

UNIVERSIDADE DO VALE DO RIO DOS SINOS — UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA
NÍVEL MESTRADO

MATEUS RAUBACK AUBIN

He-LASTIC:

**UM MODELO DE ELASTICIDADE EM NUVEM BASEADO EM DUAS CAMADAS
PARA ACELERAR O CÁLCULO DE *BEST-FIT* EM SISTEMAS DE SUBSTITUIÇÃO
FILOGENÉTICA**

São Leopoldo
2019

Mateus Rauback Aubin

He-LASTIC:

**um Modelo de Elasticidade em Nuvem Baseado em Duas Camadas para Acelerar o
Cálculo de *best-fit* em Sistemas de Substituição Filogenética**

Dissertação apresentada como requisito parcial
para a obtenção do título de Mestre pelo
Programa de Pós-Graduação em Computação
Aplicada da Universidade do Vale do Rio dos
Sinos — UNISINOS

Orientador:
Prof. Dr. Rodrigo da Rosa Righi

São Leopoldo
2019

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)

Aubin, Mateus Rauback

He–lastic: um modelo de elasticidade em nuvem baseado em duas camadas para acelerar o cálculo de *best-fit* em sistemas de substituição filogenética / Mateus Rauback Aubin — 2019.

186 f.: il.; 30 cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, 2019.

“Orientador: Prof. Dr. Rodrigo da Rosa Righi, Unidade Acadêmica de Pesquisa e Pós-Graduação”.

1. Computação Paralela e Distribuída. 2. Computação em Nuvem. 3. Elasticidade. 4. Bioinformática. 5. Filogenética. I. Título.

CDU 004.75

Bibliotecária responsável: Fulana da Silva — CRB 12/3456

(Esta folha serve somente para guardar o lugar da verdadeira folha de aprovação, que é obtida após a defesa do trabalho. Este item é obrigatório, exceto no caso de TCCs.)

AGRADECIMENTOS

Ao Prof. Dr. Rodrigo da Rosa **Righi** pelo tempo e a confiança continuamente investidos em mim desde a graduação.

A minha noiva, **Natália**, por me dar condições para desenvolver este trabalho e me apoiar incondicionalmente.

Aos meus **pais** que sempre me incentivaram a buscar e aprimorar meus conhecimentos.

Aos demais **professores** que influenciaram na minha trajetória desde a Graduação até aqui no Mestrado, em especial aos professores Cristiano André da Costa e Jorge Luis Victória Barbosa pelas contribuições durante o desenvolvimento desta dissertação.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (**CAPES**), pela bolsa de estudos.

A Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (**FAPERGS**), por contemplar nossa proposta, Combinando Elasticidade Multi-Métrica em Nuvem nas Modalidades Vertical e Horizontal para Análise de Genes e Redução de Pragas Agrícolas, no Edital 02/2017 - PqG.

RESUMO

Atualmente são escassos os exemplos de áreas do conhecimento que não se beneficiam de recursos tecnológicos para aumentar a produtividade . Também figurando como uma das beneficiadas por esta tendência, a bioinformática, que surge da união entre biologia, ciência da computação e medicina, estabeleceu técnicas, algoritmos e bases de dados que se tornaram fundamentais para o execício da pesquisa, seja no âmbito acadêmico ou profissional. Avanços recentes como as tecnologias de Sequenciamento genético de Próxima Geração (NGS) trouxeram abundância de dados para a bioinformática e a filogenética, que foca no estudo das relações genealógicas entre organismos, aumentando a qualidade dos estudos outrora limitados apenas a observações manuais. Com o objetivo de aumentar o conhecimento sobre bioinformática e filogenética, de especial relevância para o projeto contemplado pela FAPERGS em colaboração com o laboratório de Biologia da UNISINOS, buscamos realizar um mapeamento do estado da arte através de uma revisão da literatura que contribuiu para a elaboração de uma taxonomia capaz de classificar os projetos de software encontrados em termos de suas categorias, métodos e finalidades. Concluímos com base na bibliografia que, contrastando com os recentes avanços na filogenética, os projetos de software estão tecnologicamente defasados, demandando investimentos em modernização principalmente no que diz respeito a aplicação de técnicas mais recentes como elasticidade de recursos, balanceamento de carga e aceleração por GPUs. Nossa principal contribuição se dá através do modelo He–lastic onde propomos uma abordagem de otimização para o projeto jModelTest que contempla o uso de elasticidade de recursos em um ambiente de computação em nuvem, resultando em uma arquitetura de elasticidade em duas camadas, baseada nas tecnologias de FaaS e Orquestração de Contêineres e que tem por objetivo absorver, na primeira camada, as tarefas de curta duração, deixando as demais para a segunda camada de elasticidade. Seguindo adiante buscamos futuramente aprimorar a taxonomia e, principalmente, implementar o modelo proposto através de um protótipo, utilizando-o na avaliação dos cenários propostos neste trabalho e objetivando uma análise comparativa com o estado atual do projeto jModelTest sob a luz de critérios como a eficiência computacional e econômica da nossa proposta.

Palavras-chave: Computação Paralela e Distribuída. Computação em Nuvem. Elasticidade. Bioinformática. Filogenética.

ABSTRACT

Currently there are few examples of areas of knowledge that do not benefit from technological resources to increase productivity. Also one of the beneficiaries of this trend, bioinformatics, which emerges from the combination of knowledge in biology, computer science and medicine, has established techniques, algorithms and databases that have become fundamental for research, whether in the academic or professional spheres. Recent advances such as Next Generation Sequencing (NGS) technologies have brought abundance of data to bioinformatics and phylogenetics, which focuses on the study of genealogical relationships between organisms, increasing the quality of studies once limited to manual observations. With the objective of increasing the knowledge about bioinformatics and phylogenetics, of special relevance to the project funded by FAPERGS in collaboration with the Biology laboratory of UNISINOS, we sought to carry out a mapping of the state of the art through a literature review that contributed to the formulation of a taxonomy capable of classifying the software projects found in terms of their categories, methods and purposes. We conclude, on the basis of the bibliography, that in contrast to the recent advances in phylogenetics, software projects are technologically outdated, requiring investments in modernization mainly regarding the application of more recent techniques such as resource elasticity, load balancing and GPU acceleration . Our main contribution is through the He–lastic model, where we propose an optimization approach for the jModelTest project that contemplates the use of resource elasticity in a cloud computing environment, resulting in a elasticity architecture composed of two layers based on FaaS and Container Orchestration technologies and that aims to absorb, in the first layer, short duration tasks, leaving the remaining tasks for the second layer of elasticity. In the future, we intend to improve the taxonomy and, mainly, to implement the proposed model through a prototype, using it in the evaluation of the scenarios proposed in this work and aiming at a comparative analysis with the current state of the jModelTest project under light of criteria such as computational and economic efficiency of our proposal.

Keywords: Parallel and Distributed Computing. Cloud Computing. Elasticity. Bioinformatics. Phylogenetics.

