maiores desvios, especialmente em regiões com picos acentuados de tempo de execução, indicando dificuldade em capturar variações abruptas. O modelo LSTM apresenta melhora significativa, sendo capaz de acompanhar melhor o comportamento geral da série, embora ainda apresente

discrepâncias em pontos de alta variação.

O modelo BLSTM, por sua vez, apresenta o melhor alinhamento entre valores observados e preditos. As suas predições acompanham de forma mais fiel tanto o comportamento global quanto as variações locais da série, incluindo picos e pontos de inflexão. Esse resultado evidencia sua maior capacidade de capturar dependências temporais bidirecionais e modelar padrões complexos presentes nos dados.

De forma geral, os resultados demonstram que o retreino do modelo com dados enriquecidos por variações de entrada contribuiu significativamente para a melhoria da capacidade preditiva do Orama⁺⁺. Entre as arquiteturas avaliadas, o modelo BLSTM apresentou desempenho superior em todas as métricas analisadas, além de maior estabilidade durante o treinamento e melhor aderência aos dados observados.

Esses resultados indicam que a incorporação de informações relacionadas ao nível de entrada, aliada ao uso de arquiteturas recorrentes bidirecionais, é fundamental para capturar padrões complexos de execução em ambientes FaaS. Dessa forma, o BLSTM foi selecionado como o modelo preditivo padrão do Orama⁺⁺, proporcionando estimativas mais precisas e confiáveis para suporte à tomada de decisão em cenários reais de implantação.

6.9 Predições de Tempo de Execução e Custos

A escolha do provedor de nuvem mais adequado em ambientes FaaS é uma tarefa árdua, pois envolve a análise conjunta de métricas de desempenho e custo, além de depender das características específicas de cada aplicação. Em cenários reais, essa avaliação geralmente exige a implementação e execução das aplicações em múltiplos provedores, o que implica alto custo operacional e esforço significativo. Nesse contexto, foram realizadas estimativas de tempo de execução e custo utilizando o Orama⁺⁺ para diferentes casos de uso, com o objetivo de avaliar sua capacidade de apoiar a tomada de decisão sem a necessidade de provisionamento real dos ambientes. Essas estimativas permitem antecipar o comportamento das aplicações em diferentes provedores, viabilizando uma análise comparativa eficiente e de baixo custo.

A Tabela 6.7 apresenta as estimativas de tempo de execução e custo geradas pelo Orama⁺⁺ para diferentes casos de uso, incluindo aplicações nativas do *framework* e *workloads* provenientes de trabalhos relacionados. Todas as estimativas consideram um cenário padronizado com nível de concorrência igual a 1, 10 mil invocações, 1 vCPU e 2GB de memória, permitindo uma comparação direta entre provedores e configurações.

De forma geral, os resultados evidenciam a capacidade do Orama⁺⁺ de identificar diferenças significativas de desempenho e custo entre provedores, mesmo sem a necessidade

Tabela 6.7: Estimativa de tempo de execução e custos para diferentes provedores e casos de uso.

Entrada	Provedor	Configuração	Tempo (ms)	Preço (US\$)
Calculadora				
Nenhuma	AWS	<i>sa-east-1-first-6-billions</i>	2862.51	0.96
Nenhuma	AWS	<i>sa-east-1-next-9-billions</i>	2862.51	0.86
Nenhuma	GCP	<i>tier-1</i>	1840.47	0.53
Nenhuma	GCP	<i>tier-2</i>	1840.47	0.74
Nenhuma	Azure	<i>all-regions</i>	896.30	0.29
Nenhuma	Azure	<i>all-regions-flex</i>	896.30	0.67
Nenhuma	Alibaba	<i>tier-1</i>	429.90	0.11
Nenhuma	Alibaba	<i>tier-2</i>	429.90	0.10
API com armazenamento em objetos (POST)				
Nenhuma	AWS	<i>sa-east-1-first-6-billions</i>	1653.47	0.55
Nenhuma	AWS	<i>sa-east-1-next-9-billions</i>	1653.47	0.50
Nenhuma	GCP	<i>tier-1</i>	1210.37	0.35
Nenhuma	GCP	<i>tier-2</i>	1210.37	0.49
Nenhuma	Azure	<i>all-regions</i>	3455.17	1.11
Nenhuma	Azure	<i>all-regions-flex</i>	3455.17	2.56
Nenhuma	Alibaba	<i>tier-1</i>	12526.32	3.26
Nenhuma	Alibaba	<i>tier-2</i>	12526.32	2.77
API com DBaaS (POST)				
Nenhuma	AWS	<i>sa-east-1-first-6-billions</i>	1623.82	0.54
Nenhuma	AWS	<i>sa-east-1-next-9-billions</i>	1623.82	0.49
Nenhuma	GCP	<i>tier-1</i>	1582.75	0.46
Nenhuma	GCP	<i>tier-2</i>	1582.75	0.64
Nenhuma	Azure	<i>all-regions</i>	1009.57	0.33
Nenhuma	Azure	<i>all-regions-flex</i>	1009.57	0.75
Nenhuma	Alibaba	<i>tier-1</i>	7801.72	2.03
Nenhuma	Alibaba	<i>tier-2</i>	7801.72	1.73
Alinhamento de Sequência Genética				
Baixa	AWS	<i>sa-east-1-first-6-billions</i>	23460.73	7.82
Baixa	AWS	<i>sa-east-1-next-9-billions</i>	23460.73	7.04
Baixa	GCP	<i>tier-1</i>	19285.18	5.53
Baixa	GCP	<i>tier-2</i>	19285.18	7.74
Baixa	Azure	<i>all-regions</i>	16155.54	5.17
Baixa	Azure	<i>all-regions-flex</i>	16155.54	11.96
Baixa	Alibaba	<i>tier-1</i>	14688.45	3.82
Baixa	Alibaba	<i>tier-2</i>	14688.45	3.25
PageRank (SeBS)				
Nenhuma	AWS	<i>sa-east-1-first-6-billions</i>	959.93	0.32
Nenhuma	AWS	<i>sa-east-1-next-9-billions</i>	959.93	0.29
Nenhuma	GCP	<i>tier-1</i>	84.34	0.03
Nenhuma	GCP	<i>tier-2</i>	84.34	0.04
Nenhuma	Azure	<i>all-regions</i>	53.22	0.02
Nenhuma	Azure	<i>all-regions-flex</i>	53.22	0.04
Nenhuma	Alibaba	<i>tier-1</i>	1123.63	0.29
Nenhuma	Alibaba	<i>tier-2</i>	1123.63	0.25
Multiplicação de Matrizes (FaaS-dom)				
Nenhuma	AWS	<i>sa-east-1-first-6-billions</i>	674.50	0.23
Nenhuma	AWS	<i>sa-east-1-next-9-billions</i>	674.50	0.20
Nenhuma	GCP	<i>tier-1</i>	295.32	0.09
Nenhuma	GCP	<i>tier-2</i>	295.32	0.12
Nenhuma	Azure	<i>all-regions</i>	469.36	0.15
Nenhuma	Azure	<i>all-regions-flex</i>	469.36	0.35
Nenhuma	Alibaba	<i>tier-1</i>	1653.17	0.43
Nenhuma	Alibaba	<i>tier-2</i>	1653.17	0.37

de implantação real das aplicações. No caso da aplicação Calculadora, observa-se que o provedor Alibaba apresenta simultaneamente o menor tempo de execução e o menor custo, configurando-se como a melhor escolha. Já o Azure apresenta bom desempenho, porém com custo superior, enquanto AWS e GCP apresentam tempos mais elevados e custos intermediários.

Para a API com armazenamento em objetos, o GCP (*tier-1*) se destaca como a melhor alternativa, apresentando bom equilíbrio entre tempo de execução e custo. Embora o AWS apresente desempenho próximo, seus custos são ligeiramente superiores. O Azure, por sua vez, apresenta maior tempo de execução e custo elevado, enquanto o Alibaba apresenta desempenho significativamente inferior, resultando em custos mais altos.

Na API com DBaaS, o Azure (*all-regions*) apresenta o melhor compromisso entre desempenho e custo, com menor tempo de execução e baixo custo relativo. O GCP e o AWS apresentam desempenho semelhante, porém com custos um pouco mais elevados. O Alibaba novamente apresenta tempos mais altos, impactando negativamente o custo final.

No caso do alinhamento de sequência genética, que representa uma carga mais intensiva, o Alibaba se destaca como a melhor opção, apresentando o menor tempo de execução e o menor custo entre os provedores analisados. O Azure e o GCP apresentam desempenho intermediário, enquanto o AWS apresenta maior tempo de execução e custo mais elevado. Esse resultado demonstra a importância de considerar características específicas da carga de trabalho na escolha do provedor.

Para o PageRank (SeBS)¹, observa-se uma mudança significativa no cenário: o Azure apresenta o melhor desempenho e o menor custo, seguido de perto pelo GCP. O AWS e o Alibaba apresentam tempos mais elevados, tornando-se menos competitivos nesse caso. De forma semelhante, na Multiplicação de Matrizes (FaaS-dom)², o GCP (*tier-1*) se destaca como a melhor opção, com menor tempo de execução e custo reduzido, enquanto o AWS e o Azure apresentam desempenho intermediário e o Alibaba apresenta os piores resultados.

Um aspecto importante desta análise é que os casos de uso PageRank e Multiplicação de Matrizes não fazem parte do conjunto de aplicações originalmente utilizadas no treinamento do modelo. Ainda assim, o Orama⁺⁺ foi capaz de gerar estimativas coerentes e consistentes, evidenciando sua capacidade de generalização para *workloads* distintos. Esse resultado reforça a robustez do modelo preditivo, especialmente após o retreino com dados enriquecidos.

¹Repositório do PageRank: <https://github.com/spcl/serverless-benchmarks>

²Repositório da Multiplicação de Matrizes: <https://github.com/faas-benchmarking/faasdom>

Em resumo, os resultados demonstram que o Orama⁺⁺ permite identificar, de forma antecipada, a melhor escolha de provedor e configuração para cada tipo de aplicação, considerando simultaneamente desempenho e custo. Essa capacidade elimina a necessidade de provisionamento e experimentação em múltiplos ambientes reais, reduzindo esforço, tempo e custos operacionais.

6.10 Síntese dos Resultados e Evolução do Orama⁺⁺

A análise conjunta dos experimentos conduzidos evidencia um processo evolutivo consistente do *framework* Orama⁺⁺, tanto em termos de capacidade experimental quanto de suporte à tomada de decisão em ambientes FaaS. Inicialmente, os Experimentos 1 e 2 demonstraram a viabilidade do *framework* na execução automatizada de *benchmarks* e na comparação entre provedores, ainda em uma arquitetura centralizada. Esses resultados permitiram identificar fatores fundamentais que influenciam o desempenho, especialmente o impacto predominante da concorrência e a existência de *trade-offs* entre latência e confiabilidade.

Com a introdução da arquitetura distribuída nos Experimentos 3, 4 e 5, o Orama⁺⁺ passou a suportar cenários mais realistas e escaláveis, incorporando variabilidade geográfica, altos níveis de concorrência e integrações com serviços externos. Essa evolução ampliou significativamente a capacidade de análise do *framework*, permitindo observar padrões mais complexos de comportamento dos provedores, como diferentes estratégias de escalonamento, variações regionais e impactos da latência de rede. Além disso, evidenciou-se que, embora existam diferenças estruturais entre provedores, o perfil da carga e as características da aplicação exercem influência determinante sobre o desempenho observado.

A consolidação dos dados gerados nesses experimentos viabilizou a construção de um modelo preditivo capaz de estimar o tempo de execução de funções sem a necessidade de implantação real. Inicialmente, o modelo apresentou desempenho consistente, mas com limitações na captura de cenários mais complexos. A incorporação do Experimento 6, com maior variabilidade de entrada e características dinâmicas, representou um avanço significativo, permitindo aprimorar a capacidade de generalização do modelo. O uso de arquiteturas mais sofisticadas, como BLSTM, resultou em ganhos expressivos de precisão e estabilidade nas predições.

Como resultado desse processo evolutivo, o Orama⁺⁺ evolui de uma ferramenta de *benchmarking* para um instrumento efetivo de apoio à decisão. Ao permitir a estimativa prévia de desempenho e custo com base em características do código e do cenário de execução, o *framework* reduz significativamente o esforço necessário para avaliação de alternativas em múltiplos provedores. Dessa forma, torna-se possível selecionar, de maneira

mais objetiva e fundamentada, o ambiente mais adequado para a implantação de uma função FaaS, considerando não apenas métricas isoladas, mas o comportamento esperado em diferentes condições de uso. Vale ressaltar que características dinâmicas, como os parâmetros de entrada das funções, aspectos temporais e mudanças nas estratégias dos provedores podem alterar a precisão das previsões apresentadas pelo *framework*. Por esse motivo, este trabalho possui diversas oportunidades de evolução, conforme será abordado na Seção 7.3.

Dessa forma, considerando as questões de pesquisa apresentadas na Seção 1.1, observa-se que para a questão “*Como reduzir a variabilidade experimental?*”, o Orama⁺⁺ busca reduzir a variabilidade experimental por meio da padronização e do controle sistemático das condições de execução dos *benchmarks* em ambientes FaaS multi-nuvem. Para isso, o *framework* automatiza o provisionamento dos cenários experimentais, garantindo que aplicações equivalentes sejam implantadas com configurações consistentes entre diferentes provedores. Além disso, os experimentos são executados de forma repetitiva e parametrizada, permitindo controlar fatores como número de requisições concorrentes, intervalo entre invocações, quantidade de repetições e execução de chamadas adicionais para identificação de efeitos de *cold start*.

Para a questão “*Como modelar o comportamento heterogêneo de diferentes provedores de nuvem?*”, o *framework* organiza os resultados experimentais em uma base estruturada de métricas e metadados, permitindo que técnicas estatísticas e modelos de Aprendizado de Máquina identifiquem padrões associados à heterogeneidade inerente aos provedores FaaS, como diferenças de escalabilidade, políticas de alocação, *cold starts* e variabilidade de desempenho sob concorrência. Dessa forma, o Orama⁺⁺ fornece subsídios para modelar e prever o comportamento de aplicações em cenários multi-nuvem heterogêneos.

A questão “*Como generalizar a predição de tempo e custo entre os diferentes serviços de FaaS?*” pode ser respondida por meio do modelo de predição baseado em características intrínsecas da aplicação e em parâmetros operacionais independentes de uma plataforma específica. O *framework* combina métricas extraídas do código-fonte, como complexidade estrutural e padrões de processamento, com fatores de execução, como nível de entrada, nível de concorrência e quantidade de memória RAM configurada, permitindo modelar o comportamento da aplicação a partir de atributos semanticamente válidos em diferentes ambientes FaaS. Com isso, o modelo captura relações gerais entre carga de trabalho, configuração operacional e desempenho observado, reduzindo o acoplamento a mecanismos internos de provedores específicos e favorecendo a construção de previsões mais robustas e transferíveis em cenários multi-nuvem.

6.11 Considerações Finais

Este capítulo apresentou uma avaliação abrangente do *framework* Orama⁺⁺, evidenciando sua evolução desde a execução automatizada de *benchmarks* até a incorporação de uma abordagem distribuída e orientada por dados. Os experimentos permitiram identificar que o desempenho em ambientes FaaS é fortemente influenciado por fatores como concorrência, latência, integração com serviços externos e características da aplicação, destacando a necessidade de análises sistemáticas para embasar decisões. A consolidação desses dados viabilizou o treinamento de modelos preditivos capazes de estimar o comportamento das funções sem a necessidade de implantação real.

Com isso, o Orama⁺⁺ consolida-se como uma ferramenta de apoio à decisão, permitindo avaliações mais rápidas, objetivas e fundamentadas sobre a escolha de provedores e configurações de execução. No próximo capítulo, são apresentadas as conclusões deste trabalho, bem como suas principais contribuições e possíveis direções para trabalhos futuros.

Capítulo 7

Conclusão

Esta tese apresentou uma abordagem para apoiar a tomada de decisão sobre o provisionamento de aplicações em ambientes FaaS multi-nuvem. A motivação central residiu na dificuldade de identificar, de forma precisa e antecipada, qual provedor de nuvem oferece o melhor compromisso entre desempenho e custo para uma determinada aplicação, considerando a heterogeneidade das plataformas e a variabilidade inerente ao modelo *serverless*.

Em consonância com o objetivo geral estabelecido nesta tese, foi desenvolvido o Orama⁺⁺, um *framework* capaz de integrar experimentação estruturada, análise estatística e técnicas de IA para apoiar a seleção de provedores FaaS. O *framework* automatiza o provisionamento de cenários experimentais, a execução reproduzível de *benchmarks* e a coleta sistemática de métricas de desempenho em múltiplos provedores de nuvem, atendendo diretamente aos objetivos específicos relacionados à criação de mecanismos de provisionamento automatizado e orquestração de experimentos em ambientes multi-nuvem.

Além disso, o trabalho sistematizou a aplicação de métodos estatísticos, incluindo projeto *fatorial* 2^k e testes *t* pareados, permitindo identificar fatores que influenciam o desempenho das aplicações FaaS. Os resultados experimentais demonstraram que variáveis como concorrência, características das funções e particularidades da infraestrutura dos provedores impactam significativamente o comportamento das aplicações, validando a importância da análise estatística como mecanismo de interpretação e comparação entre plataformas.

Como principal contribuição, este trabalho propõe a integração de técnicas de IA, com ênfase no uso de Redes Neurais, para estimar o comportamento de aplicações FaaS em diferentes provedores. A partir de características do código e de parâmetros de execução, os modelos desenvolvidos são capazes de prever métricas relevantes, tais como tempo de execução e custo, permitindo antecipar cenários de desempenho sem a necessidade de execução exaustiva em todos os ambientes. Essa capacidade de predição constitui um elemento central para o apoio à tomada de decisão, reduzindo incertezas e viabilizando

escolhas mais informadas.

Adicionalmente, a partir das estimativas de tempo de execução produzidas pelos modelos de predição, o Orama⁺⁺ implementa mecanismos para estimativa de custos em diferentes provedores de nuvem, considerando parâmetros associados ao modelo de precificação das plataformas FaaS, tais como custo por invocação, tempo efetivo de execução e recursos computacionais alocados. Com isso, o trabalho também contempla o objetivo específico relacionado à projeção de custos de execução em ambientes *serverless*, possibilitando a comparação prévia entre diferentes cenários de implantação.

O *framework* Orama⁺⁺ atua como a base experimental que viabiliza essa abordagem, sendo responsável pela automação do provisionamento, execução e coleta de dados necessários para o treinamento e validação dos modelos preditivos. Embora também ofereça suporte à execução de *benchmarks* estruturados e reproduzíveis, essa funcionalidade assume um papel complementar, fornecendo os dados empíricos que sustentam os modelos de Aprendizado de Máquina, e permitindo a validação das estimativas geradas.

Os resultados obtidos indicam que o uso de Redes Neurais é eficaz na modelagem do comportamento de aplicações FaaS, capturando relações não lineares entre características do código, parâmetros de entrada e métricas de desempenho. Dessa forma, o trabalho demonstra que é possível avançar de uma abordagem puramente empírica, baseada em experimentação extensiva, para um modelo híbrido que combina experimentação e predição, aumentando significativamente a eficiência do processo de avaliação.

Adicionalmente, evidências desta pesquisa demonstram que o Orama⁺⁺ atende de forma efetiva aos objetivos propostos, ao oferecer uma abordagem integrada que combina experimentação sistemática, análise estatística e modelagem preditiva em ambientes FaaS multi-nuvem. O *framework* destaca-se pela capacidade de automatizar a execução de *benchmarks* reproduzíveis, estruturar a coleta de dados e viabilizar a construção de modelos capazes de estimar, de forma antecipada, o tempo de execução e os custos das aplicações. Dessa forma, o Orama⁺⁺ reduz a dependência de experimentação empírica extensiva, aumenta a previsibilidade do comportamento das aplicações e fornece subsídios concretos para decisões mais informadas na seleção de provedores, consolidando-se como uma solução robusta e alinhada às demandas identificadas nesta tese.

7.1 Desafios

O desenvolvimento desta tese envolveu diversos desafios técnicos, científicos e metodológicos, decorrentes tanto da complexidade inerente aos ambientes FaaS quanto da necessidade de integrar diferentes áreas de conhecimento, incluindo Computação em Nuvem, experimentação computacional, análise estatística e IA. Esses temas estiveram presentes

ao longo de todas as etapas da pesquisa, desde a definição da infraestrutura experimental até a construção e validação dos modelos preditivos propostos.