LISTA DE FIGURAS

1	Exemplo de arquivo no formato FASTA contendo múltiplas sequências moleculares representadas através da codificação de aminoácidos.	25
2	Representação gráfica de uma árvore filogenética em uma tentativa de reconstruir a hipotética Árvore da Vida até o ancestral universal.	26
3	Taxas relativas de substituição para três sistemas baseados em cadeias de Markov	30
4	Taxonomia da Elasticidade apresentando quais são as Modalidades e Políticas disponíveis além das diferentes estratégias de elasticidade.	35
5	Diagrama ilustrando as principais diferenças no que tange às estratégias de virtualização baseadas em Contêineres e Máquinas Virtuais	37
6	Diagrama representando a divisão de responsabilidades sobre as diversas camadas que compõem um ambiente computacional contrastando ambientes locais com diferentes modelos de serviço habilitados pela computação em nuvem (IaaS, PaaS, Contêineres,e FaaS).	42
7	Diagrama representando a estrutura de busca e triagem utilizada neste trabalho para a revisão de literatura.	47
8	Taxonomia elaborada para a classificação dos trabalhos identificados durante a etapa de revisão de literatura.	49
9	Comparativo entre a evolução do poder computacional das CPUs versus GPUs medido em GFLOP/s.	53
10	Gráfico apresentando os projetos mais relevantes para esta pesquisa segundo o critério de que somados componham até 85% do total de citações dentre os trabalhos encontrados na revisão de literatura.	60
11	Linha do tempo do projeto jModelTest: histórico de publicações relevantes para compreensão do contexto, objetivos e evolução da ferramenta.	61
12	Pseudocódigo retratando o cerne do fluxo de trabalho executado pelo jModelTest durante o teste de adequação de sistemas de substituição moleculares. .	62
13	Exemplo minimalista do fluxo de tarefas típico para realização de análises filogenéticas no laboratório de filogenética da UNISINOS	63
14	Representação de carga em múltiplos processadores ou nodos durante a execução de processos com complexidade variável, exemplificando o cenário encontrado pela aplicação jModelTest durante o teste de adequação de sistemas de substituição de sequências moleculares onde parte significativa dos recursos pode permanecer ociosa enquanto aguarda a conclusão dos cálculos que exigem maior poder computacional.	65
15	Visão conceitual do modelo He–lastic ilustrando como suas duas camadas de elasticidade interagem para atender requisições.	68
16	Diagrama conceitual de alto nível da arquitetura do modelo He–lastic elucidando a relação entre seus componentes, assim como, a divisão em ambiente local e de computação em nuvem, com destaque para os elementos FaaS e Orquestrador de Contêineres, que representam as unidades de elasticidade do modelo, assim como as Filas de Mensagens que coordenam a comunicação entre os componentes do modelo	72
17	Diagrama UML de atividades do modelo He–lastic representando a interação, agrupada por etapa do processamento, entre os componentes da arquitetura. .	74

18	Diagrama ilustrando as principais diferenças entre os componentes das camadas de elasticidade: Apesar de compartilharem a estrutura de alto nível, o FaaS tem importantes limitações no que diz respeito ao tempo de execução e capacidade de processamento, o que permite ao provedor de computação em nuvem otimizar a gestão da elasticidade e multiplexar o uso de recursos entre usuários; em contrapartida, o orquestrador de contêineres permite que o usuário determine o comportamento da elasticidade e abdica da limitação no tempo de execução em troca de um maior fardo operacional e um ambiente de execução que deve ser gerido pelo próprio usuário.	77
19	Estratégia de paralelismo adotada pelo programa jModelTest: um arquivo com múltiplas sequências moleculares alinhadas serve como entrada para um número de tarefas de avaliação (uma para cada sistema de substituição selecionado) e que ficam, por sua vez, em uma fila de tarefas e são processados conforme existam recursos disponíveis dentro um conjunto fixo preestabelecido.	79
20	Estratégia de paralelismo adotada pelo modelo He–lastic: um arquivo com múltiplas sequências moleculares alinhadas serve como entrada para um número de tarefas de avaliação (uma para cada sistema de substituição selecionado) que ficam, por sua vez, em uma fila de tarefas sendo primeiramente processadas na camada FaaS e encaminhadas para a camada de Orquestração de Contêineres em caso de falha por tempo de processamento expirado. . . .	79
21	Linha de comando utilizada pelo jModelTest para controlar o <i>software</i> PhyML separada em parcela fixa e variável e apresentando amostras de preenchimento da parcela variável.	95
22	Mapa das regiões e respectivas zonas de disponibilidade fornecidas pelo provedor de computação em nuvem AWS, com os círculos verdes representando regiões em construção.	97
23	Variação no custo por região do provedor AWS no uso de recursos computacionais em instâncias da família m5.	97
24	Variação no custo por região do provedor AWS para a transferência de 1 TB de dados para a Internet.	97
25	Trecho de código do arquivo <code>modeltest.py</code> responsável por coordenar a execução do cálculo de adequação de sistemas de substituição molecular através da biblioteca PhyML, emulando o comportamento do jModelTest por meio de <i>traces</i> de execução.	101
26	Diagrama de arquitetura da implementação do protótipo do modelo He–lastic descrevendo as principais etapas no fluxo de execução.	104
27	Parâmetros de configuração do protótipo conforme especificados no arquivo <code>serverless.yml</code> e suas equivalências sobre as definições da Tabela 13.	107
28	Análise da evolução do custo de execução no modelo de computação FaaS conforme escolhas de parâmetros de Tempo Limite de Execução (t) e Memória Alocada (y).	113
29	Simulação do comportamento da camada de longa duração do modelo He–lastic mostrando a relação entre a quantidade de requisições e a disponibilidade de recursos para processamento.	114
30	Análise de custo por camada para o modelo He–lastic em um cenário de absorção de 50% das requisições pela camada FaaS	117

31	Comportamento esperado sobre o uso de recursos ao longo do tempo conforme modelos de execução baseado em recursos fixos, elasticidade e H _e lastic, evidenciando as diferenças no que diz respeito à capacidade ociosa e a evolução no aproveitamento de recursos ao longo do tempo.	119
32	Representação gráfica da Tabela 29 mostrando os tempos de execução por arquivo e cenário conforme a mediana.	123
33	Histograma da frequência dos tempos de execução para cada um dos sistemas de substituição avaliados durante a execução do arquivo 12-stamatakis-59 no cenário C_{j36}	125
34	Histograma da frequência dos tempos de execução para cada um dos sistemas de substituição avaliados durante a execução do arquivo 07-primate-mtDNA no cenário C_{j2}	125
35	Perda de eficiência observada durante as execuções do jModelTest através das médias dos tempos de execução por arquivo (pontos) e transição de cenários (cores) com a média geral em destaque.	127
36	Utilização média de CPU ao longo de uma execução contemplando todos os arquivos do <i>dataset</i> de testes em uma VM com 8 <i>cores</i> de processamento equivalente ao cenário C_{j8} (Duração total: 6 horas, 28 minutos).	128
37	Utilização média de CPU ao longo de execução contemplando todos os arquivos do <i>dataset</i> de testes em uma VM com 16 <i>cores</i> de processamento equivalente ao cenário C_{j16} (Duração total: 3 horas, 44 minutos).	128
38	Utilização média de CPU ao longo de uma execução contemplando todos os arquivos do <i>dataset</i> de testes em uma VM com 36 <i>cores</i> de processamento equivalente ao cenário C_{j36} (Duração total: 2 horas, 17 minutos).	128
39	Detalhe da Figura 38 excluindo a execução do arquivo 12-stamatakis-59. . .	128
40	Utilização média de CPU ao longo de uma execução contemplando os quatro maiores arquivos do <i>dataset</i> de testes em uma VM com 72 <i>cores</i> de processamento com o intuito de verificar o comportamento de do jModelTest (Duração total: 1 hora, 43 minutos).	128
41	Gráfico de evolução do tempo de execução observado a cada arquivo e cenário quando expressados e ordenados em termos do percentual do tempo máximo de execução.	133
42	Comportamento de elasticidade para uma execução do arquivo 02-rodents no cenário C_{c36}	140
43	Comportamento de elasticidade para uma execução do arquivo 08-HIV_vpu.ref2 no cenário C_{c36}	140
44	Comportamento de elasticidade contra-intuitivo para uma execução do arquivo 08-HIV_vpu.ref2 no cenário C_{c36}	140
45	Comportamento de elasticidade contra-intuitivo para uma execução do arquivo 07-primate-mtDNA no cenário C_{c36}	142
46	Comportamento de elasticidade para uma execução do arquivo 12-stamatakis-59 no cenário C_{c36}	142
47	Consumo percentual de CPU para cada VM envolvida na execução do arquivo 12-stamatakis-59 no cenário C_{c36} , onde cada linha representa uma VM ativa com quatro <i>cores</i> de processamento.	142

48	Mapa de calor da diferença nos tempos de execução entre os cenários de avaliação do modelo He–lastic com FaaS e Orquestração de Contêineres versus os cenários do jModelTest, onde cores mais fortes representam maiores diferenças.	147
49	Diagrama conceitual da abordagem tradicional para obtenção de elasticidade em um contexto de computação em nuvem: um平衡ador de carga distribui as requisições para diversas réplicas de uma máquina virtual enquanto um monitor de recursos avaliam o estado das réplicas e sugerem ações ao gerenciador de elasticidade com o objetivo de adequar os recursos disponíveis à carga computacional.	186

LISTA DE TABELAS

1	Codificação padrão para dados de sequências de DNA.	24
2	Codificação padrão para dados de sequências de RNA.	24
3	Codificação padrão para dados de sequências de aminoácidos.	24
4	Lista dos mais utilizados sistemas de substituição de sequências moleculares.	28
5	Comparativo das principais características nas ofertas de FaaS por parte de três grandes provedores de computação em nuvem.	40
6	Palavras-chaves utilizadas na composição dos termos de busca desta revisão da literatura.	45
7	Critérios de inclusão no conjunto de dados para trabalhos encontrados nesta revisão da literatura.	45
8	Editoras e fontes utilizadas na elaboração da presente revisão de literatura. .	46
9	Quantidade de trabalhos da revisão de literatura quando classificados conforme as categorias presentes na taxonomia proposta	50
10	Quantidade de trabalhos da revisão de literatura quando classificados de acordo com as características relevantes para a computação distribuída. . . .	51
11	Comparativo entre trabalhos encontrados na revisão de literatura classificados de acordo com a taxonomia proposta e atributos de interesse	55
12	Detalhamento das variáveis e funções utilizadas na definição de custo de execução do jModelTest.	66
13	Parâmetros necessários para a configuração do modelo He-lastic.	69
14	Comparativo entre as principais características das unidades de elasticidade utilizadas no modelo He-lastic	76
15	Configuração do processamento efetuado pelo jModelTest durante o uso de <i>Clustering Search</i> mostrando a quantidade e os sistemas de substituição molecular de acordo com a etapa.	78
16	Comparativo entre as principais variáveis que compõem a equação de Custo (Equação 4.9) para o modelo FaaS de acordo com os limites impostos por três grandes provedores de computação em nuvem.	83
17	Repositórios contendo o código-fonte dos projetos utilizados na avaliação do modelo He-lastic.	87
18	Conjuntos de dados utilizados na execução dos cenários de testes para o modelo He-lastic e o jModelTest.	88
19	Parâmetros disponibilizados pelo jModelTest que influenciam diretamente na quantidade de sistemas de substituição molecular que farão parte do teste de adequação.	89
20	Cenários de avaliação para o software jModelTest.	92
21	Cenários de avaliação para o modelo He-lastic.	92
22	Tipos de CPUs utilizadas no AWS Lambda e sua equivalência em termos de Famílias EC2.	98
23	Artefatos de código presentes nos três módulos do protótipo, com o reuso destacado em tons de azul.	102
24	Análise do comportamento das métricas Custo e Financeiro aplicados no modelo de computação FaaS conforme parâmetros requeridos pelo modelo He-lastic onde as cores identificam linhas com o mesmo custo total apesar de parâmetros distintos.	112

25	Análise do comportamento das métricas Custo e Financeiro aplicados no modelo de computação por Orquestração de Contêineres evidenciando a associatividade de custos.	115
26	Análise de custo para o modelo He–lastic ressaltando a relação entre os custos financeiros originados nas camadas FaaS e Orquestrador de Contêineres.	116
27	Quantidade de amostras coletadas na avaliação do jModelTest para cada arquivo do <i>dataset</i> conforme o cenário.	120
28	Coeficiente de Variação (CV) na avaliação do jModelTest contemplando tempos de execução obtidos por arquivo e cenário de avaliação.	120
29	Média e Mediana dos tempos de execução observados no jModelTest por arquivo e cenário de avaliação.	122
30	Quantidade de amostras coletadas na avaliação do modelo He–lastic para cada arquivo do <i>dataset</i> conforme o cenário.	130
31	Coeficiente de Variação (CV) na avaliação do He–lastic contemplando tempos de execução obtidos por arquivo e cenário de avaliação.	130
32	Mediana dos tempos de execução observados no modelo He–lastic por arquivo e cenário de avaliação.	132
33	Processos e Tempo de Execução médios do cenário somente FaaS (C_{f0}) do modelo He–lastic.	135
34	Tempos de Execução médios dos cenários somente Orquestrador de Contêineres do modelo He–lastic.	138
35	Razão dos sistemas de evolução processados pela camada de Orquestração de Contêineres sobre o total por arquivo e cenário observada na avaliação do He–lastic.	144
36	Diferenças nos tempos de execução entre os cenários de avaliação do modelo He–lastic com FaaS e Orquestração de Contêineres versus os cenários do jModelTest, com os melhores e piores resultados destacados em verde e vermelho, respectivamente.	146
37	Consolidação dos resultados obtidos nas execuções dos cenários mistos do modelo He–lastic em comparação com os resultados das execuções do jModelTest.	150
38	Resultados de performance e custo financeiro das avaliações dos cenários do modelo He–lastic para cada arquivo do <i>dataset</i> , contemplando apenas FaaS (C_{f0}), apenas Orquestrador de Contêineres (C_{c36}) e os cenários Mistos com melhor desempenho (C_{m1} , C_{m2} e C_{m4}).	153
39	Resultados de performance e custo financeiro das avaliações dos cenários do jModelTest por arquivo do <i>dataset</i>	154
40	Consolidação dos resultados obtidos dentre os cenários selecionados do modelo He–lastic e seus cenários análogos no jModelTest.	156
41	Evolução dos custos financeiros para manter disponíveis servidores capazes de processar os cenários avaliados para o jModelTest.	159
42	Quantidade de citações para cada trabalho encontrado no levantamento bibliográfico	183