Um dos principais problemas esteve relacionado à heterogeneidade dos provedores de nuvem. Embora plataformas FaaS ofereçam abstrações semelhantes do ponto de vista conceitual, cada provedor possui mecanismos próprios de provisionamento, gerenciamento de recursos, escalabilidade e monitoramento. Essas diferenças dificultaram a criação de cenários experimentais equivalentes entre plataformas distintas, exigindo adaptações específicas para garantir maior uniformidade na implantação e execução dos *benchmarks*. Adicionalmente, aspectos internos dos provedores, frequentemente não documentados publicamente, introduziram variabilidade nos resultados experimentais, dificultando a interpretação e comparação direta entre ambientes.

Outro aspecto relevante esteve associado à variabilidade inerente ao paradigma *serverless*. Fatores como *cold start*, concorrência, latência de rede, compartilhamento de infraestrutura e políticas internas de escalonamento introduziram oscilações significativas nos tempos de execução observados durante os experimentos. Dessa forma, foi necessário definir estratégias experimentais capazes de reduzir ruídos estatísticos e aumentar a confiabilidade dos resultados coletados. Nesse contexto, a realização de execuções repetitivas, o uso de análises estatísticas formais e a definição criteriosa dos parâmetros experimentais tornaram-se elementos fundamentais para garantir maior robustez metodológica.

A implementação do *framework* Orama⁺⁺ também apresentou desafios significativos. O desenvolvimento de uma infraestrutura capaz de automatizar o provisionamento, a execução distribuída de *benchmarks*, a coleta de métricas e a integração entre múltiplos provedores demandou a construção de mecanismos de abstração e orquestração suficientemente flexíveis para acomodar diferentes arquiteturas e serviços de nuvem. Além disso, a necessidade de manter compatibilidade com múltiplas plataformas e diferentes serviços associados, como armazenamento de objetos e bancos de dados, aumentou substancialmente a complexidade da solução proposta.

Outro aspecto crítico esteve relacionado à construção do *dataset* experimental utilizado no treinamento dos modelos de predição. A obtenção de dados representativos exigiu a execução de um grande volume de experimentos em diferentes provedores, regiões geográficas, níveis de concorrência e configurações de aplicação. Esse processo demandou elevado esforço computacional, gerenciamento cuidadoso dos custos associados aos serviços de nuvem e mecanismos para armazenamento, organização e validação dos dados coletados. Adicionalmente, a variabilidade dos ambientes exigiu cuidados adicionais no tratamento e normalização dos dados antes de sua utilização no treinamento dos modelos de IA.

No contexto da modelagem preditiva, uma das principais dificuldades consistiu na

identificação de características capazes de representar adequadamente o comportamento das aplicações FaaS. Como o objetivo da pesquisa envolvia prever o tempo de execução antes da implantação das funções, tornou-se necessário investigar métricas extraídas diretamente do código-fonte, incluindo métricas de complexidade de software, como métricas de Halstead e indicadores associados ao tamanho e à estrutura das funções. A definição dessas características e sua correlação com métricas reais de desempenho exigiram diversas etapas de experimentação e refinamento dos modelos.

Além disso, o treinamento de modelos baseados em Redes Neurais apresentou dificuldades relacionadas à generalização das predições. Como os ambientes FaaS são altamente dinâmicos, pequenas alterações na infraestrutura dos provedores podem impactar significativamente o comportamento das aplicações. Dessa forma, foi necessário buscar estratégias capazes de minimizar problemas de sobreajuste (*overfitting*) e aumentar a capacidade de generalização dos modelos para diferentes cenários de execução.

Outro assunto importante esteve relacionado à reprodutibilidade experimental. Em ambientes de Computação em Nuvem, fatores externos ao controle do pesquisador, como mudanças na infraestrutura dos provedores, indisponibilidades temporárias de serviços e variações de desempenho decorrentes do compartilhamento de recursos, dificultam a reprodução exata dos experimentos ao longo do tempo. Assim, a definição de mecanismos automatizados de provisionamento, coleta de dados e documentação dos cenários experimentais tornou-se essencial para aumentar a transparência e a repetibilidade da pesquisa.

Além dos desafios técnicos e científicos, a pesquisa também envolveu dificuldades relacionadas à evolução contínua das tecnologias de Computação em Nuvem. Durante o período de desenvolvimento da tese, diversos provedores atualizaram serviços, modificaram APIs, alteraram políticas de precificação e introduziram novos mecanismos de gerenciamento em suas plataformas FaaS. Essas mudanças exigiram constantes adaptações no *framework* Orama⁺⁺ e nos experimentos realizados, impactando diretamente o processo de desenvolvimento e validação da solução proposta.

Por fim, destaca-se que a própria natureza interdisciplinar desta pesquisa representou uma preocupação adicional. A integração entre Computação em Nuvem, Análise Estatística, Engenharia de Software e Aprendizado de Máquina exigiu a combinação de diferentes metodologias, ferramentas e fundamentos teóricos. Apesar dessas dificuldades, os problemas enfrentados contribuíram diretamente para o amadurecimento da proposta desenvolvida nesta tese, resultando em uma abordagem mais robusta, abrangente e alinhada às demandas reais de avaliação e predição em ambientes FaaS multi-nuvem.

7.2 Ameaças à Validade

A avaliação experimental de plataformas de Computação em Nuvem, especialmente no paradigma *serverless*, apresenta limitações relacionadas à falta de controle sobre a infraestrutura dos provedores e às variações do ambiente de execução. No contexto da validade de conclusão, a principal ameaça está na instabilidade natural desses ambientes, em que fatores como concorrência de recursos entre usuários e o efeito de *cold start* podem alterar os tempos de execução observados. Para reduzir esse impacto, o Orama⁺⁺ utilizou um grande volume de execuções experimentais, totalizando 332.190 testes, além de parametrizações específicas para separar os efeitos de inicialização das funções. Também foram aplicados métodos estatísticos para garantir que os resultados fossem consistentes e não consequência de oscilações momentâneas da infraestrutura ou da rede.

Quanto à validade interna, a principal dificuldade foi garantir que os resultados observados fossem realmente causados pelas variáveis analisadas, e não por fatores externos. Diferenças geográficas entre os ambientes de teste e os *data centers* dos provedores poderiam influenciar os tempos de resposta devido à latência de rede. Para minimizar esse problema, foi utilizada uma infraestrutura distribuída com *workers* localizados em diferentes regiões. Além disso, o módulo automatizado de provisionamento do Orama⁺⁺ garantiu implantações com configurações equivalentes em todos os provedores avaliados, permitindo comparações mais justas entre as plataformas.

Em relação à validade de construto, buscou-se assegurar que as métricas utilizadas representassem adequadamente as características analisadas. A proposta desta tese parte da hipótese de que o tempo de execução e o custo de funções FaaS podem ser estimados a partir de propriedades extraídas do código-fonte antes da implantação. Para isso, foram utilizadas métricas de complexidade de Halstead para representar volume e complexidade algorítmica do software. As Redes Neurais Artificiais foram empregadas para identificar relações não lineares entre essas métricas e o comportamento das aplicações, tendo sua eficácia validada pela comparação entre as previsões geradas e os resultados obtidos em ambiente real.

Por fim, a validade externa está relacionada à capacidade de generalizar os resultados desta tese para outros cenários e aplicações. Para reduzir o risco de especialização excessiva dos modelos, foram utilizados quatro estudos de caso distintos, incluindo aplicações simples, aplicações orientadas a dados e uma aplicação científica para alinhamento de sequências genéticas. Além disso, como os provedores de nuvem realizam atualizações constantes e pouco transparentes em suas infraestruturas, a acurácia dos modelos pode variar ao longo do tempo. Para lidar com essa limitação, o Orama⁺⁺ foi desenvolvido de forma modular e com código aberto, permitindo a execução contínua de novos *benchmarks* e o retreinamento dos modelos de IA.

7.3 Trabalhos Futuros

A partir das limitações apresentadas no Capítulo 4, e como continuidade deste trabalho, diversas direções podem ser exploradas para ampliar o suporte à tomada de decisão em ambientes FaaS. Uma primeira linha de investigação envolve o aprimoramento dos modelos de Aprendizado de Máquina, incluindo a exploração de arquiteturas mais avançadas de Redes Neurais e técnicas de aprendizado contínuo, permitindo adaptar os modelos a mudanças dinâmicas nos provedores de nuvem. A incorporação de novas variáveis explicativas, como métricas de infraestrutura e padrões de carga em tempo real, também pode contribuir para aumentar a precisão das previsões.

Adicionalmente, a expansão do escopo para cenários mais complexos, como aplicações compostas por múltiplas funções, fluxos orientados a eventos e ambientes híbridos, pode enriquecer a capacidade analítica do *framework*. Esses cenários refletem com maior fidelidade o uso real de FaaS e apresenta novas oportunidades para modelagem e previsão.

Uma outra direção consiste na utilização de modelos de linguagem de grande escala para a geração automática de novos casos de uso. Especificamente, pode-se explorar a criação de implementações de algoritmos clássicos, como algoritmos de ordenação e busca, em diferentes linguagens e padrões de implementação. Essa abordagem pode ampliar significativamente o conjunto de cenários avaliados pelo *framework*, contribuindo para a diversidade dos dados utilizados no treinamento dos modelos, e possibilitando análises mais abrangentes sobre o comportamento de aplicações FaaS.

Outra possibilidade de continuidade deste trabalho consiste na validação das previsões realizadas para aplicações que não foram utilizadas durante o treinamento dos modelos. Para isso, pretende-se confrontar os resultados estimados pelo *framework* com resultados reais obtidos a partir da implantação e execução de casos de uso descritos na literatura científica. Além de permitir uma análise mais robusta da capacidade de generalização dos modelos propostos, essa abordagem também possibilita comparar os resultados obtidos pelo *framework* com as estratégias de previsão adotadas em trabalhos relacionados, contribuindo para uma avaliação mais abrangente da efetividade da solução proposta.

Por fim, a redução do acoplamento entre componentes e a definição de interfaces mais padronizadas para integração com provedores representam um passo importante para aumentar a extensibilidade da solução. Isso pode facilitar a incorporação de novas plataformas e ampliar o alcance do *framework*, consolidando-o como uma ferramenta de apoio à decisão em ambientes de Computação em Nuvem cada vez mais diversos.

Em complemento às contribuições técnicas e científicas apresentadas nesta tese, destaca-se que os resultados também se materializaram na produção e disseminação de conhecimento por meio de publicações acadêmicas. Essas produções refletem a evolução do trabalho ao longo do doutorado, bem como a validação das abordagens propostas pela

comunidade científica. Nesse contexto, a próxima seção apresenta a produção científica derivada desta tese, evidenciando os principais trabalhos publicados e sua relação com as contribuições desenvolvidas.

7.4 Produção Científica Derivada desta Tese

Os resultados desta pesquisa foram apresentados por meio de publicações científicas em conferências e periódicos, tanto nacionais quanto internacionais, além de um capítulo de livro publicado no exterior. Essas produções refletem o amadurecimento da investigação ao longo do doutorado e evidenciam a validação dos resultados pela comunidade científica.

A Tabela 7.1 apresenta trabalhos relacionados às áreas de Computação em Nuvem, FaaS, *Fog Computing* e aplicações de IA. Esses trabalhos foram desenvolvidos ao longo da trajetória de pesquisa do autor e de seus colaboradores, contribuindo para o aprofundamento do conhecimento necessário para a identificação das lacunas existentes na literatura e para a definição do problema de pesquisa desta tese.

Tabela 7.1: Publicações adjacentes ao tema da tese.

Anexo	Ano	Autoria	Tipo de publicação	Local de publicação	Qualis	Título
IV [159]	2021	Autor	Conferência Internacional	CARLA ¹	B3	<i>Performance Analysis of Main Public Cloud Big Data Services Processing Brazilian Government Data</i>
V [160]	2021	Autor	Conferência Nacional	WSCAD ²	B4	<i>A Framework for Executing Protein Sequence Alignment in Cloud Computing Services</i>
VI [161]	2021	Co-autor	Conferência Nacional	ERAD-CO ³	não indexado	Análise de Desempenho de Funções como Serviço em Nuvem Privada
VII [162]	2021	Co-autor	Conferência Nacional	ERAD-CO ³	não indexado	Framework de Autenticação para FaaS em Nuvem Pública
VIII [163]	2021	Co-autor	Conferência Nacional	ERAD-CO ³	não indexado	Análise do Custo-benefício de Funções como Serviço e Infraestrutura como Serviço
IX [164]	2021	Autor	Conferência Nacional	ERAD-CO ³	não indexado	Function-as-a-Service: Desenvolvendo Aplicações na Próxima Geração da Computação em Nuvem
X [165]	2022	Co-autor	Periódico Internacional	ACM CSUR ⁴	A1	<i>Orchestration in Fog Computing: A Comprehensive Survey</i>
XI [166]	2022	Co-autor	Conferência Internacional	CLOSER ⁵	A3	<i>From the Sky to the Ground: Comparing Fog Computing with Related Distributed Paradigms</i>
XII [167]	2022	Co-autor	Conferência Internacional	CLOSER ⁵	A3	<i>Comparison of FaaS Platform Performance in Private Clouds</i>
XIV [168]	2022	Co-autor	Periódico Internacional	Computer Networks	A2	<i>Monitoring fog computing: A review, taxonomy and open challenges</i>
XV [169]	2022	Autor	Conferência Internacional	WEBIST ⁶	A4	<i>Deep-vacuity: A Proposal of a Machine Learning Platform based on High-performance Computing Architecture for Insights on Government of Brazil Official Gazettes</i>
XVII [170]	2023	Co-autor	Periódico Internacional	ACM CSUR ⁴	A1	<i>Computational Resource Allocation in Fog Computing: A Comprehensive Survey</i>
XXI [14]	2023	Autor	Periódico Internacional	IEEE Access	A1	<i>FaaS-oriented Node.js applications in an RPC approach using the Node2FaaS framework</i>
XXIV [171]	2024	Co-autor	Conferência Internacional	MobiQuitous ¹⁰	A4	<i>Achieving Observability on Fog Computing with the use of open-source tools</i>
XXVI [172]	2024	Co-autor	Conferência Internacional	UCC ¹²	A3	<i>A Cost-Efficient Resource Allocation for Fog Computing with Users and Providers Perspective</i>

Entre essas publicações, destacam-se estudos sobre orquestração em ambientes de *Fog Computing* [165], análises comparativas entre paradigmas distribuídos [166] e investigações sobre desempenho de plataformas FaaS em nuvens privadas [167]. Esses trabalhos forneceram subsídios teóricos e experimentais que contribuíram diretamente para a concepção do *framework* Orama⁺⁺, e para a definição dos experimentos realizados nesta tese.

Tabela 7.2: Publicações diretamente relacionadas à tese

Anexo	Ano	Autoria	Tipo de publicação	Local de publicação	Qualis	Título
XIII [149]	2022	Autor	Conferência Internacional	CLOSER ⁵	A3	<i>Orama: A Benchmark Framework for Function-as-a-Service</i>
XVI [173]	2022	Autor	Periódico Internacional	CCPE ⁷	A2	<i>AFMC: An alignment framework for multiple computing services and providers</i>
XVIII [150]	2023	Autor	Periódico Internacional	SNCS ⁸	não indexado ¹⁵	<i>Insights into the Performance of Function-as-a-Service Oriented Environments Using the Orama Framework</i>
XIX [151]	2023	Autor	Conferência Internacional	CLOSER ⁵	A3	<i>FaaS Benchmarking over Orama Framework's Distributed Architecture</i>
XX [152]	2023	Autor	Conferência Internacional	CISTI ⁹	A4	<i>How Leading Public Cloud Providers Deliver FaaS Environments: A Comparative Study</i>
XXII [153]	2023	Autor	Conferência Nacional	WSCAD ²	B4	<i>How FaaS with DBaaS performs in different regions: an evaluation by the Orama Framework</i>
XXIII [174]	2024	Autor	Periódico Internacional	SNCS ⁸	não indexado ¹⁵	<i>Main FaaS Providers Behavior Under High Concurrency: An Evaluation with Orama Framework Distributed Architecture</i>
XXV [175]	2024	Autor	Capítulo de livro	CCSS ¹¹	não indexado	<i>Empowering Statistical Analysis in FaaS Environments Through the Orama Framework</i>
XXVII [176]	2025	Autor	Conferência Nacional	SSCAD ¹³	B4	<i>Predicting FaaS Runtime with the Orama Framework Using Machine Learning</i>
XXVIII [177]	2025	Autor	Conferência Nacional	ERI-GO ¹⁴	não indexado	<i>Aplicação de Machine Learning à Predição de Tempo de Execução em FaaS com o Framework Orama</i>
XXIX [120]	2026	Autor	Conferência Internacional	CLOSER ⁵	A3	<i>Orama++: Extending Serverless Benchmarking with Tiobe and Halstead Metrics for Improved Performance Prediction</i>

A Tabela 7.2 apresenta as publicações diretamente relacionadas aos resultados desta pesquisa. Esses trabalhos descrevem diferentes etapas do desenvolvimento do *framework* Orama⁺⁺ e da investigação experimental conduzida ao longo do doutorado.

¹Latin American High Performance Computing Conference

²Simpósio em Sistemas Computacionais de Alto Desempenho

³Escola Regional de Alto Desempenho do Centro-Oeste

⁴ACM Computing Surveys

⁵International Conference on Cloud Computing and Services Science

⁶International Conference on Web Information Systems and Technologies

⁷Concurrency and Computation: Practice and Experience

⁸Springer Nature Computer Science

⁹Iberian Conference on Information Systems and Technologies

¹⁰Mobile and Ubiquitous Systems: Computing, Networking and Services

¹¹Springer Nature Cloud Computing and Services Science

¹²IEEE International Conference on Utility and Cloud Computing

¹³Simpósio em Sistemas Computacionais e Arquiteturas de Alto Desempenho

¹⁴Escola Regional de Informática de Goiás

¹⁵Periódico lançado em 2020 e ainda não indexado pela CAPES

Entre essas publicações, destaca-se a proposta inicial do *framework* Orama⁺⁺ [149], estudos experimentais sobre o comportamento de provedores FaaS [151, 174], análises comparativas entre provedores públicos de nuvem [152] e investigações sobre o impacto de bancos de dados e regiões geográficas no desempenho das funções [153]. Mais recentemente, foram publicados trabalhos relacionados ao uso de técnicas de Aprendizado de Máquina para predição de tempo de execução em ambientes FaaS [120, 176].

Em reconhecimento às contribuições, os trabalhos “*Comparison of FaaS Platform Performance in Private Clouds*” [167] (Anexo XII), “*From the Sky to the Ground: Comparing Fog Computing with Related Distributed Paradigms*” [166] (Anexo XI) e “*Orama++: Extending Serverless Benchmarking with Tiobe and Halstead Metrics for Improved Performance Prediction*” [120] (Anexo XXIX) receberam indicações para “*best paper*” nas conferências onde foram apresentados.

Essas publicações evidenciam a evolução progressiva da pesquisa, desde a construção da infraestrutura experimental até o desenvolvimento de modelos de predição de desempenho, consolidando a relevância científica e a contribuição desta tese para a área de Computação em Nuvem e sistemas *serverless*. Por fim, para proteger a propriedade intelectual apresentada nesta tese, o *framework* Orama⁺⁺ foi registrado no INPI como “Programa de Computador” sob nº BR512026002022-0 (Anexo II) e outro software, o *framework* Node2FaaS [14], produzido de forma adjacente a este trabalho, também está registrado no INPI sob nº BR512026002212-6 (Anexo III).