LISTA DE SIGLAS

ACM	Associação para Maquinas Computacionais <i>Association for Computing Machinery</i>
AKS	<i>Azure Kubernetes Service</i>
API	Interface de Programação de Aplicação <i>Application Programming Interface</i>
AWS	Amazon Serviços da Web <i>Amazon Web Services</i>
BMC	<i>BioMed Central</i>
BSP	Fases Paralelas <i>Bulk Synchronous Parallel</i>
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CPU	Unidade Central de Processamento <i>Central Processing Unit</i>
CV	Coeficiente de Variação <i>Coefficient of Variation</i>
DNA	Ácido Desoxirribonucleico <i>Deoxyribonucleic Acid</i>
EC2	<i>Elastic Compute Cloud</i>
ECS	<i>Elastic Container Service</i>
FAPERGS	Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul
FASTA	Fast ALL
FaaS	Função como Serviço <i>Function as a Service</i>
FaaS	Função como um Serviço <i>Function as a Service</i>
GCP	<i>Google Cloud Platform</i>
GFLOP	Giga Operações de Ponto Flutuante por Segundo <i>Giga Floating Point Operations per Second</i>
GPU	Unidade de Processamento Gráfico <i>Graphics Processing Unit</i>
HGT	Transferência Horizontal de Genes <i>Horizontal Gene Transfer</i>
I/O	Entrada e Saída <i>Input and/or Output</i>
IEEE	Instituto dos Engenheiros Elétricos e Eletrônicos <i>Institute of Electrical and Electronics Engineers</i>

IaaS	Infraestrutura como Serviço <i>Infrastructure as a Service</i>
LBA	Atração por Longas Ramificações <i>Long Branch Attraction</i>
MCMC	Cadeias de Markov Monte Carlo <i>Markov Chain Monte Carlo</i>
ML	Máxima Verossimilhança <i>Maximum Likelihood</i>
MPI	Interface de Passagem de Mensagens <i>Message Passing Interface</i>
MSA	Alinhamento de Múltiplas Sequências [moleculares] <i>Multiple Sequence Alignment</i>
NCBI	Centro Nacional de Informações Biotecnológicas <i>National Center for Biotechnology Information</i>
NGS	Sequenciamento de Próxima Geração <i>Next Generation Sequencing</i>
NIST	Instituto Nacional de Padrões e Tecnologia <i>National Institute of Standards and Technology</i>
PLOS	Biblioteca Pública de Ciências <i>Public Library of Science</i>
PVM	Máquina Virtual Paralela <i>Parallel Virtual Machine</i>
PaaS	Plataforma como Serviço <i>Platform as a Service</i>
RNA	Ácido Ribonucleico <i>Ribonucleic Acid</i>
UML	Linguagem Unificada de Modelagem <i>Unified Modelling Language</i>
UNISINOS	Universidade do Vale do Rio dos Sinos
VM	Máquina Virtual <i>Virtual Machine</i>

SUMÁRIO

1 INTRODUÇÃO	17
1.1 Motivação	18
1.2 Questão de Pesquisa	19
1.3 Objetivos	20
1.4 Estrutura do Texto	21
2 FUNDAMENTAÇÃO TEÓRICA	22
2.1 Bioinformática e Filogenética	22
2.1.1 Alinhamento de Sequências Moleculares	23
2.1.2 Inferência Filogenética	25
2.1.3 Sistemas de Substituição de Sequências Moleculares	28
2.2 Computação de Alto Desempenho	30
2.2.1 Computação Paralela e Distribuída	32
2.2.2 Computação em Nuvem	33
2.2.3 Elasticidade	35
2.2.4 Orquestração de Contêineres	37
2.2.5 Serverless e FaaS (Function as a Service)	39
3 TRABALHOS RELACIONADOS	44
3.1 Metodologia de Seleção de Trabalhos	44
3.2 Taxonomia Elaborada	48
3.3 Análise do Estado da Arte	49
3.4 Lacunas de Pesquisa	52
4 MODELO He-LASTIC	58
4.1 Seleção da Aplicação	59
4.2 Decisões de Projeto	64
4.3 Arquitetura	67
4.4 Estratégias para Elasticidade	75
4.5 Métricas de Avaliação	80
4.6 Considerações Parciais	85
5 METODOLOGIA DE AVALIAÇÃO	86
5.1 Etapas e Cenários	86
5.1.1 Parâmetros e Conjuntos de Dados	87
5.1.2 Estratégia de Avaliação do jModelTest	90
5.1.3 Estratégia de Avaliação do He-lastic	92
5.1.4 Análise Comparativa	94
5.2 Aplicação	94
5.3 Infraestrutura	96
5.4 Protótipo	100
5.4.1 Implementação	100
5.4.2 Ferramentas	103
5.4.3 Arquitetura do Protótipo	104
5.4.4 Desafios e Dificuldades	106
5.4.5 Parâmetros	107
5.5 Considerações Parciais	108

6 RESULTADOS	110
6.1 Estudo de Custo–Benefício	111
6.1.1 Camada FaaS	111
6.1.2 Camada do Orquestrador de Contêineres	114
6.1.3 Camadas Combinadas	115
6.2 Avaliação do jModelTest	119
6.3 Avaliação do He–lastic	129
6.3.1 Cenário Somente FaaS	134
6.3.2 Cenário Somente Orquestrador de Contêineres	137
6.3.3 Cenário Misto	143
6.4 Avaliação de Custos	151
6.5 Considerações Parciais	160
7 CONCLUSÃO	164
7.1 Contribuições	167
7.2 Trabalhos Futuros	168
REFERÊNCIAS	170
APÊNDICE A – INFORMAÇÕES COMPLEMENTARES	183

1 INTRODUÇÃO

“You must not come lightly to the blank page.”

Stephen King

A biologia, no seu estudo dos seres vivos, é uma das mais antigas faces da ciência e por isso diversas foram as mudanças em seus modos de pesquisa ao longo dos anos. Eventualmente equipamentos de informática passaram a dividir espaço com microscópios nos laboratórios, aliando novas tecnologias ao vasto conhecimento acumulado em busca de respostas sobre as origens e os mecanismos de funcionamento da vida.

Atualmente são escassos os exemplos de áreas do conhecimento que não se beneficiam de recursos tecnológicos para aumentar a produtividade (DENNING; ROSENBLUM, 2009). Não é diferente com a biologia que, através da união de esforços com a medicina e a informática criou uma área específica de estudo, a bioinformática. Através desta colaboração surgiram algoritmos e bases de dados que se mostram, hoje em dia, fundamentais para o exercício da pesquisa, seja ela acadêmica ou profissional (OSTELL; MCENTYRE, 2007). Projetos como o de mapeamento do genoma humano (VENTER et al., 2001), iniciado em 1990 com o apoio de múltiplos países, impulsionaram a colaboração entre estas áreas, garantindo avanços significativos nos seus respectivos domínios do conhecimento.

Embora os avanços tenham sido significativos e amplamente variados, o acesso crescente a tecnologias como as de mapeamento genético aumentou o volume de dados gerado a partir de estudos biológicos, antes limitados a observações manuais (MOUNT, 2004). Bancos de dados de amostras genéticas, como o NCBI¹ (OSTELL; MCENTYRE, 2007), guardam uma infinidade de indivíduos das mais variadas espécies, possibilitando que pesquisadores realizem estudos tão amplos quanto aqueles que buscam mapear a árvore da vida, ou tão especializados quanto aqueles que se focam no estudo de uma única doença, por exemplo.