Referências

- [1] Mell, Peter, Tim Grance *et al.*: *The nist definition of cloud computing*. 2011. 1, 11, 12, 13
- [2] Nupponen, Jussi e Davide Taibi: *Serverless: What it is, what to do and what not to do*. Em *2020 IEEE ICSCA-C*, páginas 49–50, 2020. 1, 12, 14
- [3] Schleier-Smith, Johann, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica e David A. Patterson: *What serverless computing is and should become: The next phase of cloud computing*. ACM, 64(5):76–84, abril 2021, ISSN 0001-0782. 1, 14, 15, 17, 22, 30
- [4] AWS: *AWS lambda*. <https://aws.amazon.com/en/lambda>, 2021. <https://aws.amazon.com/pt/lambda>, [online; 11-Aug-2021]. 1, 14
- [5] Google: *Cloud functions*. <https://cloud.google.com/functions/>, 2021. <https://cloud.google.com/functions/>, acesso em 2021/08/10, [Online; 10-Aug-2021]. 1, 14
- [6] Microsoft: *Azure functions*. <https://azure.microsoft.com/pt-br/services/functions/>, 2021. <https://azure.microsoft.com/pt-br/services/functions/>, acesso em 2021/08/10, [online; 11-Aug-2021]. 1, 15
- [7] Cloud, Alibaba: *Alibaba cloud function*. <https://www.alibabacloud.com/product/function-compute>, 2021. <https://www.alibabacloud.com/product/function-compute>, acesso em 2021/11/23, [online; 11-Aug-2021]. 1, 15
- [8] Horovitz, Shay, Roei Amos, Ohad Baruch, Tomer Cohen, Tal Oyar e Afik Deri: *Faastest - machine learning based cost and performance faas optimization*. Em Coppola, Massimo, Emanuele Carlini, Daniele D’Agostino, Jörn Altmann e José Ángel Bañares (editores): *Economics of Grids, Clouds, Systems, and Services*, páginas 171–186, Cham, 2019. Springer International Publishing, ISBN 978-3-030-13342-9. 2, 30, 99, 103, 104
- [9] Eismann, Simon, Long Bui, Johannes Grohmann, Cristina Abad, Nikolas Herbst e Samuel Kounev: *Sizeless: predicting the optimal size of serverless functions*. Em *Proceedings of the 22nd International Middleware Conference*, Middleware ’21, página 248–259, New York, NY, USA, 2021. Association for Computing Machinery, ISBN 9781450385343. <https://doi.org/10.1145/3464298.3493398>. 2, 100, 103, 104, 105

- [10] Tomaras, Dimitrios, Michail Tsenos e Vana Kalogeraki: *Prediction-driven resource provisioning for serverless container runtimes*. Em *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, páginas 1–6, 2023. 2, 99, 103, 104, 105
- [11] Filippini, Federica, Luca Cavenaghi, Nicolas Calmi, Marco Savi e Michele Ciavotta: *ML-based performance modeling in edge faas systems*. Em *European Conference on Service-Oriented and Cloud Computing*, páginas 112–127. Springer, 2025. 2, 101, 103, 104
- [12] Jain, Raj: *The art of computer systems: Techniques for experimental design, measurement, simulation, and modeling*, 1991. ISBN:13.978-0471503361. 3, 18, 19
- [13] Hsu, Henry e Peter A. Lachenbruch: *Paired t Test*. John Wiley & Sons, Ltd, 2014, ISBN 9781118445112. 3, 18, 19
- [14] Carvalho, Leonardo Rebouças de e Aletéia Patricia Favacho de Araújo: *FaaS-oriented node.js applications in an rpc approach using the node2faas framework*. IEEE Access, 11:112027–112043, 2023. 3, 152, 154
- [15] Hirschberg, D. S.: *A linear space algorithm for computing maximal common subsequences*. Commun. ACM, 18(6):341–343, junho 1975, ISSN 0001-0782. <https://doi.org/10.1145/360825.360861>. 8, 73
- [16] Halstead, Maurice H: *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977. 9, 54
- [17] Fano, Robert M: *The mac system: The computer utility approach*. IEEE spectrum, 2(1):56–64, 1965. 11
- [18] Surbiryala, Jayachander e Chunming Rong: *Cloud computing: History and overview*. Em *2019 IEEE Cloud Summit*, páginas 1–7, 2019. 11
- [19] Albertin, Alberto Luiz e Rosa Maria de Moura Albertin: *Transformação digital: gerando valor para o "novo futuro"*. Gv-Executivo, 20(1):26–29, 2021. 12
- [20] Amazon Web Services, Inc.: *Amazon web services*, 2025. <https://www.aws.com> [Acessado em: 07/03/2025]. 12
- [21] Microsoft: *Microsoft azure*, 2025. <https://azure.microsoft.com> [Acessado em: 07/03/2025]. 12
- [22] Cloud, Google: *Google cloud platform*, 2025. <https://cloud.google.com> [Acessado em: 07/03/2025]. 12
- [23] Cloud, Google: *Google app engine*, 2025. <https://cloud.google.com/appengine> [Acessado em: 07/03/2025]. 12
- [24] Heroku: *Heroku*, 2025. <https://www.heroku.com> [Acessado em: 07/03/2025]. 12
- [25] Azure, Microsoft: *Microsoft azure app service*, 2025. <https://azure.microsoft.com/products/app-service> [Acessado em: 07/03/2025]. 12

- [26] Google: *Google workspaces*, 2025. <https://workspace.google.com> [Acessado em: 07/03/2025]. 13
- [27] Microsoft: *Office 365*, 2025. <https://www.office.com> [Acessado em: 07/03/2025]. 13
- [28] Salesforce, Inc.: *Salesforce*, 2025. <https://www.salesforce.com> [Acessado em: 07/03/2025]. 13
- [29] Duan, Yucong, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C. Narendra e Bo Hu: *Everything as a service (xaas) on the cloud: Origins, current and future trends*. Em *2015 IEEE 8th International Conference on Cloud Computing*, páginas 621–628, 2015. 13
- [30] Lehner, Wolfgang e Kai Uwe Sattler: *Database as a service (dbaas)*. Em *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, páginas 1216–1217. IEEE, 2010. 13, 25
- [31] Hussain, Mohammed e Hanady Abdulsalam: *Secaas: security as a service for cloud-based applications*. Em *Proceedings of the Second Kuwait Conference on e-Services and e-Systems*, páginas 1–4, 2011. 13
- [32] Syed, Naeem, Adnan Anwar, Zubair Baig e Sherali Zeadally: *Artificial intelligence as a service (aiaas) for cloud, fog and the edge: State-of-the-art practices*. ACM Computing Surveys, 2025. 13, 30
- [33] Google: *Firebase*, 2025. <https://firebase.google.com> [Acessado em: 18/03/2025]. 15, 71
- [34] Google: *Google kubernetes engine*, 2025. <https://cloud.google.com/kubernetes-engine> [Acessado em: 18/03/2025]. 15
- [35] Microsoft: *Azure storage*, 2025. <https://azure.microsoft.com/pt-br/products/category/storage> [Acessado em: 18/03/2025]. 15
- [36] Microsoft: *What is azure event grid?*, 2025. <https://learn.microsoft.com/en-us/azure/event-grid/overview> [Acessado em: 18/03/2025]. 15
- [37] Michelson, Brenda M: *Event-driven architecture overview*. Patricia Seybold Group, 2(12):10–1571, 2006. 15
- [38] Ding, Jie, Vahid Tarokh e Yuhong Yang: *Model selection techniques: An overview*. IEEE Signal Processing Magazine, 35(6):16–34, 2018. 17, 18
- [39] Lawless, Harry T, Hildegard Heymann, Harry T Lawless e Hildegard Heymann: *Descriptive analysis*. Sensory evaluation of food: Principles and practices, páginas 227–257, 2010. 18
- [40] MacFarland, Thomas W, Jan M Yates, Thomas W MacFarland e Jan M Yates: *Mann–whitney u test*. Introduction to nonparametric statistics for the biological sciences using R, páginas 103–132, 2016. 18

- [41] Kasza, Jessica e Rory Wolfe: *Interpretation of commonly used statistical regression models*. *Respirology*, 19(1):14–21, 2014. 18
- [42] Neyman, Jerzy e Egon Sharpe Pearson: *Ix. on the problem of the most efficient tests of statistical hypotheses*. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933. 18
- [43] Fisher, Ronald Aylmer: *Statistical methods for research workers*. Oliver and Boyd, Edinburgh, Scotland, 1925. 18
- [44] Neyman, Jerzy: *Outline of a theory of statistical estimation based on the classical theory of probability*. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 236(767):333–380, 1937. 18
- [45] Wilcoxon, Frank: *Individual comparisons by ranking methods*. Em *Breakthroughs in statistics: Methodology and distribution*, páginas 196–202. Springer, 1992. 18
- [46] St, Lars, Svante Wold *et al.*: *Analysis of variance (anova)*. *Chemometrics and intelligent laboratory systems*, 6(4):259–272, 1989. 19
- [47] Bortolini, Diogo e Rafael R Obelheiro: *Investigating performance and cost in function-as-a-service platforms*. Em *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, páginas 174–185. Springer, 2019. 20, 22
- [48] Rodrigues, Bárbara Ribeiro: *Investigando desempenho e custo em plataformas faas em diferentes provedores de nuvem*. Dissertação de mestrado, Universidade do Estado de Santa Catarina (UDESC), Centro de Ciências Tecnológicas, Joinville, SC, Brasil, 2019. Disponível em: https://www.udesc.br/arquivos/cct/id_cpmenu/1024/Disserta__o_Final_Investigando_Desempenho_e_Custo_em_FaaS_15874971868852_1024.pdf. 20
- [49] Rittes, Leandro e Adriano Fiorese: *Seleção, pontuação e ranqueamento de provedores serverless utilizando método de decisão multicritério*. Em *Anais do XXI Congresso Latino-Americano de Software Livre e Tecnologias Abertas*, páginas 274–281, Porto Alegre, RS, Brasil, 2024. SBC. <https://sol.sbc.org.br/index.php/latinoware/article/view/31538>. 21
- [50] Chaudhary, Shalini, Devendra Kumar Somwanshi, Vandana Singh Rajawat, Madhav Sharma e Priya Verma: *Optimizing cold start latency in serverless computing: A comprehensive review of techniques and emerging solutions*. Em *Proceedings of the 6th International Conference on Information Management & Machine Intelligence*, páginas 1–8, 2024. 22, 23
- [51] Kondrashov, Leonid, Boxi Zhou, Hancheng Wang e Dmitrii Ustiugov: *The high cost of keeping warm: Characterizing overhead in serverless autoscaling policies*. arXiv preprint arXiv:2509.03104, 2025. 23

- [52] Bucur, Vlad, Catalin Dehelean e Liviu Miclea: *Object storage in the cloud and multi-cloud: State of the art and the research challenges*. Em *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, páginas 1–6. IEEE, 2018. 24
- [53] Needleman, Saul B. e Christian D. Wunsch: *A general method applicable to the search for similarities in the amino acid sequence of two proteins*. *Journal of Molecular Biology*, 48(3):443–453, março 1970. 26, 73
- [54] Turing, Alan: *Computing machinery and intelligence*. *Mind*, 59:433–460, 1950. 29
- [55] Shinde, Pramila P e Seema Shah: *A review of machine learning and deep learning applications*. Em *2018 Fourth International Conference on Computing Communication Control and Automation (ICCCBEA)*, páginas 1–6. IEEE, 2018. 29, 30, 33, 34, 39
- [56] McDermott, Thomas A, Mark R Blackburn e Peter A Beling: *Artificial intelligence and future of systems engineering*. Em *Systems engineering and artificial intelligence*, páginas 47–59. Springer, 2021. 29
- [57] Tajane, Kapil, Prashant Ahire e Rahul Pitale: *Ai-driven cloud resource optimization: An in-depth analysis*. Em *2025 9th International Conference on Computing, Communication, Control and Automation (ICCCBEA)*, páginas 1–5, 2025. 29
- [58] Samanpour, Ali Reza, André Ruegenberg e Robin Ahlers: *The future of machine learning and predictive analytics*. Em *Digital marketplaces unleashed*, páginas 297–309. Springer, 2017. 30, 33
- [59] Wang, Zhaoxia, Victor Joo, Chuan Tong, Xin Xin e Hoong Chor Chin: *Anomaly detection through enhanced sentiment analysis on social media data*. Em *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, páginas 917–922. IEEE, 2014. 30
- [60] Russell, Stuart J. e Peter Norvig: *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 3ª edição, 2010, ISBN 978-0136042594. 30, 31, 39
- [61] Omitaomu, Olufemi A e Haoran Niu: *Artificial intelligence techniques in smart grid: A survey*. *Smart Cities*, 4(2):548–568, 2021. 30
- [62] Malik, Shiza, Khalid Muhammad e Yasir Waheed: *Artificial intelligence and industrial applications-a revolution in modern industries*. *Ain Shams Engineering Journal*, 15(9):102886, 2024. 30, 31
- [63] Akmal, Fadhilah, Ardian Nurcahya, Aldenia Alexandra, Intan Nurma Yulita, Dedy Kristanto e Irwan Ary Dharmawan: *Application of machine learning for estimating the physical parameters of three-dimensional fractures*. *Applied Sciences*, 14(24):12037, 2024. 31, 34, 35
- [64] Domingos, Pedro: *A few useful things to know about machine learning*. *Communications of the ACM*, 55(10):78–87, 2012. 31, 33, 37, 39

- [65] Ali, Yasir, Fizza Hussain e Md Mazharul Haque: *Advances, challenges, and future research needs in machine learning-based crash prediction models: A systematic review*. Accident Analysis & Prevention, 194:107378, 2024. 31
- [66] James, Gareth, Daniela Witten, Trevor Hastie, Robert Tibshirani e Jonathan Taylor: *Linear regression*. Em *An introduction to statistical learning: With applications in python*, páginas 69–134. Springer, 2023. 31
- [67] Suthaharan, Shan e Shan Suthaharan: *Decision tree learning*. Machine learning models and algorithms for big data classification: thinking with examples for effective learning, páginas 237–269, 2016. 31
- [68] Mammone, Alessia, Marco Turchi e Nello Cristianini: *Support vector machines*. Wiley Interdisciplinary Reviews: Computational Statistics, 1(3):283–289, 2009. 31, 34
- [69] Rigatti, Steven J: *Random forest*. Journal of Insurance Medicine, 47(1):31–39, 2017. 31
- [70] Natekin, Alexey e Alois Knoll: *Gradient boosting machines, a tutorial*. Frontiers in neurorobotics, 7:21, 2013. 31
- [71] Wei, Lu, Zhong Ma, Chaojie Yang e Qin Yao: *Advances in the neural network quantization: A comprehensive review*. Applied Sciences, 14(17):7445, 2024. 32, 39
- [72] Jiang, Chao, Canchen Jiang, Dongwei Chen e Fei Hu: *Densely connected neural networks for nonlinear regression*. Entropy, 24(7):876, 2022. 32
- [73] Tan, Fangbi: *Regression analysis and prediction using lstm model and machine learning methods*. Em *Journal of Physics: Conference Series*, volume 1982, página 012013. IOP Publishing, 2021. 32
- [74] Radman, Abduljalil e Shahrel Azmin Suandi: *Bilstm regression model for face sketch synthesis using sequential patterns*. Neural Computing and Applications, 33(19):12689–12702, 2021. 32
- [75] Lecun, Y., L. Bottou, Y. Bengio e P. Haffner: *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11):2278–2324, 1998. 32
- [76] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser e Illia Polosukhin: *Attention is all you need*. Advances in neural information processing systems, 30, 2017. 32
- [77] Rumelhart, David E, Geoffrey E Hinton e Ronald J Williams: *Learning representations by back-propagating errors*. nature, 323(6088):533–536, 1986. 34, 39
- [78] Willmott, Cort J e Kenji Matsuura: *Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance*. Climate research, 30(1):79–82, 2005. 34, 35

- [79] Rezaei-Dastjerdehei, Mohammad Reza, Amirmohammad Mijani e Emad Fatem-izadeh: *Addressing imbalance in multi-label classification using weighted cross entropy loss function*. Em *2020 27th National and 5th International Iranian Conference on Biomedical Engineering (ICBME)*, páginas 333–338, 2020. 34
- [80] Gentile, Claudio e Manfred KK Warmuth: *Linear hinge loss and average margin*. *Advances in neural information processing systems*, 11, 1998. 34
- [81] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville e Yoshua Bengio: *Generative adversarial networks*. *Communications of the ACM*, 63(11):139–144, 2020. 34
- [82] Açikkar, Mustafa: *Fast grid search: A grid search-inspired algorithm for optimizing hyperparameters of support vector regression*. *Turkish Journal of Electrical Engineering and Computer Sciences*, 32(1):68–92, 2024. 37
- [83] Wang, Xilu, Yaochu Jin, Sebastian Schmitt e Markus Olhofer: *Recent advances in bayesian optimization*. *ACM computing surveys*, 55(13s):1–36, 2023. 38
- [84] Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta e Masanori Koyama: *Optuna: A next-generation hyperparameter optimization framework*. Em *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, páginas 2623–2631, 2019. 38
- [85] Kolluru, Vinoth Kumar, Yagnesh Challagundla, Advaita Naidu Chintakunta, Bapadittya Roy, Amine Bermak e Renuka Devi SM: *Ai-driven energy optimization: household power consumption prediction with lstm networks and pytorch-ray tune in smart iot systems*. Em *2024 International Conference on Microelectronics (ICM)*, páginas 1–6. IEEE, 2024. 38
- [86] The scikit-optimize contributors: *scikit-optimize: Sequential model-based optimization in Python*. <https://github.com/scikit-optimize/scikit-optimize>, 2016–2024. Version 0.10.1. 38
- [87] Kurniawan, Oky, Yuris Alkhalifi, Lady Agustin Fitriana, Muhammad Rifqi Firdaus, Amin Nur Rais e Sofian Wira Hadi: *Hyperparameter tuning optimization on machine learning models to predict software defects*. Em *2024 International Conference on Advanced Information Scientific Development (ICAISD)*, páginas 35–40. IEEE, 2024. 38
- [88] Tian, Yingjie, Yuqi Zhang e Haibin Zhang: *Recent advances in stochastic gradient descent in deep learning*. *Mathematics*, 11(3):682, 2023. 39
- [89] Alasadi, Suad A e Wesam S Bhaya: *Review of data preprocessing techniques in data mining*. *Journal of Engineering and Applied Sciences*, 12(16):4102–4107, 2017. 39, 40
- [90] Sun, Li, Kaile Zhou, Xiaoling Zhang e Shanlin Yang: *Outlier data treatment methods toward smart grid applications*. *IEEE Access*, 6:39849–39859, 2018. 40, 42

- [91] Franchet, Camille, Robin Schwob, Guillaume Bataillon, Charlotte Syrykh, Sarah Péricart, François Xavier Frenois, Frédérique Penault-Llorca, Magali Lacroix-Triki, Laurent Arnould, Jérôme Lemonnier *et al.*: *Bias reduction using combined stain normalization and augmentation for ai-based classification of histological images*. Computers in Biology and Medicine, 171:108130, 2024. 40, 41, 42, 44
- [92] Peng, Liu e Lei Lei: *A review of missing data treatment methods*. Intell. Inf. Manag. Syst. Technol, 1:412–419, 2005. 40
- [93] Mohlin, Erik: *Optimal categorization*. Journal of Economic Theory, 152:356–381, 2014. 41
- [94] Fei, Nanyi, Yizhao Gao, Zhiwu Lu e Tao Xiang: *Z-score normalization, hubness, and few-shot learning*. Em *Proceedings of the IEEE/CVF International Conference on Computer Vision*, páginas 142–151, 2021. 41
- [95] Henderi, Henderi, Tri Wahyuningsih e Efana Rahwanto: *Comparison of min-max normalization and z-score normalization in the k-nearest neighbor (knn) algorithm to test the accuracy of types of breast cancer*. International Journal of Informatics and Information Systems, 4(1):13–20, 2021. 41
- [96] Barbato, Giulio, EM Barini, Gianfranco Genta e Raffaello Levi: *Features and performance of some outlier detection methods*. Journal of Applied Statistics, 38(10):2133–2149, 2011. 42
- [97] Wilcox, R: *Trimming and winsorization*. Encyclopedia of biostatistics, 8, 2005. 42
- [98] Franzese, Monica, Antonella Iuliano *et al.*: *Correlation analysis*. Em *Encyclopedia of bioinformatics and computational biology: ABC of bioinformatics*, volume 1, páginas 706–721. Elsevier, 2018. 42
- [99] Essam, F, Hashash El e Shiekh Raga Hassan Ali: *A comparison of the pearson, spearman rank and kendall tau correlation coefficients using quantitative variables*. Asian J. Probab. Stat, páginas 36–48, 2022. 42
- [100] Mohammed, Somia B, Ahmed Khalid, Saife Eldin F Osman e Rasha Gaffer M Helali: *Usage of principal component analysis (pca) in ai applications*. Int. J. Eng. Res. Technol, 5(12):372–375, 2016. 43
- [101] Hunter, J Stuart: *The exponentially weighted moving average*. Journal of quality technology, 18(4):203–210, 1986. 43, 44
- [102] Tulsyan, Rajiv, Pranjali Shukla, Tushar Singh e Ayush Kumar: *The impact of javascript frameworks on website performance and user experience*. Em *2024 IEEE International Conference on Big Data & Machine Learning (ICBDML)*, páginas 299–305. IEEE, 2024. 48
- [103] Ollila, Risto, Niko Mäkitalo e Tommi Mikkonen: *Modern web frameworks: A comparison of rendering performance*. Journal of Web Engineering, 21(3):789–813, 2022. 48, 86