Além disso, mudanças nas arquiteturas de computadores e sistemas invalidaram diversas das hipóteses sobre as quais se apoiavam algumas das mais antigas soluções. Dentre as alterações mais emblemáticas estão a evolução dos processadores através do paralelismo, abandonando a corrida pelas cada vez maiores frequências de *clock* (BAKER; BUYYA, 1999), e o surgimento da computação em nuvem, que muda completamente a economia de escala e a necessidade de se comprar e manter recursos computacionais frente a oferta de computação virtualmente infinita pelos provedores (MELL; GRANCE, 2011).

Chegamos, então, a um ponto onde se faz necessária uma intervenção nos algoritmos e programas de bioinformática visando readequá-los a este novo cenário baseado em grandes volumes de dados e na computação em nuvem. Para tanto o presente trabalho se propõe a realizar

¹ Acessível através da URL: <https://www.ncbi.nlm.nih.gov/>

uma análise no cenário tecnológico no que tange aos processos e métodos de análise e inferência filogenética. Mais do que isso, a intenção é gerar ganhos que possam se estender a toda a comunidade acadêmica por meio de um modelo de computacional baseado em elasticidade de duas camadas.

1.1 Motivação

Resultado da poderosa combinação entre biologia, medicina e tecnologia da informação, a bioinformática se apresenta como um campo de estudo interdisciplinar focado em desvendar os mistérios da vida. Uma de suas vertentes, a filogenética ([Seção 2.1](#)), estreita o foco de pesquisa para a compreensão da evolução dos organismos e suas relações ancestrais através do uso de conhecimentos da genética e da genômica, combinados com estudos da teoria evolutiva. O advento das tecnologias chamadas de NGS, (*Next Generation Sequencing*) reduziu barreiras para a obtenção de sequências genéticas, ampliando a disponibilidade de dados genéticos e contribuindo para o objetivo definitivo da filogenética ([MOUNT, 2004; YANG, 2014](#)), encontrar o ancestral universal a todos os organismos vivos.

Apesar dos avanços no que diz respeito a obtenção de material genético, quando tratando-se de aspectos computacionais foi identificada, através de um extensivo levantamento bibliográfico ([Seção 3.3](#)), uma lacuna tecnológica que se torna evidente quando investigamos na intersecção entre filogenética e técnicas computacionais como computação distribuída, aceleração por GPU, o uso de elasticidade de recursos e técnicas de balanceamento de carga. Especulamos que tal situação tenha advindo do fato de que os algoritmos e métodos da filogenética são considerados maduros e bem definidos, enquanto as técnicas computacionais mencionadas se popularizaram anos depois. Esta situação é especialmente notória no que diz respeito ao uso de computação em nuvem e, mais especificamente, na elasticidade de recursos. [Jonas et al. \(2017\)](#) corroboram com esta hipótese ao postular que grande parte dos projetos de software para aplicações científicas e de análise não foi escrito por cientistas da computação e que este grupo foi deixado de fora da revolução que se tornou a computação em nuvem.

Através da oportunidade de colaboração com o laboratório de biologia da UNISINOS por meio do projeto contemplado pela FAPERGS – “Combinando Elasticidade Multi-Métrica em Nuvem nas Modalidades Vertical e Horizontal para Análise de Genes e Redução de Pragas Agrícolas”, o objetivo do presente trabalho é dar um significativo passo também rumo aos objetivos desta iniciativa, realizando um reconhecimento do estado da arte e estabelecendo as lacunas de pesquisa mais evidentes no contexto do projeto. Com a intenção de fomentar um entendimento compartilhado entre os colaboradores do projeto busca-se estabelecer, também, uma taxonomia referente à classificação dos objetivos dos trabalhos encontrados no estado da arte. Assim temos, nesta proposta, um dos primeiros resultados em relação a esta colaboração, aplicando conhecimentos de computação de alto desempenho, através da elasticidade oriunda da computação em nuvem, com o propósito de mitigar as dificuldades encontradas por pesquisadores

que atuam na filogenética.

Ao mapear os pontos de melhoria e propor novas abordagens para as ferramentas utilizadas no contexto de filogenética torna-se possível acelerar o processamento de dados e a quantidade de amostras sob análise, aumentando não só a eficiência dos programas como também dos profissionais da bioinformática. Dentre os benefícios esperados com esta abordagem é possível citar, em primeiro lugar, o aumento no uso de análises que talvez não sejam feitas em função da sua complexidade computacional e tempo de execução proibitivo e, em segundo lugar, a redução no tempo gasto pelos pesquisadores que já utilizam tais processos e ferramentas, possibilitando seu investimento na análise dos resultados obtidos. Em última instância estes fatores combinados têm potencial para contribuir em termos de qualidade e profundidade das análises no âmbito da filogenética, avançando a produção científica.

1.2 Questão de Pesquisa

Instigado pelos motivos citados embarcamos nesta jornada buscando aplicar nossos conhecimentos no campo da elasticidade pela computação em nuvem aos projetos da filogenética, definindo do seguinte modo nossa questão de pesquisa:

como seria **projeto um modelo de elasticidade** de recursos em um ambiente de computação em nuvem capaz de aperfeiçoar o processo de **adequação de sistemas** de substituição de sequências moleculares da **filogenética** de maneira **eficiente** do ponto de vista **computacional e econômico**?

Para viabilizar a resposta à esta pergunta se torna necessário esclarecer os elementos destacados na questão de pesquisa e que são detalhados ao longo deste documento nos seguintes pontos:

- **Projeto de um Modelo:** a partir da definição de um conjunto de decisões sobre o projeto ([Seção 4.2](#)) foi possível elaborar e detalhar a arquitetura de um modelo computacional ([Seção 4.3](#)) capaz de endereçar a presente questão de pesquisa;
- **Elasticidade:** apresentada por meio da fundamentação teórica na [Seção 2.2](#);
- **Adequação de Sistemas:** através de uma análise comparativa ([Seção 3.3](#)) foi encontrada uma quantidade significativa de projetos e delineamos as lacunas de pesquisa ([Seção 3.4](#)) para, finalmente, escolher uma aplicação candidata para o desenvolvimento ([Seção 4.1](#));
- **Filogenética:** também apresentada por meio da fundamentação teórica na [Seção 2.1](#);

- **Eficiente Computacional e Economicamente:** através do detalhamento das estratégias de elasticidade ([Seção 4.4](#)) foi formulada uma metodologia de avaliação ([Capítulo 5](#)) que será fundamental para estabelecer a eficiência de nossa proposta.

1.3 Objetivos

Conhecendo as motivações e a questão de pesquisa no âmbito deste trabalho, podemos estabelecer que seu objetivo geral é:

propor um modelo computacional que empregue a elasticidade de recursos oriunda da computação em nuvem para aumentar a eficiência computacional e econômica no problema do teste de adequação de sistemas de substituição de sequências moleculares no contexto da filogenética.