- [104] Elrom, Elad e Elad Elrom: *React router and material-ui*. React and Libraries: Your Complete Guide to the React Ecosystem, páginas 79–113, 2021. 48, 86
- [105] SHAH, Hezbollah e Tariq Soomro: *Node.js challenges in implementation*. Global Journal of Computer Science and Technology: Network, Web and Security, 17, Junho 2017. 48, 49, 86
- [106] Hahn, Evan. 2016. 48, 49, 55, 86
- [107] Casas, Sandra, Diana Cruz, Graciela Vidal e Marcela Constanzo: *Uses and applications of the openapi/swagger specification: a systematic mapping of the literature*. Em *2021 40th International Conference of the Chilean Computer Science Society (SCCC)*, páginas 1–8. IEEE, 2021. 48, 50, 52, 86
- [108] Pautasso, Cesare e Erik Wilde: *Restful web services: principles, patterns, emerging technologies*. Em *Proceedings of the 19th international conference on World wide web*, páginas 1359–1360, 2010. 49
- [109] Ravshanovich, Azizbek Rajabov: *Database structure: Postgresql database*. PSIXOLOGIYA VA SOTSIOLOGIYA ILMIY JURNALI, 2(7):50–55, 2024. 49
- [110] Uzayr, Sufyan bin, Nicholas Cloud, Tim Ambler, Sufyan bin Uzayr, Nicholas Cloud e Tim Ambler: *Knex and bookshelf*. JavaScript Frameworks for Modern Web Development: The Essential Frameworks, Libraries, and Tools to Learn Right Now, páginas 377–426, 2019. 49, 50, 86
- [111] Xia, Chuanlong, Guangcan Yu e Meng Tang: *Efficient implement of orm (object/relational mapping) use in j2ee framework: Hibernate*. Em *2009 International Conference on Computational Intelligence and Software Engineering*, páginas 1–3. IEEE, 2009. 49
- [112] Truskowski, Wojciech, Rafał Klewek e Maria Skublewska-Paszkowska: *Comparison of mysql, mssql, postgresql, oracle databases performance, including virtualization*. Journal of Computer Sciences Institute, 16:279–284, 2020. 50, 87
- [113] Brikman, Yevgeniy: *Terraform: up and running: writing infrastructure as code*. " O'Reilly Media, Inc.", 2022. 50, 87
- [114] Raptis, Theofanis P e Andrea Passarella: *A survey on networked data streaming with apache kafka*. IEEE access, 11:85333–85350, 2023. 51, 53, 87
- [115] Erinle, Bayo: *Performance testing with JMeter 2.9*. Packt Publishing Ltd, United Kingdom, 2013. ISBN: 9781782165842. 52, 88
- [116] RabbitMQ Team: *Rabbitmq documentation*. <https://www.rabbitmq.com/>, 2024. Acessado em: 16 maio 2024. 53
- [117] Dobbelaere, Philippe e Kyumars Sheykh Esmaili: *Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper*. Em *Proceedings of the 11th ACM international conference on distributed and event-based systems*, páginas 227–238, 2017. 54

- [118] Khan, Bilal e Aamer Nadeem: *Evaluating the effectiveness of decomposed halstead metrics in software fault prediction*. PeerJ Computer Science, 9:e1647, 2023. 54
- [119] Steneker, Maikel: *Towards an empirical validation of the TIOBE Quality Indicator*. Tese de Doutoramento, Doctoral dissertation, Eindhoven University of Technology, 2016. 54
- [120] Carvalho, L. Rebouças de, G. P. Rocha Filho e A. Araujo: *Orama++: Extending serverless benchmarking with tiobe and halstead metrics for improved performance prediction*. Em *Proceedings of the 16th International Conference on Cloud Computing and Services Science*, páginas 13–24, 2026, ISBN 978-989-758-829-7. 55, 153, 154
- [121] Ramos, Miguel Esteban e Marco Tulio Valente: *Análise de métricas estáticas para sistemas javascript*. Em *Anais do II Workshop on Software Visualization, Evolution and Maintenance*, páginas 30–37, 2014. 55, 86
- [122] Mufid, Mohammad Robihul, Arif Basofi, M Udin Harun Al Rasyid, Indhi Farhandika Rochimansyah *et al.*: *Design an mvc model using python for flask framework development*. Em *2019 International Electronics Symposium (IES)*, páginas 214–219. IEEE, 2019. 56, 86
- [123] Koziol, Quincey, Russ Rew, Mark Howison *et al.*: *Hdf5*. Em *High Performance Parallel I/O*, páginas 227–244. Chapman and Hall/CRC, 2014. 56
- [124] Sapre, Atharva e Shubham Vartak: *Scientific computing and data analysis using numpy and pandas*. International Research Journal of Engineering and Technology, 7:1334–1346, 2020. 56, 86
- [125] Hao, Jiangang e Tin Kam Ho: *Machine learning made easy: a review of scikit-learn package in python programming language*. Journal of Educational and Behavioral Statistics, 44(3):348–361, 2019. 56, 86
- [126] Pang, Bo, Erik Nijkamp e Ying Nian Wu: *Deep learning with tensorflow: A review*. Journal of Educational and Behavioral Statistics, 45(2):227–248, 2020. 56, 86
- [127] Reis, David, Bruno Piedade, Filipe F Correia, João Pedro Dias e Ademar Aguiar: *Developing docker and docker-compose specifications: A developers’ survey*. Ieee Access, 10:2318–2329, 2021. 56, 88
- [128] Amazon Web Services: *Amazon simple storage service (s3)*. <https://aws.amazon.com/s3/>, 2022. Acessado em: 19 abr. 2022. 67
- [129] Google Cloud: *Cloud storage*. <https://cloud.google.com/storage>, 2022. Acessado em: 19 abr. 2022. 67
- [130] Microsoft Azure: *Azure blob storage*. <https://azure.microsoft.com/services/storage/blobs/>, 2022. Acessado em: 19 abr. 2022. 67
- [131] Alibaba Cloud: *Object storage service (oss)*. <https://www.alibabacloud.com/product/oss>, 2022. Acessado em: 19 abr. 2022. 67

- [132] Services, Amazon Web: *Dynamodb*, 2025. <https://aws.amazon.com/dynamodb> [Acessado em: 18/03/2025]. 71
- [133] Microsoft: *Cosmosdb*, 2025. <https://azure.microsoft.com/products/cosmos-db> [Acessado em: 18/03/2025]. 71
- [134] Yang, Zhenhai, Kai Tai Fang e Samuel Kotz: *On the student's t-distribution and the t-statistic*. Journal of Multivariate Analysis, 98(6):1293–1304, 2007. 77
- [135] De Carvalho, Leonardo Rebouças e Aleteia Patricia Favacho De Araujo: *Performance comparison of terraform and cloudify as multicloud orchestrators*. Em *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, páginas 380–389. IEEE, 2020. 87
- [136] Copik, Marcin, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski e Torsten Hoefler: *Sebs: A serverless benchmark suite for function-as-a-service computing*, 2021. 96, 99, 104, 105
- [137] Somu, Nikhila, Nilanjan Daw, Umesh Bellur e Purushottam Kulkarni: *Panopticon: A comprehensive benchmarking tool for serverless applications*. Em *2020 COM-SNETS*, páginas 144–151, 2020. 96, 99, 104
- [138] Grambow, Martin, Tobias Pfandzelter, Luk Burchard, Carsten Schubert, Max Zhao e David Bermbach: *Befaas: An application-centric benchmarking framework for faas platforms*, 2021. 96, 99, 104, 105
- [139] Back, Timon e Vasilios Andrikopoulos: *Using a microbenchmark to compare function as a service solutions*. Em *ECSOCC*, páginas 146–160. Springer, 2018. 96, 99, 104
- [140] Kuhlenkamp, Jörn, Sebastian Werner, Maria C. Borges, Karim El Tal e Stefan Tai: *An evaluation of faas platforms as a foundation for serverless big data processing*. Em *Proceedings of the 12th IEEE/ACM, UCC'19*, página 1–9, NY, USA, 2019. ACM, ISBN 9781450368940. 97, 99, 104
- [141] Wen, Jinfeng, Yi Liu, Zhenpeng Chen, Junkai Chen e Yun Ma: *Characterizing commodity serverless computing platforms*, 2020. DOI:10.48550/ARXIV.2012.00992. 97, 99, 104
- [142] Maissen, Pascal, Pascal Felber, Peter Kropf e Valerio Schiavoni: *Faasdom*. Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems, July 2020. <http://dx.doi.org/10.1145/3401025.3401738>. 97, 99, 104
- [143] Barcelona-Pons, Daniel e Pedro García-López: *Benchmarking parallelism in faas platforms*. Future Generation Computer Systems, 124:268–284, 2021, ISSN 0167-739X. 97, 99, 104
- [144] Dakkak, Abdul, Cheng Li, Simon Garcia de Gonzalo, Jinjun Xiong e Wen mei Hwu: *Trims: Transparent and isolated model sharing for low latency deep learning inference in function-as-a-service*. Em *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, páginas 372–382, 2019. 100, 103, 104

- [145] Pandey, Manish e Young Woo Kwon: *Funcmem: reducing cold start latency in serverless computing through memory prediction and adaptive task execution*. Em *Proceedings of the 39th ACM/SIGAPP symposium on applied computing*, páginas 131–138, 2024. 101, 103, 104
- [146] Quevedo, Sebastián, Freddy Merchán, Rafael Rivadeneira e Federico X Dominguez: *Evaluating apache openwhisk-faas*. Em *2019 IEEE fourth ecuador technical chapters meeting (ETCM)*, páginas 1–5. IEEE, 2019. 101
- [147] Cao, Keyan, Yefan Liu, Gongjie Meng e Qimeng Sun: *An overview on edge computing research*. IEEE access, 8:85714–85728, 2020. 101
- [148] Thomas Pusztai, Stefan Nastić: *Chunkfunc: Dynamic slo-aware configuration of serverless functions*. IEEE Electronic Library (IEL) Journals, páginas 1–16, 2025. 102, 103, 104
- [149] Carvalho, Leonardo Reboucas De e Aleteia Patricia Favacho Araujo: *Orama: A benchmark framework for function-as-a-service*. Em *Proceedings of the 12th International Conference on Cloud Computing and Services Science - CLOSER*, páginas 313–322. INSTICC, SciTePress, 2022, ISBN 978-989-758-570-8. 108, 153, 154
- [150] Carvalho, Leonardo Rebouças de e Aleteia Araujo: *Insights into the Performance of Function-as-a-Service Oriented Environments Using the Orama Framework*. SN Computer Science, 4(3):305, 2023. 113, 153
- [151] Carvalho, Leonardo Rebouças de, Bruno Kamienski e Aletéia PF Araújo: *Faas benchmarking over orama framework’s distributed architecture*. Em *CLOSER*, páginas 67–77, 2023. 119, 153, 154
- [152] De Carvalho, Leonardo Rebouças, Flavio Vidal, Bruno Kamienski e Aleteia Araujo: *How leading public cloud providers deliver faas environments: A comparative study*. Em *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*, páginas 1–6, 2023. 123, 153, 154
- [153] Carvalho, Leonardo, Bruno Kamienski e Aleteia Araujo: *How faas with dbaas performs in different regions: an evaluation by the orama framework*. Em *Anais do XXIV Simpósio em Sistemas Computacionais de Alto Desempenho*, páginas 241–252, Porto Alegre, RS, Brasil, 2023. SBC. <https://sol.sbc.org.br/index.php/sscad/article/view/26524>. 128, 153, 154
- [154] Mercatelli, Daniele e Federico M Giorgi: *Geographic and genomic distribution of sars-cov-2 mutations*. Frontiers in microbiology, 11:1800, 2020. 135
- [155] Dawood, Ali A: *Mutated covid-19 may foretell a great risk for mankind in the future*. New microbes and new infections, 35:100673, 2020. 135
- [156] Diller, Julia R, Helen M Parrington, John T Patton e Kristen M Ogden: *Rotavirus species b encodes a functional fusion-associated small transmembrane protein*. Journal of virology, 93(20):10–1128, 2019. 135

- [157] Sundeep, Lakshmi, Lusiba Shaffic, Rakshambika Ramkumar, Mounika Vaidani, Shaik Roshifa, Aswin Yesurethanam e Vijaya Chevula: *Insilico analysis of alpha-1 antitrypsin pathway in ischemic stroke patients*. 2024. 135
- [158] Pal, Mahendra, Gemechu Berhanu, Chaltu Desalegn e Venkataramana Kandi: *Severe acute respiratory syndrome coronavirus-2 (sars-cov-2): an update*. Cureus, 12(3), 2020. 135
- [159] Carvalho, Leonardo Rebouças de, Marcelo Augusto da Cruz Motta e Aleteia Patricia Favacho de Araújo: *Performance analysis of main public cloud big data services processing brazilian government data*. Em Nesmachnow, Sergio, Harold Castro e Andrei Tchernykh (editores): *High Performance Computing*, páginas 49–61, Cham, 2021. Springer International Publishing, ISBN 978-3-030-68035-0. 152
- [160] Carvalho, Leonardo, Alba Melo e Aleteia Araujo: *A framework for executing protein sequence alignment in cloud computing services*. Em *Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho*, páginas 48–59, Porto Alegre, RS, Brasil, 2021. SBC. <https://sol.sbc.org.br/index.php/sscad/article/view/18511>. 152
- [161] Motta, Marcelo, Leonardo Carvalho e Aleteia Araujo: *Análise de desempenho de funções como serviço em nuvem privada - dataprev*. Em *Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste*, páginas 17–21, Porto Alegre, RS, Brasil, 2021. SBC. <https://sol.sbc.org.br/index.php/eradco/article/view/18418>. 152
- [162] Fernandes, Matheus, Leonardo Carvalho e Aleteia Araujo: *Framework de autenticação para faas em nuvem pública*. Em *Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste*, páginas 32–34, Porto Alegre, RS, Brasil, 2021. SBC. <https://sol.sbc.org.br/index.php/eradco/article/view/18421>. 152
- [163] Bandeira, Waliff, Leonardo Carvalho e Aleteia Araujo: *Análise do custo-benefício de funções como serviço e infraestrutura como serviço*. Em *Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste*, páginas 38–40, Porto Alegre, RS, Brasil, 2021. SBC. <https://sol.sbc.org.br/index.php/eradco/article/view/18423>. 152
- [164] Carvalho, Leonardo Rebouças de e Aleteia Patricia Favacho de Araujo: *Function-as-a-service: Desenvolvendo aplicações na próxima geração da computação em nuvem*. Em *Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste*, páginas 38–40, Porto Alegre, RS, Brasil, 2021. SBC. 152
- [165] Costa, Breno, Joao Bachiega, Leonardo Rebouças de Carvalho e Aleteia P. F. Araujo: *Orchestration in fog computing: A comprehensive survey*. ACM Comput. Surv., 55(2), janeiro 2022, ISSN 0360-0300. <https://doi.org/10.1145/3486221>. 152, 153
- [166] Bachiega Jr., Joao, Breno Costa, Leonardo R. Carvalho, Victor H. C. Oliveira, William X. Santos, Maria Clícia S. de Castro e Aleteia Araujo: *From the sky to the ground: Comparing fog computing with related distributed paradigms*. Em *Proceedings of the 12th International Conference on Cloud Computing and Services Science*

- *CLOSER*, páginas 158–169. INSTICC, SciTePress, 2022, ISBN 978-989-758-570-8. 152, 153, 154
- [167] Motta, Marcelo Augusto Da Cruz, Leonardo Reboucas De Carvalho, Michel Junio Ferreira Rosa e Aleiteia Patricia Favacho De Araujo: *Comparison of faas platform performance in private clouds*. Em *Proceedings of the 12th International Conference on Cloud Computing and Services Science - CLOSER*, páginas 109–120. INSTICC, SciTePress, 2022, ISBN 978-989-758-570-8. 152, 153, 154
- [168] Costa, Breno, João Bachiega, Leonardo Rebouças Carvalho, Michel Rosa e Aleiteia Araujo: *Monitoring fog computing: A review, taxonomy and open challenges*. *Computer Networks*, 215:109189, 2022, ISSN 1389-1286. <https://www.sciencedirect.com/science/article/pii/S1389128622002845>. 152
- [169] R. De Carvalho, Leonardo, Felipe S. Lopes, Jefferson Chaves, Marcos C. Lima, Flávio E. Gomes De Deus, Aletéia P. F. A. von Paungarthem e Flavio De Barros Vidal: *Deep-vacuity: A proposal of a machine learning platform based on high-performance computing architecture for insights on government of brazil official gazettes*. Em *Proceedings of the 18th International Conference on Web Information Systems and Technologies - WEBIST*, páginas 136–143. INSTICC, SciTePress, 2022, ISBN 978-989-758-613-2. 152
- [170] Bachiega, Joao, Breno Costa, Leonardo R. Carvalho, Michel J. F. Rosa e Aleiteia Araujo: *Computational resource allocation in fog computing: A comprehensive survey*. *ACM Comput. Surv.*, 55(14s), julho 2023, ISSN 0360-0300. <https://doi.org/10.1145/3586181>. 152
- [171] Costa, Breno, Abhik Banerjee, Prem Prakash Jayaraman, Leonardo R. Carvalho, João Bachiega e Aleiteia Araujo: *Achieving observability on fog computing with the use of open-source tools*. Em Zaslavsky, Arkady, Zhaolong Ning, Vana Kalogeraki, Dimitrios Georgakopoulos e Panos K. Chrysanthis (editores): *Mobile and Ubiquitous Systems: Computing, Networking and Services*, páginas 319–340, Cham, 2024. Springer Nature Switzerland, ISBN 978-3-031-63992-0. 152
- [172] Bachiega, João, Breno G. S. Costa, Michel Junio Ferreira Rosa, Leonardo R. Carvalho, Marcelo Antonio Marotta, Aleiteia Araujo e Rajkumar Buyya: *A cost-efficient resource allocation for fog computing with users and providers perspective*. 2024 IEEE/ACM 17th International Conference on Utility and Cloud Computing (UCC), páginas 1–8, 2024. <https://api.semanticscholar.org/CorpusID:277549747>. 152
- [173] Carvalho, Leonardo Reboucas de, Alba Cristina Alves Melo e Aleiteia Araujo: *Afmc: An alignment framework for multiple computing services and providers*. *Concurrency and Computation: Practice and Experience*, 35(18):e7595, 2022. <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.7595>. 153
- [174] Carvalho, Leonardo Rebouças de, Bruno Kamienski e Aleiteia Araujo: *Main FaaS Providers Behavior Under High Concurrency: An Evaluation with Orama Framework Distributed Architecture*. *SN Computer Science*, 5(5):541, 2024. 153, 154

- [175] Carvalho, Leonardo Rebouças de, Bruno Kamienski e Aleteia Araujo: *Empowering statistical analysis in faas environments through the orama framework*. Em Steen, Maarten van, Donald Ferguson e Claus Pahl (editores): *Cloud Computing and Services Science*, páginas 25–44, Cham, 2024. Springer Nature Switzerland, ISBN 978-3-031-68165-3. 153
- [176] Carvalho, Leonardo, Geraldo Rocha Filho e Aleteia Araujo: *Predicting faas runtime with the orama framework using machine learning*. Em *Anais do XXVI Simpósio em Sistemas Computacionais de Alto Desempenho*, páginas 49–60, Porto Alegre, RS, Brasil, 2025. SBC. <https://sol.sbc.org.br/index.php/sscad/article/view/37886>. 153, 154
- [177] Carvalho, Leonardo Rebouças de, Geraldo Pereira Rocha Filho e Aleteia Araujo: *Aplicação de machine learning à predição de tempo de execução em faas com o framework orama*. Em *Anais da XIII Escola Regional de Informática de Goiás*, páginas 384–388, Porto Alegre, RS, Brasil, 2025. SBC. <https://sol.sbc.org.br/index.php/erigo/article/view/40358>. 153

Anexo I

Guia Rápido do *framework* Orama⁺⁺



Guia Rápido de Utilização do Framework Orama++

O que é o Orama++?

O Orama++ é um *framework* voltado para avaliação e comparação de ambientes *Function-as-a-Service* (FaaS) em múltiplos provedores de nuvem, como AWS Lambda, Google Cloud Functions, Azure Functions e Alibaba Function Compute. Ele automatiza o provisionamento, execução de benchmarks, coleta de métricas, análise estatística e predição de tempo de execução e custos.

Público-alvo

O framework é indicado para pesquisadores, arquitetos de software, engenheiros de nuvem, estudantes de pós-graduação e equipes DevOps que desejam avaliar desempenho, custo e comportamento de aplicações *serverless* em diferentes provedores.