Contudo, visando detalhar este objetivo, assim como seus requisitos e resultados esperados, devemos dividi-lo em termos dos seus componentes, ou seja, os objetivos específicos desta pesquisa, sendo eles:

Objetivo 1: Familiarizar-se com a bioinformática e mais especificamente a filogenética;

Objetivo 2: Mapear o estado da arte no que diz respeito aos softwares utilizados no âmbito da filogenética;

Objetivo 3: Estabelecer uma taxonomia para classificação destes projetos e as características relevantes do ponto de vista computacional para identificar candidatos à receber as melhorias propostas;

Objetivo 4: Projetar um modelo capaz de fornecer ao projeto candidato capacidades de elasticidade de recursos com vistas a torná-lo apto para uso em ambientes de computação em nuvem;

Objetivo 5: Estabelecer uma arquitetura e estratégias capazes de, através da elasticidade, obter ganhos de performance, aproveitamento de recursos e redução no fardo operacional, resultando em decréscimo no custo de operação do software;

Objetivo 6: Avaliar, através de análises comparativas, o quanto eficaz será nosso modelo frente ao projeto candidato na sua forma atual.

1.4 Estrutura do Texto

O restante do texto está estruturado da seguinte forma: no [Capítulo 2](#) é apresentada a base teórica que sustenta esta pesquisa, composta majoritariamente por conhecimentos das áreas da Bioinformática e da Computação de Alto Desempenho. Dentre estes domínios são de especial interesse para esta dissertação os métodos de Inferência Filogenética e a Computação em Nuvem, mais especificamente o conceito de Elasticidade.

No [Capítulo 3](#) apresentam-se os trabalhos relacionados, assim como os critérios de seleção e escolha da literatura científica que contribuíram para determinar o estado da arte no que tange os assuntos abordados neste trabalho. Uma análise comparativa auxilia o leitor a compreender o panorama da área de estudo e contribui para a identificação de possíveis lacunas de pesquisa, onde esta dissertação se insere e busca, através do modelo proposto, sua contribuição científica.

Em seguida, no [Capítulo 4](#), o leitor será apresentado ao modelo He–lastic, sendo esta uma estratégia para preencher algumas das lacunas de pesquisa previamente identificadas. Aborda-se neste capítulo as questões relacionadas à decisões de projeto, arquitetura estratégias de elasticidade e as métricas de avaliação para o modelo proposto, fornecendo um panorama conceitual da solução apresentada

No [Capítulo 5](#) é apresentada a metodologia de avaliação do modelo proposto, abordando as etapas e os cenários que fazem parte da avaliação, aproveitando, também, para descrever em maiores detalhes as questões práticas da avaliação, abordando características da aplicação, da infraestrutura utilizada, assim como o desenvolvimento do protótipo e seus parâmetros.

A seguir, no [Capítulo 6](#), são apresentados os resultados obtidos conforme o protocolo de avaliação previamente descrito, onde são abordadas questões que dizem respeito a análise de custo–benefício da estratégia de elasticidade adotada pelo modelo proposto, os resultados da avaliação da aplicação de referência, e os resultados da avaliação do modelo He–lastic em suas variações, reservando também espaço para discussão sobre a comparação em termos de custo financeiro entre as duas abordagens.

Enfim, no [Capítulo 7](#), encerra-se a pesquisa com a apresentação das principais contribuições oriundas da adoção do modelo proposto, assim como, os possíveis caminhos disponíveis para aprofundar e aperfeiçoar a proposta desta dissertação em trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

“Knowledge speaks, but wisdom listens.”

Jimi Hendrix

Este capítulo apresenta conceitos fundamentais para o completo entendimento do presente trabalho, assim como, conceitos básicos das áreas de estudo e referências adicionais. Sendo imprescindíveis para a apreciação deste trabalho, conhecimentos nas áreas de bioinformática, filogenética e computação de alto desempenho.

Em um primeiro momento serão abordadas questões como: *a)* o que são sequências moleculares; *b)* como funciona o processo de inferência filogenética; *c)* quais são os métodos utilizados para este fim; e *d)* como modelos matemáticos são usados para auxiliar este processo. A seguir, apresentam-se questões relevantes ao domínio da ciência da computação, mais especificamente, no que diz respeito à computação de alto desempenho e alguns dos seus facilitadores como: *a)* técnicas de computação paralela e distribuída; *b)* estratégias de decomposição de problemas; *c)* o surgimento da computação em nuvem e suas implicações; *d)* o conceito de elasticidade de recursos e sua importância; *e)* o emprego de orquestração de contêineres como facilitador da elasticidade; e *f)* a ascensão das plataformas de *Function as a Service* (FaaS).

2.1 Bioinformática e Filogenética

A bioinformática pode ser definida, segundo [Ostell e McEntyre \(2007\)](#), como uma abordagem computacional para a gestão de conhecimento e análise de dados biomédicos, ou seja, um campo de estudo multidisciplinar que une os conhecimentos das áreas de biologia, medicina e tecnologia da informação. Tal combinação mostra-se relevante diante da crescente capacidade computacional e do volume de dados disponíveis para análises, auxiliando acadêmicos da biologia e da medicina na expansão do conhecimento científico. Uma definição ainda mais detalhada pode ser encontrada em [Moore \(2007\)](#), ao afirmar que o objetivo da bioinformática é “facilitar a gestão, análise e interpretação de dados oriundos de observação e experimentos, contemplando o desenvolvimento e a implementação de bancos de dados, análise de dados, mineração de dados e a interpretação e inferência de dados biológicos”.

Assim como a bioinformática, que advém da combinação entre biologia e ciência da computação, a filogenética emerge da associação entre os conhecimentos em biologia evolutiva e, mais recentemente, da genética. [Darwin \(1859\)](#) delineou as fundações da teoria evolutiva e toda uma área de estudo, em sua obra “A Origem das Espécies”, além de postular que todas as formas de vida na Terra descendem de um ancestral primordial. Neste contexto, Darwin equipou cientistas com um *framework* para inferência de relacionamentos entre espécies a partir de informações

morfológicas, paleontológicas e biogeográficas (KEANE, 2006). Posteriormente, Mendel (1866), através do seu artigo “Experimentos na hibridização de plantas”, introduziu conceitos básicos da genética como hereditariedade (através das características recessivas e dominantes), além da variabilidade de populações, fornecendo a teoria que até hoje suporta a genética como uma área de estudo.

Embora atualmente a relação entre estas duas áreas seja evidente, foi apenas no início do século XX que ambas correntes de pensamento foram unificadas em uma formulação teórica e matemática posteriormente batizada por Huxley (1974) de neo-Darwinismo. Avanços tecnológicos e o sequenciamento genético possibilitaram o surgimento da biologia molecular e da genômica (estudos com base no sequenciamento completo de um indivíduo) como áreas de estudo, sendo estas, o elo faltante para a definição da filogenética. Adachi e Hasegawa (1996) afirmam que os objetivos de estudos filogenéticos são duplos: de um lado reconstruir a mais correta relação genealógica entre organismos; e de outro, estimar a divergência de tempo desde que tais organismos compartilharam um ancestral comum. Extrapolando esta definição, Xia (2014), estabelece como verdadeiro objetivo da filogenia encontrar o ancestral universal a todos os organismos vivos, denominado *cenancestor*.