Cenário Prático

Uma equipe deseja descobrir qual provedor FaaS oferece melhor desempenho e menor custo para uma API *serverless* de cálculo matemático utilizada por milhares de usuários simultaneamente. Para isso, o Orama++ pode ser utilizado para implantar a função em AWS e GCP, executar benchmarks com diferentes níveis de concorrência e analisar os resultados. Caso a equipe deseje obter estimativas sem precisar implantar ambientes, é possível usar uma ferramenta do framework que é capaz de estimar o tempo de execução com base no código-fonte da aplicação, bem como estimar custos em diferentes configurações dos provedores.

Passo a Passo

1. Instale Docker e Docker Compose na máquina que executará o framework.
2. Clone o repositório do Orama++ (<https://github.com/unb-faas/orama>) e acesse a pasta "src".
3. Crie o arquivo ".env" com as portas e credenciais dos provedores desejados (AWS, GCP, Azure ou Alibaba). Consulte [orientações](#) no repositório do projeto.
4. Execute o comando "docker compose up -d --build" para iniciar todos os serviços do framework.
5. Execute "docker compose exec backend knex seed:run" para popular o banco inicial.
6. Acesse a interface web pelo navegador utilizando "http://[IP_DO_SERVIDOR]:3000".
7. Para realizar um *benchmark*:
 - a. Escolha o caso de uso desejado, como a API de calculadora.
 - b. Configure parâmetros do *benchmark*, como memória, concorrência, região e taxa de invocação.
 - c. Execute os testes e acompanhe os resultados coletados automaticamente pelo Orama++.
 - d. Utilize os relatórios estatísticos e o módulo preditivo para comparar desempenho e custo entre provedores.
8. Para estimar o tempo de execução a partir do código-fonte:
 - a. Acesse o menu "Code Prediction".
 - b. Escreva o código-fonte da sua função.
 - c. Selecione os níveis de concorrência e entrada (*input*).
 - d. Visualize as predições para cada provedor.
 - e. Inclua configurações dos provedores (*tiers*) e visualize as estimativas de custos.

Anexo II

Certificado de Registro de Programa de Computador - *framework* Orama⁺⁺



REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DO DESENVOLVIMENTO, INDÚSTRIA, COMÉRCIO E SERVIÇOS
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512026002022-0**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1º de janeiro subsequente à data de 30/06/2025, em conformidade com o §2º, art. 2º da Lei 9.609, de 19 de Fevereiro de 1998.

Título: Orama Framework

Data de publicação: 30/06/2025

Data de criação: 30/06/2025

Titular(es): FUNDAÇÃO UNIVERSIDADE DE BRASILIA

Autor(es): ALETÉIA PATRÍCIA FAVACHO DE ARAÚJO VON PAUMGARTTEN; LEONARDO REBOUÇAS DE CARVALHO

Linguagem: JAVA SCRIPT; PYTHON; NODEJS; OUTROS

Campo de aplicação: IF-07

Tipo de programa: IA-01

Algoritmo hash: SHA-256

Resumo digital hash: 918CA01D793E7E517263B3C966C47512374C15EFD560781FAC54FFCFD94B9746

Expedido em: 31/03/2026

Aprovado por:

ERICA GUIMARAES CORREA

Chefe da Divisão de Programas de Computador e Topografias de Circuitos Integrados

Anexo III

Certificado de Registro de Programa de Computador - *framework* Node2FaaS



REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DO DESENVOLVIMENTO, INDÚSTRIA, COMÉRCIO E SERVIÇOS
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL
DIRETORIA DE PATENTES, PROGRAMAS DE COMPUTADOR E TOPOGRAFIAS DE CIRCUITOS

Certificado de Registro de Programa de Computador

Processo Nº: **BR512026002212-6**

O Instituto Nacional da Propriedade Industrial expede o presente certificado de registro de programa de computador, válido por 50 anos a partir de 1º de janeiro subsequente à data de 01/10/2022, em conformidade com o §2º, art. 2º da Lei 9.609, de 19 de Fevereiro de 1998.

Título: Node2FaaS framework

Data de publicação: 01/10/2022

Data de criação: 01/10/2022

Titular(es): FUNDAÇÃO UNIVERSIDADE DE BRASILIA

Autor(es): ALETÉIA PATRÍCIA FAVACHO DE ARAÚJO VON PAUMGARTTEN; LEONARDO REBOUÇAS DE CARVALHO

Linguagem: NODEJS; OUTROS

Campo de aplicação: IF-07

Tipo de programa: DS-08

Algoritmo hash: SHA-256

Resumo digital hash: FFBE2D87C332F2EC510435C32FEF3D9FB35AB6162021FC9923708452ADAF0F76

Expedido em: 07/04/2026

Aprovado por:

ERICA GUIMARAES CORREA

Chefe da Divisão de Programas de Computador e Topografias de Circuitos Integrados

Anexo IV

Artigo publicado na conferência
CARLA/2021

Performance Analysis of Main Public Cloud Big Data Services Processing Brazilian Government Data

Leonardo Rebouças de Carvalho^[0000–0001–7459–281X], Marcelo Augusto da Cruz Motta^[0000–0002–4114–4735], and Aleiteia Patricia Favacho de Araújo^[0000–0003–4645–6700]

¹ Department of Computer Science - University of Brasilia, Brasilia, Brazil
leouesb@gmail.com

² motta_marcelo@hotmail.com

³ aleiteia@unb.br

Abstract. The growing amount of information generated by big data systems has driven the use of tools that facilitate their processing, such as Hadoop and its entire ecosystem. These tools can run on computational clouds whose benefits include payment on-demand, self-service, and elasticity. This article evaluates three cloud services that delivers fully-configured Hadoop ecosystems: AWS Elastic Map Reduce (EMR), Google Dataproc, and Microsoft HDInsight. This evaluation was made by measuring their performance and computational resource consumption by performing workloads using data from the Bolsa Família, a social welfare program of the Brazilian Government. The results showed that HDInsight had better runtime performance. Variations in the consumption of resources related to memory, disk activity, cost, and processing were found, providing an insight into the strategy of each provider that can be useful in the decision-making processes.

Keywords: Big Data · Cloud Computing · Hadoop · Bolsa Família Program · Google Cloud Platform · Amazon Web Services · Microsoft AZURE.

1 Introduction

The growing volume of data generated by society has produced an overwhelming flow of data [1]. The volume of digital data is expected to jump from 33 zettabytes in 2018 to 175 zettabytes in 2025 [39]. The amount of data generated by China alone by 2020 is expected to exceed 10 times the amount of grains of sand on the entire planet Earth [41]. This is not just an increase in business data volume, but an increase of several orders of magnitude, commonly generated outside of traditional enterprise applications and generally composed of unstructured or semi-structured information types in huge amounts and continuous flow. This reality has generated a demand for alternatives that can exploit these masses of

Anexo V

Artigo publicado na conferência
WSCAD/2021

A Framework for Executing Protein Sequence Alignment in Cloud Computing Services

Leonardo Reboucas de Carvalho¹, Alba Cristina Alves Melo¹, Aleteia Araujo¹

¹Campus Darcy Ribeiro, Department of Computer Science – University of Brasília
Brasília, Brazil.

leouesb@gmail.com, {alves,aleteia}@unb.br

Abstract. *Protein sequence alignment is a task of great relevance in Bioinformatics and the Hirschberg algorithm is widely used for this task. This work proposes a framework for executing sequence alignment with the Hirschberg algorithm in different cloud computing services. In experiments, our framework was used to align HIV-1 protease sequences using different instances of AWS EC2 and different configurations of AWS Lambda functions. The results show that, for this application, there is a tradeoff between the expected execution time and the cost, e.g., in most cases AWS Lambda provides the best runtime, however at a higher USD cost. In this context, it is important to have a framework that helps in deciding which approach is most appropriate.*

1. Introduction

Studies in Bioinformatics generally produce computational challenges that require large volumes of computing resources. An example of this type of challenge is protein alignment. In this task, two strings will be compared with the purpose of identifying their similarities and differences. With a focus on differences, researchers can make great discoveries about the protein in question. In this context cloud computing can collaborate with Bioinformatics.

The potential requirement for large volumes of computational resources that a sequence alignment application has signals that its processing is suitable to be sent to a cloud computing environment, since this type of environment offers the possibility of fast and dynamic allocation of resources [MELL and Grance 2011]. Cloud computing currently offers several service models and each feature of these models can result in benefits or losses, whether in runtime, cost or other aspects. It is therefore important to choose the type of cloud service best suited to the challenge to be faced.

In this paper, we propose a framework for the provision and execution of a sequence alignment application composed of multiple tasks in different service models. In the provision phase, our framework decides which model (AWS EC2 on demand or AWS Lambda) is more appropriate for a particular sequence alignment application, considering two goals (cost and execution time). The alignments are then executed concurrently in the chosen cloud model and, at the end of the execution, the results are returned to the user. Finally, the unprovision phase terminates the cloud environment.

The remainder of this paper is organized as follows. Section 2 presents the background of this work, including the fundamentals of sequence alignment and cloud computing. Section 3 describes the proposal of this work, while Section 4 discusses related

Anexo VI

Artigo publicado na conferência
ERAD-CO/2021

Análise de Desempenho de Funções como Serviço em Nuvem Privada - DATAPREV

Marcelo A. da Cruz Motta¹, Leonardo Reboucas de Carvalho¹, Aleteia P. F. de Araujo¹

¹Departamento de Ciência da Computação
Universidade de Brasília (UNB) – Brasília, DF – Brazil

marcelo.motta@aluno.unb.br, leouesb@gmail.com, aleteia@unb.br

Abstract. *Function as a Service (FaaS) is a cloud service model that allows users to develop, run, and manage application functionality without worrying about the complexity of managing the infrastructure. Given the remarkable growth in its offer by public cloud providers and considering its use in private cloud environments, this paper presents a performance analysis of current open source technologies related to the Function as a Service (Faas) model, using the environment of the private cloud of the Empresa de Tecnologia e Informações da Previdência Social - DATAPREV.*

Resumo. *Função como serviço (FaaS) é um modelo de serviço em nuvem que permite aos usuários desenvolver, executar e gerenciar funcionalidades de aplicativos sem se preocupar com a complexidade da gestão da infraestrutura. Diante do notável crescimento na sua oferta pelos provedores públicos de nuvem e considerando sua utilização em ambientes de nuvem privada, este trabalho apresenta uma análise de desempenho das atuais tecnologias de código aberto relacionadas ao modelo de Function as a Service (Faas), utilizando o ambiente de nuvem privada da Empresa de Tecnologia e Informações da Previdência Social - DATAPREV.*

1. Introdução

Dentre as várias formas de oferta de serviços em nuvem atualmente destaca-se o *Function as a Service* (FaaS) [SPOIALA 2017], também conhecido como *Serverless*, no qual o cliente submete um trecho de código ao provedor, que irá garantir seu processamento independentemente do número de acionamentos. Em 2014 a AWS (Amazon Web Services) lançou o AWS Lambda [Amazon Web Services 2021], que é o modelo de serviço de nuvem denominado Function-as-a-Service [Schleier-Smith et al. 2021]. Desde então, diversos outros provedores de nuvem pública passaram a oferecer em seus catálogos serviços orientados a esse novo modelo, por exemplo o *Google Cloud Platform* também oferece um serviço de FaaS, o *Google Cloud Function* [Google 2021], a Microsoft o *Azure Functions* [Microsoft 2021], outros provedores como Alibaba, Oracle e IBM também possuem suas opções de FaaS.

Diante do leque de novas possibilidades para esse modelo, o interesse sobre ele cresceu consideravelmente, tem sido tão representativo que existe a perspectiva de que, em pouco tempo, torne-se a forma de adoção de nuvem predominante dentre os modelos disponíveis atualmente [Schleier-Smith et al. 2021]. Isso ocorre porque além de simplificar consideravelmente a execução de software na nuvem, suprimindo a necessidade de

Anexo VII

Artigo publicado na conferência
ERAD-CO/2021

Framework de Autenticação para FaaS em Nuvem Pública

Matheus de Sousa Lemos Fernandes¹, Leonardo Rebouças de Carvalho², and
Aleteia Patricia Favacho de Araujo³

^{1,2,3}Departamento de Ciência da Computação - Universidade de Brasília (UnB) - Brasília
- DF - Brazil

1

Abstract. *Access control is an important functionality on internet applications. On managed cloud providers, however, access control security is often simplified, and user access is often done with simple usernames and passwords, which does not provide enough security for cloud based applications. This paper presents a framework developed to authenticate users on cloud providers, in order to access FaaS services. To ensure user authenticity, this framework uses OpenID Connect, a modern authentication technology.*

Resumo. *Controle de acesso é uma importante funcionalidade em aplicações na internet. Em plataformas de nuvem, entretanto, a segurança no controle de acesso acaba sendo simplificada, e o acesso é frequentemente realizado utilizando os comuns usuário e senha, o que muitas vezes não oferece a segurança necessária em um ambiente de nuvem pública. Este trabalho apresenta um framework desenvolvido para autenticar usuários junto a um provedor de nuvem, de modo a utilizar serviços de FaaS. Para garantir a autenticidade do usuário, utilizou-se o OpenID Connect, uma moderna tecnologia de autenticação.*

1. Introdução

Função como um Serviço (*Function as a Service*) é um paradigma de computação que envolve a definição de unidades atômicas de código, chamadas funções, na qual cada função deve realizar apenas uma tarefa. Essas funções podem ser chamadas por diversos meios (chamadas de *API*, *gatilhos*, *logs*), e seus resultados podem ser retornados ao usuário ou conectadas a outros serviços de nuvem [AWS 2021].

Ambientes de *FaaS*, entretanto, não possuem mecanismos de autenticação próprios, e dependem dos mecanismos de acesso disponíveis na própria plataforma. O uso de um serviço dentro do próprio provedor pode muitas vezes não ser estratégico para o usuário do serviço. O usuário muitas vezes precisa manter os dados de seus usuários em seus próprios ambientes privados.

Nesse caso, é vantajoso utilizar uma ferramenta de autenticação própria, utilizando padrões abertos e previamente auditados.

Assim sendo, esse *framework* apresenta um mecanismo de autenticação utilizando a técnica de *Single-Sign-On (SSO)* [Indu et al. 2018], delegando a um provedor de identidade especializado a tarefa de confirmar a identidade dos usuários da plataforma.

Anexo VIII

Artigo publicado na conferência
ERAD-CO/2021

Análise do Custo-benefício de Funções como Serviço e Infraestrutura como Serviço

Waliff Cordeiro Bandeira¹, Leonardo Rebouças de Carvalho²,
Aleteia Patricia Favacho de Araujo³

¹Departamento de Ciência da Computação - Universidade de Brasília (UnB) - Brasília - DF - Brazil

Abstract. *Function as a Service (FaaS) offers high performance in its applications to be used with high availability and payment based on demand. These features, however, come with a high added monetary value, which makes it not feasible to use this service full-time for various types of applications and contexts. To better understand this inflection point, it is necessary to test and compare, in a practical way, what is the acceptable threshold to choose a service among other options, before starting the application development. For this, Infrastructure as a Service (IaaS) will be used, which is a service that provides computing resources, IaaS was selected in order to compare and analyze the cost-benefit at the expense of FaaS.*

Resumo. *O Function as a Service (FaaS) oferece alto desempenho em suas aplicações para ser utilizado com alta disponibilidade e pagamento baseado na demanda. Esses recursos, entretanto, vêm com um alto valor monetário agregado, o que faz com que não seja viável a utilização desse serviço em tempo integral para vários tipos de aplicações e contextos. Para melhor entender esse ponto de inflexão, faz-se necessário testar e comparar, de forma prática, qual o limiar aceitável para escolher um serviço dentre outras opções antes de iniciar o desenvolvimento da aplicação. Para isso será utilizada a Infraestrutura como Serviço (IaaS), que é um serviço de disponibilização de recursos computacionais, o IaaS foi escolhido a fim de comparar e analisar o custo-benefício em detrimento do FaaS.*

1. Introdução

Atualmente, o *Function as a Service* (FaaS) - em português, Função como Serviço - está presente na realidade de diversos desenvolvedores. Essa tecnologia permite a criação de funções que retornam o resultado esperado em um curto espaço de tempo, em uma interface simples e de fácil acesso, disponível na maior parte dos grandes provedores de serviço em nuvem como AWS, Azure e GCP. Logo, o FaaS tem sido visto como um serviço de extremo valor para diversas aplicações que possuem o tempo como um fator decisivo em seu desempenho.

Segundo o artigo [Sewak and Singh 2018], a utilização de FaaS cresce em um ritmo de 32,7% ao ano, o que evidencia a necessidade de um melhor entendimento sobre o custo da utilização desse serviço, levando-se em consideração seu alto desempenho e disponibilidade mas também ao considerar seu alto custo agregado. Para essa análise comparativa, é importante considerar outros serviços que oferecem a capacidade de processamento em nuvem para que se possa compreender o custo-benefício desses serviços oferecidos pelos

Anexo IX

Artigo publicado na conferência
ERAD-CO/2021

Capítulo

1

Function-as-a-Service: Desenvolvendo Aplicações na Próxima Geração da Computação em Nuvem

Leonardo Rebouças de Carvalho and Aleteia Patricia Favacho de Araujo

Abstract

Cloud Computing has sparked a revolution by enabling quick access to significant contingents of computing resources. The infrastructure context has undergone major changes with the virtualization of various resources that were traditionally handled physically. However, the approach to software development for the cloud has not changed substantially from traditional computing. With the introduction of Function-as-a-Service (FaaS) this paradigm tends to undergo major transformations. A new way to develop cloud-oriented applications that draws the full power of this platform must culminate in the next generation of Cloud Computing.

Resumo

A Computação em Nuvem provocou uma revolução ao permitir acesso rápido a contingentes significativos de recursos computacionais. O contexto de infraestrutura sofreu grandes alterações com a virtualização de diversos recursos que tradicionalmente eram tratados fisicamente. Entretanto, a abordagem do desenvolvimento de software para a nuvem não mudou substancialmente em relação à computação tradicional. Com a introdução de Function-as-a-Service (FaaS) esse paradigma tende a sofrer grandes transformações. Uma nova forma de desenvolver aplicações orientadas à nuvem que extraia toda a capacidade dessa plataforma deve culminar na próxima geração da computação em nuvem.

1.1. Introdução

A receita destinada para computação em nuvem por empresas mundo afora está prevista para atingir 45% até 2026 [1]. Considerando que em 2021 esses gastos representavam apenas 17% da totalidade dos gastos corporativos é possível observar a tendência de um crescimento acentuado desse segmento da computação [1]. Não é para menos, afinal,

Anexo X

Artigo publicado no periódico ACM
Computing Surveys/2022

Orchestration in Fog Computing: A Comprehensive Survey

BRENO COSTA, JOAO BACHIEGA JR., LEONARDO REBOUÇAS DE CARVALHO, and ALETEIA P. F. ARAUJO, University of Brasilia, Brazil

Fog computing is a paradigm that brings computational resources and services to the network edge in the vicinity of user devices, lowering latency and connecting with cloud computing resources. Unlike cloud computing, fog resources are based on constrained and heterogeneous nodes whose connectivity can be unstable. In this complex scenario, there is a need to define and implement orchestration processes to ensure that applications and services can be provided, considering the settled agreements. Although some publications have dealt with orchestration in fog computing, there are still some diverse definitions and functional intersection with other areas, such as resource management and monitoring. This article presents a systematic review of the literature with focus on orchestration in fog computing. A generic architecture of fog orchestration is presented, created from the consolidation of the analyzed proposals, bringing to light the essential functionalities addressed in the literature. This work also highlights the main challenges and open research questions.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Cloud computing**; **Distributed architectures**;

Additional Key Words and Phrases: Fog computing, orchestration, monitoring, resource management

ACM Reference format:

Breno Costa, Joao Bachiega Jr., Leonardo Rebouças de Carvalho, and Aleteia P. F. Araujo. 2022. Orchestration in Fog Computing: A Comprehensive Survey. *ACM Comput. Surv.* 55, 2, Article 29 (January 2022), 34 pages. <https://doi.org/10.1145/3486221>

1 INTRODUCTION

Cloud computing is already a mature paradigm that has been in place since 2006 [83]. It offers virtualized resources, created upon a huge shared infrastructure, that are consumed by the customers on a pay-per-use business model [139] and with a variety of cost options [19, 62]. Cloud computing is based on dozens of large datacenters, distributed around the world, that are placed in the center of the network and accessed by the Internet. Cloud computing also provides automatic scalability to its services.

Cloud computing datacenters are composed of thousands of resource-rich homogeneous physical servers that are interconnected by a redundant and stable network [157]. To optimize the infrastructure in use and comply with service and quality agreements made with the customers, a resource orchestration framework is in place.

Authors' address: B Costa, J. Bachiega Jr., L. R. de Carvalho, and A. P. F. Araujo, University of Brasilia, Department of Computer Science, Campus Darcy Ribeiro, Asa Norte, Brasilia, DF, 70910-900, Brazil; emails: {brenogcosta, joao.bachiega.jr, leouesb}@gmail.com, aleteia@unb.br.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

0360-0300/2022/01-ART29 \$15.00

<https://doi.org/10.1145/3486221>

Anexo XI

**Artigo publicado na conferência
CLOSER/2022**

From the Sky to the Ground: Comparing Fog Computing with Related Distributed Paradigms

Joao Bachiega Jr.¹, Breno Costa¹, Leonardo R. Carvalho¹, Victor H. C. Oliveira¹, William X. Santos¹, Maria Clicia S. de Castro² and Aleteia Araujo¹

¹Department of Computer Science, University of Brasília (UnB), Brazil

²Department of Informatics and Computer Science, State University of Rio de Janeiro (UERJ), Rio de Janeiro, Brazil

Keywords: Edge Computing, Fog Computing, Cloud Computing, Sky Computing.

Abstract: Fog computing is a paradigm that enables provisioning resources and services at the network edge, closer to end devices, complementing cloud computing and recently it's being embraced by sky computing. Beyond this computational paradigm, there are many other related technologies which also make use of distributed computing to improve the quality of service delivered to the end-users. The main contribution of this paper is to present a comparison between fog computing and nine other relevant related paradigms, namely Sky Computing, Cloud Computing, Edge Computing, Mobile Edge Computing, Mobile Cloud Computing, Mobile Ad hoc Cloud Computing, Mist Computing, Cloudlet Computing, and Dew Computing, highlighting the similarities and differences between them based on the main fog characteristics. A graphical characterization for each paradigm, highlighting its computational power, communication type and position in a three-layers architecture (Cloud-Fog-IoT) and some relevant challenges in this area are also presented.

1 INTRODUCTION

Over the years, computing paradigms have been evolving. Since the beginning of the modern computer era until about 1985, computers were large and expensive. Moreover, there was no connectivity between them and they operated independently of each other. With the development of powerful microprocessors and high-speed computer networks, it was possible to change this scenario (Tanenbaum and Van Steen, 2007).

After that, computing technologies evolved from distributed, parallel, clustering, peer-to-peer (P2P), and grid until the cloud computing. There is no doubt that cloud computing is one of the most important computational paradigms, playing an essential role in almost every aspect of our everyday life. It brought the impression of infinite computational resources (e.g. storage, processing) provided in a pay-per-use basis, allowing fast scalability at a reasonable cost. Nevertheless, as a centralized computing model, computation happens in the cloud data centers, located in specific places around the world, and far from the end-users (Mukherjee et al., 2018).

As in a cycle, nowadays is the time when dis-

tributed computing paradigms are back in the spotlight, aiming to overcome the limitations of centralized cloud computing processes. Internet of Things (IoT) development demands this (Yousefpour et al., 2019). Thus, new concepts were born ranging from *Sky Computing* (Keahey et al., 2009), which is a level above cloud computing and uses resources and services from several providers simultaneously, until *Fog Computing* (Bonomi et al., 2012), which is a cloud close to the ground, addressing several issues of connected devices, like high-bandwidth, geographical dispersion, and low latency needs. Fog is both complementary to, and an extension of, traditional cloud-based models.

Although fog computing is a recent paradigm, in the last 3 years it was the subject of approximately 20% of all publications related to computational paradigms, including Sky Computing, Edge Computing, Mobile Edge Computing, Mobile Cloud Computing, Mobile Ad hoc Cloud Computing, Mist Computing, Cloudlet Computing and Dew Computing.

The main contribution of this paper is to present a comparison between fog computing and other related paradigms, highlighting similarities and differences

Anexo XII

**Artigo publicado na conferência
CLOSER/2022**

Comparison of FaaS Platform Performance in Private Clouds

Marcelo Augusto Da Cruz Motta^a, Leonardo Reboucas De Carvalho^b,
Michel Junio Ferreira Rosa^c and Aleteia Patricia Favacho De Araujo^d

Department of Computing Science, University of Brasilia, Brasilia, Brazil

Keywords: FaaS, Function-as-a-Service, Private Cloud, OpenWhisk, Fission, OpenFaaS.

Abstract: Since its appearance in mid-2014, there has been notable growth in the adoption of cloud services via the Function-as-a-Service (FaaS) model, with several public cloud providers offering this model in their catalog. Various papers have mostly evaluated the performance of these models in public cloud environments. However, the implementation of this model in a private cloud environment has not been explored enough by the academic community. This article presents a two-level factorial design-based assessment of the main open-source operating platforms (Such as OpenWhisk, Fission, and OpenFaaS) currently available to serve as a comparative tool that can be used in decision-making processes. Results showed that regarding the stress test the OpenWhisk platform has greater reliability. On the other hand, regarding the processing of the Matrix, Factors, and Filesystem functions, Fission remains at similar behavior at all concurrency levels.

1 INTRODUCTION

Since the launch of the Function-as-a-service (FaaS) (Schleier-Smith et al., 2021) cloud service model in 2014 by AWS (Amazon Web Services) as AWS Lambda (Amazon Web Services, 2021), several providers have also added FaaS products to their catalogs. Initially used in tasks of low complexity, the model is quite lenient and soon came to expand its range of applicability. From this point on, the growth of this cloud service model has been quite representative, and it has been appointed as a candidate to become the predominant model adopted by users.

In this context, some academic papers such as (Carvalho and Araújo, 2019), (García López et al., 2018), (Malawski et al., 2020), have evaluated the performance and behavior of this model, using different types of metrics. However, a greater interest in the literature for the implementation of this service model in public clouds have been noticed, even considering the possibility of using FaaS technologies in private clouds. Given this scenario, this paper presents a performance analysis based on a factorial design considering the main open-source FaaS platforms currently available, which are: Fission (Fission,

2021), OpenFaaS (OpenFaaS, 2021) and OpenWhisk (Apache OpenWhisk, 2021) in order to identify the best performing and lowest cost tool, aiming its use in real world operations in private clouds.

An equivalent environment containing these tools were provisioned in Dataprev's ¹ private cloud infrastructure to allow a comprehensive analysis on the quality and on the efficiency. To achieve this goal, corresponding workloads were planned and executed on each platform using the Apache JMeter (Halili, 2008) tool and some metrics were collected from them, allowing comparative analysis under different conditions. Results showed that the OpenFaaS platform obtained a more consistent response level in several tests regarding the network-bound function. When processing the Matrix function, The OpenWhisk platform reached a more consistent level at the beginning, being outperformed by Fission in large activations.

This paper is divided into six sections, this being an introductory section. Section 2 presents the concepts regarding Function-as-a-Service and the main platforms of this model. In Section 3, related papers are presented. Section 4 presents the methodology used in this paper, while Section 5 presents the results. Finally, Section 6 presents a conclusion and future work.

¹Social Security Data Processing Company, a Brazilian state-owned company. <https://www.dataprev.gov.br/>

^a <https://orcid.org/0000-0002-4114-4735>

^b <https://orcid.org/0000-0001-7459-281X>

^c <https://orcid.org/0000-0002-0860-1834>

^d <https://orcid.org/0000-0003-4645-6700>

Anexo XIII

**Artigo publicado na conferência
CLOSER/2022**

Orama: A Benchmark Framework for Function-as-a-Service

Leonardo Reboucas De Carvalho^a and Aleteia Patricia Favacho Araujo^b

Department of Computing Science, University of Brasília, Brasília, Brazil

Keywords: Cloud, FaaS, Benchmark, AWS, GCP, Factorial Design, T-test, Orama Framework.

Abstract: The prominent Function-as-a-Service (FaaS) cloud service model has positioned itself as an alternative for solving several problems, and, interest in cloud-oriented architectural solutions that use FaaS has therefore grown rapidly. Consequently, the importance of knowing the behavior of FaaS-based architectures under different concurrency scenarios has also become significant, especially in implementation decision-making processes. In this work, the Orama framework is proposed, which helps in the execution of benchmarks in FaaS-based environments, orchestrating the deployment of pre-built architectures, as well as the execution of tests and statistical analysis. Experiments were carried out with architectures containing multiple cloud services in conjunction with FaaS in two public cloud providers (AWS and GCP). The results were analyzed using factorial design and t-test and showed that the use cases running on AWS obtained better results in runtime compared to their counterparts on GCP, but showed considerable error rates in competition situations. It is worth mentioning that the Orama framework was used from in the automated provisioning of use cases, execution of benchmarks, analysis of results and deprovisioning of the environment, supporting the entire process.

1 INTRODUCTION

Studies indicate that Function-as-a-Service (FaaS) (Malawski et al., 2020), also called Serverless (Nupponen and Taibi, 2020) or Backend-as-a-Service (Schleier-Smith et al., 2021), will become the main computing paradigm of the Cloud Era (Schleier-Smith et al., 2021). All this success can be explained, in part, by the great ease in using this service model, as well as in the automatic and transparent elasticity of delivery.

The growing adoption of FaaS-based architectures has increased concern about performance aspects related to environments deployed by this type of service. Some works such as FaaSdom (Maissen et al., 2020) and Sebs (Copik et al., 2021) for example, have investigated the behavior of FaaS services, mainly exploring the characteristics of programming languages and providers' strategies.

However, productive applications often involve more than one cloud service. In this context, it is important to know the behavior of this type of application when faced with different levels of concurrency. For this reason, this work proposes the Orama Frame-

work to provide assistance in the execution of benchmarks in FaaS-oriented solutions, from automated deployment, to test execution and statistical analysis.

The Orama framework has built-in FaaS architectures ready to be deployed in cloud environments and then submitted to user-customized test batteries. When the tests are finished, depending on the configuration, the user will be able to access a 2^k factorial design (Jain, 1991), where k is the number of factors analyzed.

Experiments were performed to validate the framework on two public cloud providers (AWS and GCP). For the tests, six Orama built-in use cases were applied, which enabled exploration at various levels from simple activation of FaaS functions to integration with database services and object storage.

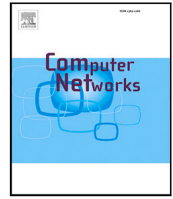
This article is divided into seven parts, the first being an introduction. Section 2 presents the theoretical foundation that supports the work, while Section 3 describes the proposal, that is, the Orama Framework. Section 4 presents the related works, while Section 5 describes the methodology used in the experiments carried out to validate the framework. Section 6 shows the results obtained and finally, Section 7 presents the conclusions and future work.

^a <https://orcid.org/0000-0001-7459-281X>

^b <https://orcid.org/0000-0003-4645-6700>

Anexo XIV

Artigo publicado no periódico
Computer Networks/2022



Review article

Monitoring fog computing: A review, taxonomy and open challenges

Breno Costa^{*}, João Bachiega Jr., Leonardo Rebouças Carvalho, Michel Rosa, Aleteia Araujo

Department of Computer Science - University of Brasília, Brazil

ARTICLE INFO

Keywords:
Monitoring
Orchestration
Fog computing
Taxonomy
Fog monitoring

ABSTRACT

Fog computing is a distributed paradigm that provides computational resources in the users' vicinity. Fog orchestration is a set of functionalities that coordinate the dynamic infrastructure and manage the services to guarantee the Service Level Agreements. Monitoring is an orchestration functionality of prime importance. It is the basis for resource management actions, collecting status of resource and service and delivering updated data to the orchestrator. There are several cloud monitoring solutions and tools, but none of them comply with fog characteristics and challenges. Fog monitoring solutions are scarce, and they may not be prepared to compose an orchestration service. This paper updates the knowledge base about fog monitoring, assessing recent subjects in this context like observability, data standardization and instrumentation domains. We propose a novel taxonomy of fog monitoring solutions, supported by a systematic review of the literature. Fog monitoring proposals are analyzed and categorized by this new taxonomy, offering researchers a comprehensive overview. This work also highlights the main challenges and open research questions.

1. Introduction

Fog computing is a computational paradigm that complements cloud computing, providing computational resources on the network edge, closer to the users. As a distributed infrastructure, fog computing must deal with heterogeneity of network links and processing capacity of its composing nodes [1]. These characteristics bring complexity to fog management, and it is addressed by the orchestration of services and resources. Orchestration is a management function, composed of several complementary functionalities. It is responsible for dealing with infrastructure dynamicity, for taking timely actions and for assuring that Service Level Agreements (SLAs) are respected [2]. There are several proposals of fog service orchestration in the literature, although most of them, only conceptual.

Monitoring is a functionality of prime importance and it is crucial to properly orchestrate fog services [3]. It collects updated status information about fog nodes and communication links and send them to the orchestrator. With an updated view of fog infrastructure and service execution, the orchestrator can take proper actions to guarantee the SLAs, e.g., offloading a service to a resource richer node and optimizing service placement according to historic data about node failures [4]. Besides the heterogeneity of nodes being monitored, there are other related concerns about frequency, topology, and communication model. There is a trade-off between the frequency of information updates and the overhead to the nodes and to the orchestrator related to generating,

transmitting and processing status data. In such a dynamic scenario, adaptability of monitoring parameters can play an important role. In our previous work [4], we did a systematic literature review of fog service orchestration and analyzed 50 proposals. Most of them (40 out of 50) highlighted monitoring as a relevant process, but they frequently assumed that a fog monitoring solution would be available to deliver the information they needed, without presenting either implementation methods or insightful information on the subject.

Monitoring is not only about reporting availability, i.e. the capacity to answer the question of whether a node or a service is online and working properly. It is also about the capacity to explain why a node or service stopped working properly. The former is achieved by monitoring metrics, e.g. service response time. The latter is achieved by monitoring logs, i.e. unstructured strings of text, and traces, i.e. records of requests made by an user in a service. Metrics, logs and traces form what is called Instrumentation Domains of monitoring [5]. Different instrumentation domains can be used simultaneously by a monitoring solution to get different perspectives of a service. In such a scenario, there would be more capacity for decision-making on the server-side, but at the cost of increasing the complexity of monitoring, since their specific characteristics (e.g. life-cycle, data volume) would be managed accordingly. Another emergent concept that is being applied to monitoring microservices is Observability. It is referenced as a superset of monitoring that uses data analytics techniques on the collected

^{*} Corresponding author.

E-mail addresses: brenogscosta@gmail.com (B. Costa), joao.bachiega.jr@gmail.com (J. Bachiega Jr.), leouesb@gmail.com (L.R. Carvalho), micheljunioferreira@gmail.com (M. Rosa), aleteia@unb.br (A. Araujo).

<https://doi.org/10.1016/j.comnet.2022.109189>

Received 5 April 2022; Received in revised form 16 June 2022; Accepted 13 July 2022

Available online 19 July 2022

1389-1286/© 2022 Elsevier B.V. All rights reserved.

Anexo XV

**Artigo publicado na conferência
WEBIST/2022**

Deep-vacuity: A Proposal of a Machine Learning Platform based on High-performance Computing Architecture for Insights on Government of Brazil Official Gazettes

Leonardo R. De Carvalho¹^a, Felipe S. Lopes¹^b, Jefferson Chaves¹^c, Marcos C. Lima²^d, Flávio E. Gomes De Deus³^e, Aletéia P. F. A. von Paungarthem¹^f and Flavio De Barros Vidal¹^g

¹Department of Computer Science, University of Brasilia, Brasilia, DF, Brazil

²Department of Federal Police, Brasilia, DF, Brazil

³Department of Electrical Engineering, University of Brasilia, Brasilia, DF, Brazil

Keywords: Machine Learning, High Performance Computing, Government of Brazil, Official Gazette.

Abstract: Brazil publishes region information, public tenders for the hire of civil servants, and also government contracts with companies in its Official Gazettes. All this volume of information can contain interesting relationships that reveal unique characteristics of the government, such as the effectiveness of public policies and even the existence of illegal schemes. Establishing these relationships is not a trivial task and requires great effort. Therefore, this work proposes the Deep Vacuity platform, which, by using a High-Performance Computing architecture along with Machine Learning techniques, can collect, depurate, consolidate and analyze the data, offering a friendly interface for decision-makers.

1 INTRODUCTION

Information is power (Freund, 1982). Even in the 1980s, a researcher had already noted this fact that today seems so obvious. However, only if it is transformed into knowledge can information be effectively transformed into action (Connolly and Matarazzo, 1998). Brazil is one of the major countries in Latin America. With a population of around 210 million people, as described in (Instituto Brasileiro de Geografia e Estatística, 2020), spread across its 5,570 cities, this country has a governmental structure full of information.

Considering that the government should be able to guide the direction of a country, it is extremely important that the actions of this government are both widely publicized and also understood at a deeper level. In other words, it is necessary to understand

the repercussions of the acts, as well as the interrelationships, be they occasional, accidental, intentional, strategic or even fraudulent. Brazil presents an exciting challenge of extracting knowledge about the range of information that the Brazilian government publishes in its state press structure. Each municipal, state and federal entity has a different way of publicizing government acts. In this context, information technology can be made available to all citizens. Using several high-performance computer processing techniques, coupled with the execution structures of models based on machine learning, this work proposes a platform architecture for the collection and processing of information published in the various Brazilian official gazettes, whether in the municipal, state or federal field. This processing uses artificial intelligence models designed to explore specific concepts according to the objectives of each usage profile, with the platform itself being agnostic in this respect.

The manuscript is organized as follows: In Sections 2 and 3 information about the official gazettes in Brazil is presented, followed by a detailed compilation of related works on fraudulent collusion in public works contracts in official texts. Section 4 explains our proposed architecture. Section 5 describes the preliminary results and a discussion about the pro-

^a <https://orcid.org/0000-0001-7459-281X>

^b <https://orcid.org/0000-0002-6568-4541>

^c <https://orcid.org/0000-0001-8459-5248>

^d <https://orcid.org/0000-0003-3565-9924>

^e <https://orcid.org/0000-0001-7953-6227>

^f <https://orcid.org/0000-0003-4645-6700>

^g <https://orcid.org/0000-0002-6317-218X>

Anexo XVI

Artigo publicado no periódico
Wiley/CCPE/2022

SPECIAL ISSUE PAPER

WILEY

AFMC: An alignment framework for multiple computing services and providers

Leonardo Reboucas de Carvalho¹ | Alba Cristina Alves Melo | Aleteia Araujo

Department of Computer Science, University of Brasilia, Campus Darcy Ribeiro, Brasilia, Federal District, Brazil

Correspondence

Leonardo Reboucas de Carvalho, Computer Science Department, University of Brasilia, Brasilia, Federal District, Brazil.
Email: leouesb@gmail.com

Funding information

CNPq/AWS, Grant/Award Number: 440014/2020-4

Summary

The Hirschberg algorithm is commonly used for protein sequence alignment, which is a very important task in bioinformatics. This article presents the AFMC framework for using the Hirschberg method to perform sequence alignment in multiple cloud computing services of different models, such as Infrastructure-as-a-Service and Function-as-a-Service (FaaS). Experiments were carried out in which several instances of AWS EC2, Azure VMs and Google Compute Engine as well as varied configurations of AWS Lambda, Azure Function, and Google Cloud Function were used to pairwise align COVID-19 spike proteins. The services were submitted to different levels of simultaneity to align the genetic sequences. The findings reveal that there is a tradeoff between predicted execution time and cost for this application, for example, FaaS-oriented cloud service models generally took less time to process the workloads. On the other hand, it was observed that, as the level of concurrence increased, there was a marked augmentation in cost. In this context, a framework that provides multi cloud solutions for bioinformatics such as AFMC is essential.

KEYWORDS

AFMC, bioinformatics, cloud computing, Function-as-a-Service (FaaS), Hirschberg algorithm, Infrastructure-as-a-Service (IaaS), sequence alignment framework

1 | INTRODUCTION

Bioinformatics research frequently generates computational issues that need a considerable amount of processing power. Protein alignment is an example of this, requiring a great amount of computing power to compare two strings to determine their similarities and differences. Researchers can make significant discoveries about the protein at issue by focusing on differences. In this scenario, cloud computing and bioinformatics can work together.

Since a sequence alignment application may require huge amounts of computational resources, its processing should be routed to a cloud computing environment, as this sort of environment allows for quick and dynamic resource allocation.¹ Cloud computing now offers a wide variety of service models, any of which might result in gains or losses in terms of runtime, cost, or other factors. As a result, it is critical to select the cloud service that is most suited to the problem at hand.