2.1.1 Alinhamento de Sequências Moleculares

Por meio do acúmulo de material genético mapeado torna-se evidente o ímpeto de pesquisadores, no verdadeiro espírito Darwiniano, ao comparar múltiplas amostras para identificar similaridades e pontos de diferença. No entanto, antes que qualquer análise comparativa possa ser realizada, é de suma importância garantir que as amostras sejam análogas entre si, evitando o risco de, em termos simples, comparar bicicletas com bananas. Neste momento, o alinhamento de sequências se apresenta como solução, uma vez que alinhamentos corretos são úteis para a predição estrutural e funcional de proteínas e, conforme apresentado na Subseção 2.1.2, indispensáveis para a análise filogenética (CLAVERIE; NOTREDAME, 2007).

Sequências genéticas podem ser representadas como sequências de caracteres oriundas de um alfabeto limitado (DARRIBA, 2015), podendo ser chamadas de bases ou nucleotídeos no caso de ácido desoxirribonucleico (DNA) e ácido ribonucleico (RNA), ou então de aminoácido no caso das proteínas. Zwickl (2006) explica que a preferência por dados de sequências genéticas se dá primordialmente pela facilidade e a velocidade com que os caracteres podem ser coletados e a tratabilidade no caso de modelagens estatísticas. Tais sequências costumam ser armazenadas em meios digitais através de formatos de arquivo como FASTA, exemplificado na Figura 1, e NEXUS, por exemplo, empregando representações de DNA, RNA, e aminoácidos, apresentadas respectivamente nas Tabelas 1, 2 e 3.

Computacionalmente, o alinhamento de múltiplas sequências, MSA, apresenta pelo menos três grandes desafios identificados por Mount (2004), sendo eles: *a*) encontrar um alinhamento ótimo entre mais de duas sequências que compreenda equivalências, diferenças e lacunas, além

Tabela 1 – Codificação padrão para dados de sequências de DNA.

Letra	Base de DNA
A	Adenina
C	Citosina
G	Guanina
T	Timina

Fonte: Adaptado de [Darriba \(2015\)](#).

Tabela 2 – Codificação padrão para dados de sequências de RNA.

Letra	Base de RNA
A	Adenina
C	Citosina
G	Guanina
U	Uracilo

Fonte: Adaptado de [Darriba \(2015\)](#).

Tabela 3 – Codificação padrão para dados de sequências de aminoácidos.

Letra	Símbolo	Aminoácido	Letra	Símbolo	Aminoácido
A	Ala	Alanina	M	Met	Metionina
C	Cys	Cisteina	N	Asn	Asparagina
D	Asp	Ácido aspártico	P	Pro	Prolina
E	Glu	Ácido glutâmico	Q	Gln	Glutamina
F	Phe	Fenilalanina	R	Arg	Arginina
G	Gly	Glicina	S	Ser	Serina
H	His	Histidina	T	Thr	Treonina
I	Ile	Isoleucina	V	Val	Valina
K	Lys	Lisina	W	Trp	Triptofano
L	Leu	Leucina	Y	Tyr	Tirosina

Fonte: Adaptado de [Darriba \(2015\)](#).

de, considerar também, o grau de variação entre todas as sequências; *b*) identificar um método razoável para calcular um critério de pontuação das colunas alinhadas; e *c*) posicionamento e critérios de pontuação das lacunas presentes dentre as várias sequências. Somente assim, uma vez alinhadas as sequências, é possível executar uma comparação justa entre as diferentes amostras disponíveis.

Dentre os diversos métodos possíveis para alinhamento de múltiplas sequências, três deles são destacados nominalmente por Claverie e Notredame (2007): *a*) Clustal – (SIEVERS et al., 2014): o mais popular, que emprega um método progressivo para construir os alinhamentos em uma sequência por vez; *b*) MUSCLE – (EDGAR, 2004): o mais eficiente, que é capaz de produzir rapidamente alinhamentos para centenas de sequências; e *c*) Tcoffee – (NOTREDAME et al., 2000): o mais preciso, que também emprega métodos progressivos, com o diferencial de avaliar segmentos entre todas as sequências do conjunto.