In this article, the AFMC framework is proposed for provisioning, and executing a sequence alignment application that is made up of several tasks which are delivered via distinct service models. In the provision phase, this framework considers two objectives separately to determine which cloud model is best for a specific sequence alignment application (less cost or less execution time). The alignments are then run in parallel in the cloud service of choice, with the results being given to the user at the conclusion. Finally, the cloud environment is terminated during the deprovision process. In this version of the framework, one objective is taken into account (minimize cost or minimize time) and provision in an Infrastructure-as-a-Service (IaaS)-oriented environment is selected when the user's priority is cost, and otherwise provision in Function-as-a-Service (FaaS).

Anexo XVII

Artigo publicado no periódico ACM
Computing Surveys/2023

Computational Resource Allocation in Fog Computing: A Comprehensive Survey

JOAO BACHIEGA JR., BRENO COSTA, LEONARDO R. CARVALHO, MICHEL J. F. ROSA, and ALETEIA ARAUJO, University of Brasilia

Fog computing is a paradigm that allows the provisioning of computational resources and services at the edge of the network, closer to the end devices and users, complementing cloud computing. The heterogeneity and large number of devices are challenges to obtaining optimized resource allocation in this environment. Over time, some surveys have been presented on resource management in fog computing. However, they now lack a broader and deeper view about this subject, considering the recent publications. This article presents a systematic literature review with a focus on resource allocation for fog computing, and in a more comprehensive way than the existing works. The survey is based on 108 selected publications from 2012 to 2022. The analysis has exposed their main techniques, metrics used, evaluation tools, virtualization methods, architecture, and domains where the proposed solutions were applied. The results show an updated and comprehensive view about resource allocation in fog computing. The main challenges and open research questions are discussed, and a new fog computing resource management cycle is proposed.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computer systems organization** → **Cloud computing**; **Distributed architectures**;

Additional Key Words and Phrases: Fog computing, resource allocation, resource management, resource provisioning

ACM Reference format:

Joao Bachiega Jr., Breno Costa, Leonardo R. Carvalho, Michel J. F. Rosa, and Aleteia Araujo. 2023. Computational Resource Allocation in Fog Computing: A Comprehensive Survey. *ACM Comput. Surv.* 55, 14s, Article 336 (July 2023), 31 pages.

<https://doi.org/10.1145/3586181>

336

1 INTRODUCTION

Fog computing has emerged as a promising solution to meet the growing demand for expansion of the processing, network, and storage capacity closer to end users, thus complementing the fragility of cloud computing [33]. However, as this is an emerging paradigm, there are several open research questions, with many challenges to be overcome [178]. Among these challenges is computational resource allocation, which aims to provide, in an appropriate manner, the computational resources

The authors would like to thank CAPES, a Brazilian institution that funds research and publishing, for facilitating the access to many publications through CAPES' journal portal (<https://www.periodicos.capes.gov.br>).

Authors' address: J. Bachiega Jr., B. Costa, L. R. Carvalho, M. J. F. Rosa, and A. Araujo, University of Brasilia, Department of Computer Science, Campus Darcy Ribeiro, Asa Norte, Brasilia, DF, 70910-900, Brazil; emails: {joao.bachiega.jr, brenocosta, leouesb, micheljunioferreira}@gmail.com, aleteia@unb.br.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

0360-0300/2023/07-ART336 \$15.00

<https://doi.org/10.1145/3586181>

Anexo XVIII

Artigo publicado no periódico
SNCS/2023



Insights into the Performance of Function-as-a-Service Oriented Environments Using the Orama Framework

Leonardo Rebouças de Carvalho¹ · Aleteia Araujo¹

Received: 15 August 2022 / Accepted: 15 February 2023 / Published online: 1 April 2023
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2023

Abstract

Several problems can be solved using Function-as-a-Service (FaaS) and as a result, interest in cloud-oriented architectural solutions that employ FaaS has rapidly increased. Consequently, it is crucial to understand how FaaS-based infrastructures behave. The Orama framework, which supports the execution of benchmarks in FaaS-based environments, is presented in this work. Orama can orchestrate the execution of tests and statistical analysis, as well as the deployment of ready-made architectures. In the experiments, three public cloud providers were involved and it was possible to see the performance differences between the providers for equivalent use cases. In general, Azure delivers a longer cold start time, and as concurrency increases, its average response time increases as well. AWS and GCP obtained similar results, although AWS had a higher request failure rate than GCP. From the analysis of the results presented by Orama, it was possible to perceive that the framework is capable of acting in decision-making processes, presenting insights into the performance of FaaS environments.

Keywords Orama framework · FaaS · Benchmark · AWS · GCP · Azure

Introduction

According to studies, Function-as-a-Service (FaaS) [1], also known as Serverless [2] or Backend-as-a-Service [1], will overtake other computing paradigms to become the dominant one in the Cloud Era [1]. All of this success may be partially attributed to the simplicity with which this service model can be used, as well as to the delivery system's autonomous and visible flexibility.

Cloud computing and its various service delivery models have revolutionized the way computing infrastructure

demands have been addressed. In the past, there was a need for a large financial contribution to start technology projects with the purpose of making infrastructure viable, such as datacenters. From the popularization of the cloud, the start of these projects became faster and cheaper. Although the most popular cloud delivery models are aimed at delivering basic infrastructure, interest in intermediate models has grown, in which the provider assumes more responsibility for the environment and allows the user to focus more on what interests him, the application. In this context, cloud delivery models such as FaaS offer the possibility for the customer to focus on the logic of their application, while the infrastructure details, including elasticity, are left to the provider.

FaaS-based architectures are becoming more widely used, which has raised concerns regarding the performance of environments that use these services. Studies on the behavior of FaaS services were conducted, mostly focusing on the features of programming languages and providers' methods, include FaaSdom [3] and a Serverless Benchmark Suite for Function-as-a-Service Computing (Sebs) [4].

However, apps frequently use multiple cloud services. Knowing how this kind of program behaves in this situation with various levels of concurrency is crucial. To help with the execution of benchmarks in FaaS-oriented applications,

Leonardo Rebouças de Carvalho and Aleteia Araujo contributed equally to this work.

This article is part of the topical collection "Advances on Cloud Computing and Services Science" guest edited by Donald F. Ferguson, Claus Pahl and Maarten van Steen.

✉ Leonardo Rebouças de Carvalho
leouesb@gmail.com

Aleteia Araujo
aleteia@unb.br

¹ Department of Computer Science, University of Brasília, Campus Darcy Ribeiro, Federal District, Brasília 70910-000, Brazil

Anexo XIX

**Artigo publicado na conferência
CLOSER/2023**

FaaS Benchmarking over Orama Framework's Distributed Architecture

Leonardo Rebouças de Carvalho^a, Bruno Kamienski^b and Aleteia Araujo^c

Computer Science Department, University of Brasilia, Campus Darcy Ribeiro, Brasilia, Brazil

Keywords: Distributed Benchmark, Function-as-a-Service, Orama Framework.

Abstract: As the adoption of Function-as-a-Service-oriented solutions grows, interest in tools that enable the execution of benchmarks on these environments increases. The Orama framework stands out in this context by offering a highly configurable and scalable solution to aid in provisioning, running benchmarks and comparative and statistical analysis of results. In this work, a distributed architecture for the Orama framework is presented, through which it is possible to run benchmarks with high levels of concurrency, as well as with bursts of geographically dispersed requests, just as in real environments. The results of the experiment showed that the proposed architecture was able to divide the loads between the distributed workers and able to consolidate properly in the return of the results. In addition, it was possible to observe specific characteristics of the providers involved in the experiment, such as the excellent performance of Alibaba Function, whose average execution time was the lowest of the tests and free of failures. Google Cloud Function and AWS Lambda recorded intermediate results for average execution time and recorded failures. Finally, Azure Function had the worst results in average execution time and cold start.

1 INTRODUCTION

Worldwide end-user spending on public cloud services is forecast to grow 20.7% to total \$591.8 billion in 2023, up from \$490.3 billion in 2022 (Gartner, 2022). All this perspective of growth shows the solidity of this area of computing. Although the main cloud models, such as Infrastructure-as-a-Service (IaaS) (MELL and Grance, 2011) and Platform-as-a-Service (PaaS) still lead the focus of investment, other models that integrate the archetype of Everything-as-a-Service (XaaS) (DUAN et al., 2015) also indicate growth. In this context, serverless-based cloud computing models such as Function-as-a-Service (FaaS) (Schleier-Smith et al., 2021) have been predicted as the main programming paradigms of the next generation of the cloud.

In this scenario, there has been a growing interest in evaluations and benchmark tools that analyze the deliveries of providers, such as Sebs (Copik et al., 2021), PanOpticon (Somu et al., 2020), FaaS-Dom (Maissen et al., 2020), BeFaaS (Grambow et al., 2021) and Orama framework (Carvalho and Araujo, 2022). However, given the wide range of possibili-

ties for adoptable strategies by providers, as well as the different regions in which these services can be implemented, this type of study needs to be increasingly dynamic and adjustable to real-world situations. Therefore, this work presents a distributed architecture for the Orama framework (Carvalho and Araujo, 2022) in which it is possible to run benchmarks in FaaS environments, exploring a wider range of scenarios than was possible in the standalone version. Through the master/workers architecture, it is possible to divide workloads between several instances, including geographically dispersed ones, allowing both the expansion of assessable levels of concurrency, as well as expanding the capacity to represent a reality of distributed demand.

In experiments, it was verified that the distributed architecture allows high levels of requests in comparison to the standalone version. In addition, it was found that the performance calculated between the scenario whose workers were in the same region and the scenario in which the workers were geographically spread maintained equivalent results, demonstrating that the separation of workers, in the proposed architecture, does not affect the analysis of the results. In addition, it was possible to observe behaviors intrinsic to the services, such as the superior performance demonstrated by Alibaba Cloud Func-

^a <https://orcid.org/0000-0001-7459-281X>

^b <https://orcid.org/0000-0001-5817-8694>

^c <https://orcid.org/0000-0003-4645-6700>

Anexo XX

**Artigo publicado na conferência
CISTI/2023**

How leading public cloud providers deliver FaaS environments: a comparative study

Leonardo Rebouças de Carvalho
Department of Computer Science
University of Brasilia - UnB
Brasilia, Brazil
leouesb@gmail.com

Bruno Kamienski
Department of Computer Science
University of Brasilia - UnB
Brasilia, Brazil
brunosabreu@gmail.com

Flavio Vidal
Department of Computer Science
University of Brasilia - UnB
Brasilia, Brazil
fbvidal@unb.br

Aleteia Araujo
Department of Computer Science
University of Brasilia - UnB
Brasilia, Brazil
aleteia@unb.br

Abstract — Research indicates that Function-as-a-Service (FaaS) tends to be the main model of next cloud computing generation. In view of this, interest has grown in finding out how the FaaS environment behaves in different conditions. In this work, aided by the Orama framework, an analysis of the performance of FaaS environments in the providers AWS, GCP, Azure and Alibaba in five different geographic regions is carried out: USA (East), USA (West), Europe, Asia, and Australia. The results show the existence of performance differences according to the region for the same provider, as well as differences in the average performance of applications between providers.

Keywords - *FaaS, Orama framework, Benchmarking, Public Providers.*

I. INTRODUCTION

One of the fundamental characteristics of a cloud computing service is access to computational resources through the internet [1]. Since this universe is very broad and customers can be located anywhere on the planet, public cloud providers have invested substantially in establishing infrastructures in various locations on the planet, including in the stratosphere [2, 3]. All this effort is justified by the need to get closer to the end user of its customers, in order to make the experience of using computational resources in the cloud as fast as possible, bypassing any weathering caused by very long journeys on the network.

Among the various service models currently offered by public cloud providers, Function-as-a-Service (FaaS) [3] stands out. In this model the customer submits only a snippet of code, and the provider makes available the underlying infrastructure to perform the actual processing when needed. As well as traditional cloud models such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) Software-as-a-Service (SaaS), FaaS is also subject to implementation differences between provider zones and this can cause performance variance, depending on the geographic location of the user and in which zone he is using the computational resources.

Considering that FaaS has been projected as the main model for the next generation of cloud computing [3], it is essential to understand how the environments of this model delivered by providers behave in different situations, including between availability zones located in geographically distant [4] regions. In this scenario, this work proposes an analysis of the performance of four important FaaS public cloud providers (Amazon Web Services, Azure, Google Cloud Platform and Alibaba Cloud) in five regions spread across the globe (East and West regions of the United States, Europe, Asia and Australia). Conducting the benchmarks and analyzing the results obtained at exponential levels of concurrence, from 1 to 512 simultaneous requests, was aided by the Orama framework [4], which coordinates the provisioning, deprovisioning, test execution, comparative analysis and statistics processes through factorial design and t-test. The results indicate that GCP, Azure and Alibaba maintain equivalent results between their regions, but AWS presents differences in performance, especially regarding the error rate. Furthermore, it was verified that the degradation of the average execution time with the increase of the concurrence levels was less in Alibaba than in the other providers.

This article is divided into six parts, the first being this introduction. Section II presents the related works, Section III presents the theoretical foundation that supports the work, Section IV describes the methodology used in the experiments carried out, Section V shows the results obtained and finally, Section VI presents the conclusions and future work.

II. RELATED WORKS

This paper addresses the experiment-driven evaluation of FaaS platforms in different region under different concurrence levels using the Orama framework. The related works are discussed from the perspective of benchmarking FaaS platforms in general as well as work on serverless benchmark frameworks.

In the paper [5] the authors used a microbenchmark in order to investigate two aspects of the FaaS: the differences in observable behavior with respect to the computer/memory relation of each FaaS implementation by the providers, and the

Anexo XXI

**Artigo publicado no periódico IEEE
Access/2023**

Received 8 August 2023, accepted 12 September 2023, date of publication 9 October 2023, date of current version 16 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3322936

RESEARCH ARTICLE

FaaS-Oriented Node.js Applications in an RPC Approach Using the Node2FaaS Framework

LEONARDO REBOUÇAS DE CARVALHO^{ID} AND ALETÉIA PATRICIA FAVACHO DE ARAÚJO^{ID}

Department of Computer Science, University of Brasília, Brasília 70910-900, Brazil

Corresponding author: Leonardo Rebouças de Carvalho (leouesb@gmail.com)

ABSTRACT The Function-as-a-Service (FaaS) service model has aroused great interest since its introduction in the context of cloud computing. Although FaaS can be used to perform isolated tasks, it is in the composition of applications that this service model can promote significant performance improvements. This work presents an evolution in the Node2FaaS framework, whose objective is to assist in the conversion of originally monolithic node.js applications to work with FaaS in the Remote Procedure Call (RPC) communication mechanism. The new implementations in the Node2FaaS framework allowed new experiments to be conducted. Those showed significant gains in runtime for applications with CPU-bound, memory-bound, I/O-bound characteristics, especially for a Bioinformatics application that aims to align genetic sequences.

INDEX TERMS Code transformer, FaaS, function-as-a-service, node.js, Node2FaaS, serverless.

I. INTRODUCTION

With the aim of significantly reducing costs and time for providing infrastructure, cloud computing has gained an important place in today's society. Before this goal became a reality, a considerable investment in datacenters was essential to support the processing and storage necessary for the start and continuity of projects that demanded computational resources. Service models that encapsulate both the infrastructure and the development platform itself have gained ground. These models give the developer an interface for including source code, written in the most common programming languages. One of the models promotes event-triggered stateless execution with high parallelism. This service model is called Function-as-a-Service (FaaS) [1], but it can also be referred to as a serverless model, since the user has the impression that there is no server to worry about.

Despite the benefits that the FaaS offers, it is necessary to adjust the development processes to suit this new model. In addition, certain computational problems may suffer from an increase in execution time when replacing their technology stacks with a FaaS-oriented approach, since this technology

increases the total number of layers of the original solution. Moreover, each provider may have a particular way of offering interaction with their services, and the developer will need to become familiar with these interfaces. In this context, "FaaSification" is the process of converting a code structure into a format which is executable on a Function-as-a-Service platform [2]. Some proposals are available to perform FaaSification, such as: Zappa [3], PyWren [4], Lithops [5], ToLambda [6], DaF [7], M2FaaS [8], however none of them has the characteristics that Node2FaaS does, in particular the adoption of a multicloud orchestrator, among other characteristics, as will be presented in Section V.

This work presents advances in the Node2FaaS framework that allowed its use on a real application and a deeper evaluation of the benefits of the framework approach, including a cost analysis.

In the preliminary version [9], a RPC approach was introduced with the aim of incorporating the benefits of FaaS, such as elasticity and underlying infrastructure managed by the provider and cost-efficiency. By transferring responsibility for scaling as well as infrastructure management to the provider, developers can stay focused on addressing the software's core purpose. Furthermore, as charging is based on activations, the cost is adjusted to real demand. Even in the initial version, use cases were developed to

The associate editor coordinating the review of this manuscript and approving it for publication was Somchart Fugkeaw^{ID}.

Anexo XXII

Artigo publicado na conferência
WSCAD/2023

How FaaS with DBaaS performs in different regions: an evaluation by the Orama Framework

Leonardo Rebouças de Carvalho¹, Bruno Kamienski¹, Aleteia Araujo¹

¹Department of Computer Science – University of Brasilia (UnB)
Campus Darcy Ribeiro – 70.910-900 – Brasilia – DF – Brazil

{leouesb,brunosabreu}@gmail.com, aleteia@unb.br

Abstract. *Studies indicate that cloud services based on the serverless paradigm, such as Function-as-a-Service (FaaS) should become the main mechanisms of the next generation of cloud computing. Given this perspective, public cloud providers have made efforts to expand the coverage of their services in order to meet this need. However, the effort needed to maintain equivalence between different regions highlights the importance of studying the behavior of FaaS environments in different regions of providers. This work presents a study aided by the Orama framework in order to evaluate the performance of the main FaaS integrated with Database-as-a-Service (DBaaS) services in five regions spread across the globe. The results indicate that the Alibaba provider was able to guarantee good equivalence between its regions, in addition to a lower average execution time. AWS and GCP had similar results, although the error rate on AWS was the highest on average. Azure, on the other hand, had the worst performance, with the highest average execution time, in addition to significant failure rates.*

1. Introduction

Serverless computing [Nupponen and Taibi 2020] as the default cloud programming paradigm have become an increasingly present idea in recent publications and this shows the importance it has gained for cloud computing. Function-as-a-Service (FaaS) [Schleier-Smith et al. 2021] allow users to publish functions written in some programming language supported by the provider and configure a trigger. When triggered, it is the role of the provider to ensure proper processing, whether in the face of low demand or when subjected to high levels of competition. The respective adjustment in the infrastructure takes place without any user intervention. This autascaling feature, combined with the billing model based on activating functions, explains the recent success of this service model. In this context and considering the Everything-as-a-Service (XaaS) concept, the main public cloud providers have been massively investing in serverless-oriented services, especially in Function-as-a-Service (FaaS) [Schleier-Smith et al. 2021].

Since the need for providers to be as close as possible to the end user, it is a common strategy used by cloud companies to deploy infrastructures geographically distributed around the world. For this strategy to be effective, it is important that the products marketed through the cloud are available in as many geographic locations as possible, and this introduces a major challenge in this context: maintaining equivalence for the same service across regions. In addition, in real solutions it is very common that different cloud services are combined to compose the solution, therefore, cloud providers offer

Anexo XXIII

Artigo publicado no periódico
SNCS/2024



ORIGINAL RESEARCH

Main FaaS Providers Behavior Under High Concurrency: An Evaluation with Orama Framework Distributed Architecture

Leonardo Rebouças de Carvalho¹ · Bruno Kamienski¹ · Aleteia Araujo¹

Received: 4 August 2023 / Accepted: 10 April 2024
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2024

Abstract

With the increasing adoption of Function-as-a-Service (FaaS) oriented solutions, there is a growing interest in tools that facilitate benchmarking in these environments. The Orama framework has emerged as a standout solution, offering high configurability and scalability for provisioning, running benchmarks, and performing statistical analysis. This research presents the use of the distributed architecture of the Orama framework in evaluating the main public FaaS providers in situations of high competition. The experimental results demonstrate the framework's effectiveness in orchestrating load tests with high levels of concurrency. Notably, the study reveals specific characteristics of the providers involved, with Alibaba Function showing excellent performance, registering the lowest average execution time and remaining free of failures. Google Cloud Function and AWS Lambda provide intermediate results at average runtime and report some failures. Meanwhile, Azure Function performs worst in average execution time and cold start. A deeper investigation into the Alibaba Function provider was conducted to verify if its performance maintains a more complex usage scenario, in this case, a use case merging FaaS and Database-as-a-Service.

Keywords Orama framework · FaaS · Database-as-a-Service · Benchmark · AWS · GCP · Azure · Alibaba cloud

Introduction

Global end-user spending on public cloud services is expected to experience a substantial 20.7% growth, reaching a total of \$591.8 billion in 2023, compared to \$490.3 billion in 2022 [1]. This remarkable projection underscores the robustness of the computing industry in this domain.

While the primary cloud models, such as Infrastructure-as-a-Service (IaaS) [2] and Platform-as-a-Service (PaaS), continue to attract significant investment, other models that embrace the concept of Everything-as-a-Service (XaaS) [3] also exhibit promising growth. Notably, serverless-based cloud computing models, like Function-as-a-Service (FaaS) [4], have emerged as the leading programming paradigm for the next generation of cloud computing.

Recently, there has been a growing interest in evaluations and benchmarking tools to analyze cloud service providers' performance. Notable examples of such tools include Sebs [5], PanOpticon [6], FaaS-Dom [7], BeFaaS [8], and the Orama framework [9, 10]. As cloud computing continues to evolve, providers offer a wide range of strategies, and services are deployed in various regions, making it crucial for these studies to adapt dynamically to real-world scenarios. To address this need for adaptability, this research applies the distributed architecture for the Orama framework [9] to explore a broader range of FaaS scenarios in the main public providers. Leveraging a master/workers architecture, workloads can be effectively distributed among multiple instances, including those geographically dispersed,

Leonardo Rebouças de Carvalho, Bruno Kamienski and Aleteia Araujo have contributed equally to this work.

This article is part of the topical collection "Recent Trends on Cloud Computing and Services Science" guest edited by Claus Pahl and Maarten van Steen.

✉ Leonardo Rebouças de Carvalho
leouesb@gmail.com

Bruno Kamienski
brunosabreu@gmail.com

Aleteia Araujo
aleteia@unb.br

¹ Department of Computer Science, University of Brasília, Campus Darcy Ribeiro, Brasília, Federal District 70910-000, Brazil

Anexo XXIV

Artigo publicado na conferência
MobiQuitous/2024

Achieving Observability on Fog Computing with the use of open-source tools

Breno Costa^{1,2}, Abhik Banerjee², Prem Prakash Jayaraman², Leonardo R. Carvalho¹, João Bachiega Jr.¹, and Aleteia Araujo¹

¹ Department of Computer Science - University of Brasília (UnB) - Brasília - DF - Brazil
{brenogscosta, joao.bachiega.jr, leouesb}@gmail.com and aleteia@unb.br

² School of Science, Computing and Engineering Technologies - Swinburne University of Technology
Melbourne - Victoria - Australia {abanerjee, pjayaraman}@swin.edu.au

Abstract. Fog computing can provide computational resources and low-latency communication at the network edge. But with it comes uncertainties that must be managed in order to guarantee Service Level Agreements. Service observability can help the environment better deal with uncertainties, delivering relevant and up-to-date information in a timely manner to support decision making. Observability is considered a superset of monitoring since it uses not only performance metrics, but also other instrumentation domains such as logs and traces. However, as Fog Computing is typically characterised by resource-constrained nodes and network uncertainties, increasing observability in fog can be risky due to the additional load injected into a restricted environment. There is no work in the literature that evaluated fog observability. In this paper, we first outline the challenges of achieving observability in a Fog environment, based on which we present a formal definition of fog observability. Subsequently, a real-world Fog Computing testbed running a smart city use case is deployed, and an empirical evaluation of fog observability using open-source tools is presented. The results show that under certain conditions, it is viable to provide observability in a Fog Computing environment using open-source tools, although it is necessary to control the overhead modifying their default configuration according to the application characteristics.

Keywords: Observability · Fog Computing · Edge Computing · Metrics, Logs, Traces.

1 Introduction

Fog computing is a computing model that brings computation and data storage closer to where it is needed, typically at the edge of the network. The main characteristics of Fog Computing are its distributed and decentralised nature, its focus on low latency and high bandwidth communication, and its ability to support a wide range of devices and applications [7]. Fog computing can also leverage existing network infrastructure and resources, such as routers, gateways, and edge devices, to provide a more efficient and cost-effective computing environment [26].

In the context of IoT applications, Fog Computing can play a crucial role in enabling real-time data processing, analytics, and decision making at the edge of the network. This is particularly important for applications that require low-latency and high-bandwidth communication, such as industrial automation, autonomous vehicles, and smart cities [38]. In this way, Fog Computing can enable more intelligent and autonomous IoT systems, reduce network congestion and latency, and improve the overall performance and reliability of IoT applications [38].

IoT applications can have a more fragmented code base compared to traditional applications. This is because they often involve multiple devices and systems that need to work together seamlessly, which can require different programming languages, frameworks, and protocols. In this scenario, there is increased complexity in providing system maintenance and a higher risk of not meeting service level agreements (SLAs) [51]. A possible solution to deal with this scenario is to increase the observability of IoT applications to achieve timely decision-making and fast application troubleshooting [41]. Observability is a concept borrowed from control theory and recently applied to the monitoring and debugging of distributed systems [32]. It refers to the ability to understand the internal state of a system based on its external outputs [30].

Monitoring is based on the collection and analysis of performance metrics from applications and helps to determine whether SLAs are guaranteed [45]. Observability is referenced as a superset of monitoring that uses

Anexo XXV

Artigo publicado como capítulo em
CCSS/2024

Empowering Statistical Analysis in FaaS Environments Through the Orama Framework

Leonardo Rebouças de Carvalho¹[0000-0001-7459-281X], Bruno Kamienski²[0000-0001-5817-8694], and Aleteia Araujo³[0000-0003-4645-6700]

¹ University of Brasilia, Brasilia, 70910-900, Brazil leouesb@gmail.com

² University of Brasilia, Brasilia, 70910-900, Brazil brunosabreu@gmail.com

³ University of Brasilia, Brasilia, 70910-900, Brazil aleteia@unb.br

Abstract. The Function-as-a-Service cloud computing model has aroused great interest in the community. In view of this, the importance of tools that help in the execution of benchmarks in environments with this architecture has grown. The Orama framework has positioned itself as an aid tool in this type of evaluation, allowing the comparison of environments of this category through statistical analyses with factorial design and t-test, as well as a direct comparison between the evaluated levels of concurrence. In this work, the Orama framework is used for comparative execution of benchmarks between four large cloud providers (AWS, GCP, Azure and Alibaba) in two different use case scenarios. The results showed that Alibaba maintained an average time lower than its rivals, while Azure suffered from a high penalty on the first request due to the cold start. AWS and GCP presented similar performances, including failures in the face of high levels of concurrence.

Keywords: Cloud · FaaS · Benchmark · AWS · GCP · Azure · Alibaba · Factorial Design · T-test · Orama Framework.

1 Introduction

According to studies, the major computing paradigm of the Cloud Era will be Function-as-a-Service (FaaS) [18], also known as Serverless [16] or Backend-as-a-Service [18]. This success may be partially attributed to the simplicity with which this service model can be used, as well as to its automatic scaling system, and the payment model based on requests and execution time.

As FaaS-based designs are adopted more often, questions have been raised about how well these environments would operate. FaaSdom [13], Sebs [5] and Orama [3] are examples of studies on the behavior of FaaS services. The first two being focused on the characteristics of programming languages and providers' methodologies, while the last one, Orama framework, is more comprehensive and allows the elaboration of use cases that integrate different solutions within the providers' platforms.

Applications, however, usually make use of several cloud services. It is essential to understand how this type of software operates in this scenario with

Anexo XXVI

Artigo publicado na conferência
UCC/2024

A Cost-Efficient Resource Allocation for Fog Computing with Users and Providers Perspective

1st Joao Bachiega Jr. 2nd Breno Costa 3rd Michel J. F. Rosa 4th Leonardo R. Carvalho
University of Brasilia, Brazil *University of Brasilia, Brazil* *University of Brasilia, Brazil* *University of Brasilia, Brazil*
 joao.bachiega.jr@gmail.com brenogcosta@gmail.com michelj@gmail.com michelj@gmail.com

5th Marcelo A. Marotta 6th Aleteia Araujo 7th Rajkumar Buyya
University of Brasilia, Brazil *University of Brasilia, Brazil* *The University of Melbourne, Australia*
 marcelo.marotta@unb.br aleteia@unb.br rbuyya@unimelb.edu.au

Abstract—Fog computing paradigm allows allocating computational resources and services at the edge of the network, closer to the end devices and users, complementing Cloud computing. Fog nodes comprise computational capabilities (such as CPU, memory, and disk) and behavioral characteristics (such as availability, scalability, and mobility). It should also be considered that while the end-user's goal is to obtain the best available resources, the service provider's concern is meeting user requests using minimal resources. Furthermore, it is crucial to consider the financial cost of each resource in both scenarios. Taking all of this into account, obtaining a cost-effective optimized resource allocation is viewed as a challenge. In this article, we propose an efficient algorithm for resource allocation, considering both the provider's and the end-user's perspectives, exploiting computational capabilities, behavioral characteristics, and the financial cost. The tests were carried out in both real and simulated environments and demonstrated that our proposal complies with the resource allocation needs for Fog computing and has a better performance compared to a similar solution.

Index Terms—Fog computing, resource allocation, provider's perspective, end-user's perspective, cost-efficiency

I. INTRODUCTION

Fog computing has emerged as a promising solution for Cloud computing bottlenecks with regard to high latency and response time [1]. To address these bottlenecks, Fog computing is designed as a highly geographically distributed and heterogeneous network composed of low computational capacity devices with different setups closer to end-users. The devices can remotely process the workloads initially meant for the Cloud with smaller response time and latency. This network has been used in dynamic contexts, with devices sharing and competing for computational resources, such as the Internet of Things (IoT). This competition creates uncertainty for resource management in Fog computing environments, leading to possible overloaded or underused nodes, causing unnecessary power consumption, service unavailability, high response time, and decreased reliability [2]. In such a scenario, resource allocation to determine suitable resources that satisfy a required workload remains a challenge [3].

A relevant concept in a Fog computing environment is the Fog node, defined as any computational device with

system and hardware resources added to high communication capability [4]. A fog node has computational capabilities, such as CPU, storage, memory, and other hardware attributes that can be measured. In addition, reflecting the dynamicity and unpredictability of Fog computing, the Fog node is also made up of some behavioral characteristics, such as availability, interoperability, and mobility [5].

The end-user and service provider perspectives must be considered when implementing Fog computing management. On the one hand, end-users always want to obtain the best resources available, those with greater computational power and higher values for behavioral characteristics. On the other hand, service and infrastructure providers are interested in delivering minimal resources to avoid unnecessary costs or situations where resources are unavailable to other users [6]. Although it is desirable to consider the behavioral characteristics, their degradation is not an obstacle to the execution of the application. In this sense, most Fog computing resource allocation solutions focus exclusively on hardware attributes without considering behavioral characteristics [7] [8]. Despite this trend, some publications consider only the user's perspective [9] [10]. To the best of our knowledge, considering both perspectives, i.e., balancing the needs of end-users and providers with the computational capabilities and behavioral characteristics of the Fog node, is a matter of investigation [6].

This work presents an algorithm for resource allocation in Fog computing, considering hardware attributes and behavioral characteristics. Thus, the main contributions of this work are:

- A Fog node cost-efficient decision-making algorithm that considers both hardware attributes and behavioral characteristics of available nodes, considering the perspectives of end-users and providers. This model allows parameters to be weighted and balanced to create a ranking of available Fog nodes that comply with the user's request;
- A prototype of a system that efficiently uses the proposed algorithm, being scalable in terms of the number of nodes in the environment and adherent to the Fog aspects;
- An analysis of the proposed algorithm efficiency performed in a real test environment, validating the system

Anexo XXVII

Artigo publicado na conferência
SSCAD/2025

Predicting FaaS Runtime with the Orama Framework Using Machine Learning

Leonardo Rebouças de Carvalho¹, Geraldo Pereira Rocha Filho^{1,2}, Aleteia Araujo¹

¹Computer Science Department - University of Brasilia
Campus Darcy Ribeiro – Brasilia – DF – Brazil

²Department of Exact and Technological Sciences, State University of Southwest Bahia
Vitória da Conquista – BA – Brazil.

leouesb@gmail.com, geraldo.rocha@uesb.edu.br, aleteia@unb.br

Abstract. *One of the significant challenges in Function-as-a-Service (FaaS) is the unpredictability of function runtimes, which can lead to cost overruns and performance degradation when deploying applications across multiple cloud providers. This paper presents a Machine Learning (ML)-based predictor integrated into the Orama Framework, which combines static code metrics (Halstead complexity measures) with empirical performance data to estimate function runtimes directly from the source code. Three neural network architectures (Dense, LSTM, and BLSTM) were evaluated, with the BLSTM achieving the highest accuracy ($R^2 = 0.91$) and a 20% lower MSE compared to the baselines. The predictor is available via API and through a graphical interface within the Orama Framework, supporting deployment planning, multi-cloud comparisons, and cost-performance optimization for serverless applications.*

1. Introduction

Serverless computing has emerged as one of the leading paradigms for developing and deploying microservice-based applications. In this model, Function-as-a-Service (FaaS) [Schleier-Smith et al. 2021] enables the execution of functions on demand with automatic scalability, eliminating the need for infrastructure management. However, the unpredictability of function runtime remains a major challenge. In multi-cloud environments, unexpected delays can compromise SLAs, degrade user experience, and lead to significant additional costs. Recent studies indicate that variations of only a few milliseconds per execution can result in substantial financial losses when accumulated over time.

The runtime of serverless functions is influenced by multiple factors, including code complexity, runtime environment, memory configuration, scaling policies, and provider-specific characteristics such as those of AWS Lambda, Google Cloud Functions, Azure Functions, and Alibaba Function Compute. Despite its importance, developers often lack tools that accurately estimate runtime before deployment or that allow informed comparisons among providers. Traditional benchmarking approaches face limitations, including low reproducibility, limited support for multiple providers, and difficulty capturing contextual variations, which can lead to inefficient resource allocation and suboptimal deployment decisions. Predicting runtime in serverless systems is challenging because many factors interact in unpredictable ways. This complexity makes it difficult to design models that capture consistent and reliable patterns.

Anexo XXVIII

**Artigo publicado na conferência
ERI-GO/2025**

Aplicação de *Machine Learning* à Predição de Tempo de Execução em FaaS com o *Framework Orama*

Leonardo Rebouças de Carvalho¹, Geraldo Pereira Rocha Filho^{1,2}, Aleteia Araujo¹

¹Depto de Ciência da Computação - Universidade de Brasília
Campus Darcy Ribeiro – Brasília – DF – Brazil

²Depto de Ciências Exatas e Tecnológicas, Universidade Estadual do Sudoeste da Bahia
Vitória da Conquista – BA – Brazil.

leouesb@gmail.com, geraldo.rocha@uesb.edu.br, aleteia@unb.br

Resumo. Um dos principais desafios em *Function-as-a-Service* (FaaS) é a imprevisibilidade do tempo de execução das funções, o que pode causar aumento de custos e degradação de desempenho em aplicações distribuídas entre provedores de nuvem. Este artigo apresenta um preditor baseado em *Machine Learning* (ML) integrado ao *Framework Orama*, que combina métricas estáticas de código (medidas de complexidade de Halstead) e dados empíricos de desempenho para estimar o tempo de execução diretamente a partir do código-fonte. Foram avaliadas três arquiteturas de redes neurais (Dense, LSTM e BLSTM), sendo a BLSTM a que apresentou maior precisão.

1. Introdução

A computação *Serverless* [Nupponen and Taibi 2020] tornou-se um paradigma central para aplicações baseadas em microsserviços, permitindo execução sob demanda de funções via *Function-as-a-Service* (FaaS) com escalabilidade automática e sem necessidade de gerenciar infraestrutura. Porém, a imprevisibilidade do tempo de execução continua sendo um desafio, especialmente em ambientes *multicloud*, onde pequenas variações podem afetar a experiência do usuário e custos.

O desempenho das funções é influenciado por fatores como complexidade do código, ambiente de execução, memória, políticas de escalonamento e características específicas de provedores como AWS Lambda, Google Cloud Functions, Azure Functions e Alibaba Function Compute. Ferramentas tradicionais de *benchmarking* apresentam limitações de reprodutibilidade e suporte a múltiplos provedores, dificultando comparações e decisões de implantação.

Para contornar essas limitações, este trabalho estende o *Framework Orama* [Carvalho et al. 2024]¹ com um preditor de tempo de execução baseado em *Machine Learning* (ML), combinando métricas estáticas de código, principalmente as de *Halstead*, com dados empíricos de *benchmarks*. Três arquiteturas de redes neurais (Dense, LSTM e BLSTM) foram avaliadas, destacando-se a BLSTM ($R^2 = 0,91$ e MSE 20% menor). O preditor foi integrado ao Orama via APIs e interface gráfica, permitindo comparações entre provedores, planejamento de implantações e otimização de custo e desempenho em aplicações *Serverless*.

¹<https://github.com/unb-faas/orama>

Anexo XXIX

**Artigo publicado na conferência
CLOSER/2026**

Orama++: Extending Serverless Benchmarking with Tiobe and Halstead Metrics for Improved Performance Prediction

Leonardo Rebouças de Carvalho¹^a, Geraldo Pereira Rocha Filho²^b and Aleteia Araujo¹^c

¹Computer Science Department, University of Brasilia, Campus Darcy Ribeiro, Brasilia-DF, Brazil

²Department of Exact and Technological Sciences, State University of Southwest Bahia, Vitória da Conquista-BA, Brazil
leouesb@gmail.com, geraldo.rocha@uesb.edu.br, aleteia@unb.br

Keywords: Serverless Computing, FaaS, Performance Prediction, Code Complexity Metrics, Machine Learning, FaaS Benchmarking, BLSTM, LSTM, Dense, Orama, Orama++

Abstract: The unpredictability of execution time and cost in Function-as-a-Service (FaaS) environments remains a critical challenge for serverless development. This paper introduces Orama++, an evolution of the Orama benchmarking framework that incorporates a robust Machine Learning-based performance prediction system, focusing on forecasting execution time and operational costs. The study investigates the impact of different static code complexity metrics (Tiobe, Halstead, and their combinations) on the predictive power of neural network models. The effectiveness of Fully Connected Neural Networks (Dense), Long Short-Term Memory (LSTM), and Bidirectional Long Short-Term Memory (BLSTM) architectures was evaluated. The findings demonstrate that, across all evaluated scenarios, BLSTM models consistently outperformed both the Dense and LSTM architectures. In terms of accuracy, measured by the coefficient of determination (R^2), the notable results obtained with the BLSTM model were: the input scenario with Tiobe metrics achieved an average R^2 of 86%; the combined scenario reached 91%; and the scenario using Halstead metrics demonstrated the best accuracy, with 92%. Due to its superior prediction quality, the BLSTM model trained with Halstead metrics was selected and designated as the new core performance predictor for the Orama++ framework. These results validate Orama++ as an essential tool for resource and cost predictability, enabling informed decisions regarding FaaS function optimization and deployment.

1 INTRODUCTION

The increasing adoption of serverless computing has significantly reshaped the design and deployment of cloud-native applications (Nupponen and Taibi, 2020). In particular, the Function-as-a-Service (FaaS) paradigm (Schleier-Smith et al., 2021) abstracts infrastructure management from developers, offering automatic scaling, reduced operational overhead, and a fine-grained pay-per-use pricing model. These characteristics have made FaaS attractive for a wide range of workloads, from lightweight APIs to data processing and scientific computing tasks. Nevertheless, this abstraction layer introduces substantial uncertainty regarding runtime behavior, making the prediction of execution time and operational cost a persistent challenge in serverless environments

(Schleier-Smith et al., 2021).

The unpredictability of FaaS performance is attributed to factors inherent to cloud provider infrastructures, such as cold starts, resource sharing, and opaque scheduling policies (Schleier-Smith et al., 2021). As a result, identical functions may exhibit significantly different execution times across invocations, even under similar configurations. This variability directly affects cost estimation, since pricing models are tightly coupled to execution duration and allocated resources. Consequently, developers and system architects often struggle to define reliable service-level agreements (SLAs) and to perform effective resource planning, frequently resorting to empirical benchmarking and iterative experimentation, which are both time-consuming and financially inefficient.

To mitigate these challenges, several studies have proposed performance modeling and prediction techniques for FaaS platforms (Tomaras et al., 2023; Horovitz et al., 2019; Dakkak et al., 2019; Pandey

^a <https://orcid.org/0000-0001-7459-281X>

^b <https://orcid.org/0000-0001-6795-2768>

^c <https://orcid.org/0000-0003-4645-6700>