Figura 1 – Exemplo de arquivo no formato FASTA contendo múltiplas sequências moleculares representadas através da codificação de aminoácidos.

```
;LCBO - Prolactin precursor - Bovine
MDSKGSSQKGSRLLLLLVSNLLCQGVVSTPVCPNGPGNCQVSLRDLFDRAVMVSHYIHDLS
EMFNEFDKRYAQGKGFITMALNSCHTSSLPTPEDKEQAQQTHHEVILMSLILGLLRSWNDPLYHL
VTEVRGMKGAPDAILSRAIEIEEENKRLLEGMEMIFGQVIPGAKETEPYPVWSGLPSLQTKDED
ARYSAFYNLHCLRRDSSKIDTYLKLLNCRIYNNNC*

>MCHU - Calmodulin - Human, rabbit, bovine, rat, and chicken
ADQLTEEQIAEFKEAFSLFDKDGDGTITKELGTVMRSLGQNPTAEALQDMINEVDADGNGTID
FPEFLTMMARKMKDTDSEEIREAFRVFDKDGNGYISAAELRHVMTNLGEKLTDEEVDEMIREA
DIDGDGQVNYYEEFVQMMTAK*

>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIIYYGSYLYSETWNTGIMLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPIGTNLV
EWIWGGSVDKATLNRFCAFHFILPFTMVALAGVHTFLHETGSNNPLGLTSDDSKIPFHPYYTIKDFLG
LLILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTWIGSQVEPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

Fonte: Obtido pelo autor com a ajuda do laboratório de Biologia da UNISINOS.

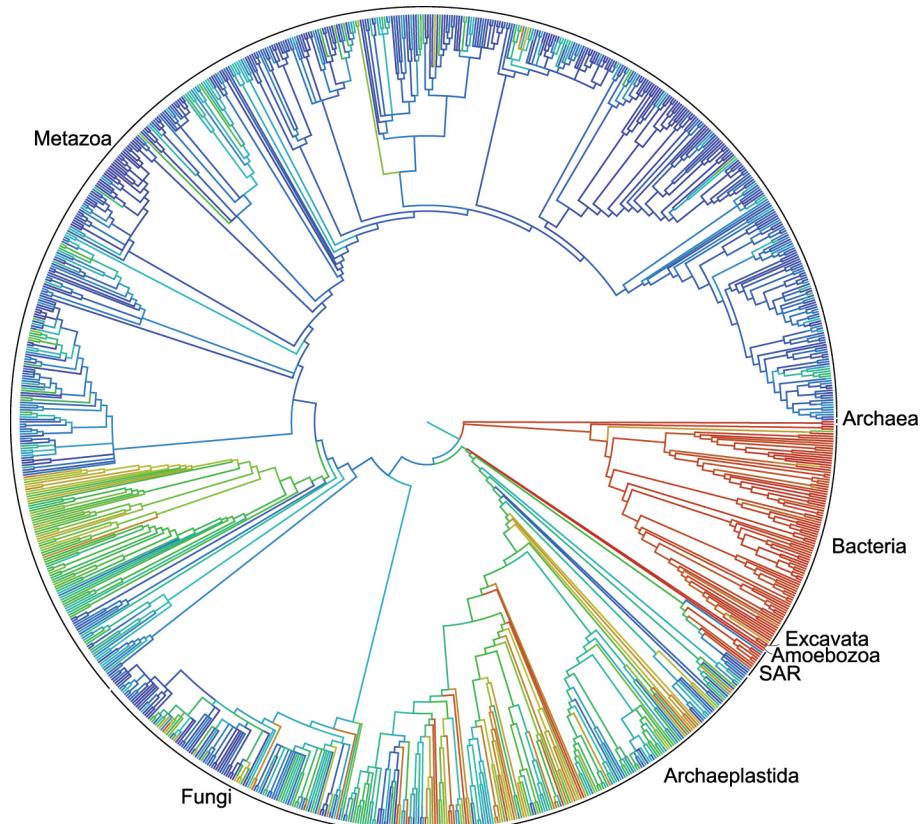
2.1.2 Inferência Filogenética

Embora a evolução das espécies seja estudada desde Darwin, a inferência filogenética possibilita embasar tais hipóteses em dados genéticos, por exemplo, sendo seu principal objetivo obter uma estrutura com as ramificações que representam a história evolutiva de um conjunto de organismos ([ZWICKL, 2006](#)). O resultado de tais análises resulta em uma árvore que representa a relação de parentesco entre diferentes indivíduos, podendo ser extrapolada para a representação de reinos ou domínios da biologia, como mostra a [Figura 2](#). Dada sua relevância para a inferência filogenética definiremos uma árvore conforme [Zwickl \(2006\)](#):

O termo árvore pode ser usado de forma intercambiável com o termo topologia. As pontas da árvore representam dados observados, no qual a inferência é baseada, onde encontram-se os nodos terminais ou folhas, também chamados de táxons. As conexões entre os táxons são chamadas de ramificações ou arestas, para cada qual pode ser atribuído um valor numérico representando seu comprimento. O tamanho das ramificações representam um medida de distância evolutiva entre os nodos, com a unidade dependente do método de inferência. As ramificações também podem ser denominadas bipartição, porque cada um deles divide os táxons em conjuntos distintos (sem sobreposições). Ramificações se cruzam em nodos internos ou vértices, que representam um táxon ancestral hipotético.

Desta forma, pode-se estabelecer que o processo de inferência filogenética consiste em, baseando-se em dados de alinhamentos de sequências, deduzir uma árvore filogenética que seja

Figura 2 – Representação gráfica de uma árvore filogenética em uma tentativa de reconstruir a hipotética Árvore da Vida até o ancestral universal.



Fonte: Hinchliff et al. (2015).

capaz de explicar os relacionamentos entre o conjunto de organismos informados. Contudo, uma metodologia é necessária para identificar árvores plausíveis, dentre todas as combinações possíveis entre os táxons. Tais metodologias são explicadas detalhadamente nas subseções seguintes, e classificadas, segundo Keane (2006), conforme as respectivas abordagens: *a*) baseada em distâncias ou caracteres; e *b*) baseada em métodos estatísticos ou qualitativos.

2.1.2.1 Matrizes de Distâncias

O método de inferência filogenética por matrizes de distâncias, embora não seja mais o estado da arte, continua exercendo importante papel como fundação aos métodos mais complexos existentes atualmente, ao prover uma árvore inicial, que é posteriormente melhorada por métodos mais precisos (GUINDON; GASCUEL, 2003; VINH; Von Haeseler, 2004). Em função de sua eficiência computacional, sendo geralmente mais rápida do que outros métodos baseados em caracteres, a inferência baseada em matrizes de distâncias pode ser facilmente aplicada em análises de diferentes tipos de dados, contanto, que as distâncias entre pares de amostras possam ser calculadas (YANG, 2014). No que diz respeito à sua abordagem, este é um método baseado

em distâncias e também qualitativo, pois seu resultado é sempre uma única árvore, sendo esta, a melhor, seguindo seus critérios.

2.1.2.2 Máxima Parcimônia

Assim como as matrizes de distância, o método de inferência baseado em máxima parcimônia também é capaz de informar uma única melhor árvore com performance otimizada. Este método é baseado no princípio da ‘navalha de Occam’, que afirma que hipóteses devem ser mantidas tão simples quanto possível, ou seja, assumir a menor quantidade de premissas. Segundo Keane (2006), Edwards e Cavalli-Sforza (1963) foram os pioneiros neste método ao propor que a melhor árvore filogenética é aquela que envolve a menor quantidade possível de evolução, implicando que métodos baseados em parcimônia encontrem a árvore com o menor número de substituições e/ou diferenças entre características de um conjunto sob análise. No que diz respeito à sua abordagem, este método pode ser usado tanto através de distâncias como caracteres e é também qualitativo. Contudo, apesar de também ser computacionalmente rápido quando comparado a outros métodos, foi demonstrado que pode ser estatisticamente inconsistente sob certas combinações de árvores, o que pode vir a comprometer seus resultados (FELSENSTEIN, 1978).

2.1.2.3 Máxima Verossimilhança

Diferentemente dos anteriores, o método baseado em máxima verossimilhança, do inglês *maximum likelihood* (ML), adota uma abordagem estatística baseada exclusivamente em caracteres. Introduzido por Felsenstein (1981), este método consiste em uma abordagem estatística para buscar o melhor casamento possível (*best-fit*) entre os dados de alinhamentos de sequências e um sistema de substituição filogenética, maximizando, portanto, a semelhança entre as amostras e um sistema de referência. Embora seja reconhecido por sua precisão, com Hordijk e Gascuel (2005) citando múltiplos exemplos de estudos afirmando que a inferência baseada em ML é capaz de encontrar a melhor árvore filogenética com maior frequência que outros métodos, o cálculo de ML é, no mínimo, NP-difícil, conforme demonstrado por Chor e Tuller (2005). Tal complexidade computacional dificulta seu emprego em análises com grandes números de amostras em alinhamentos de sequências, no entanto, conforme veremos no Capítulo 3, múltiplos aprimoramentos e heurísticas seguem sendo desenvolvidos para melhorar a performance das inferências baseadas em ML.

2.1.2.4 Inferência Bayesiana

Assim como os métodos baseados em máxima verossimilhança, os que se apoiam em inferência bayesiana são classificados por sua abordagem com base em caracteres e em métodos

Tabela 4 – Lista dos mais utilizados sistemas de substituição de sequências moleculares.

Sistema	Bibliografia
JC69	Jukes e Cantor (1969)
K80	Kimura (1980)
F81	Felsenstein (1981)
F84	Yang (1994a), originalmente implementado por Felsenstein no software DNAML a partir de 1984
HKY85	Hasegawa, Yano e Kishino (1984)
TN93	Tamura e Nei (1993)
GTR (REV)	Tavaré (1986)
UNREST	Yang (1994b)

Fonte: Elaborado pelo autor com base em Yang (2014).

estatísticos. Outra semelhança está no fato de que as análises feitas utilizando-se de inferência bayesiana também buscam o melhor casamento possível entre os dados e um sistema de substituição filogenética. Embora também seja baseada em métodos estatísticos, a inferência bayesiana tem como diferença fundamental o fato de que as probabilidades representam o grau de convicção em uma hipótese, ao invés da frequência com que ela ocorre (KEANE, 2006). Nesta abordagem é possível alimentar o conhecimento *a priori*, para que ele conte com informações sobre o processo evolutivo, como o processo de ramificações de Yule (EDWARDS, 1970), e o de nascimento e morte por Rannala e Yang (1996), de forma que estes processos possam exercer influência sobre o resultado da distribuição estatística (YANG, 2014). O motor por baixo de grande parte das ferramentas de inferência bayesiana é o algoritmo Metropolis-Hastings, também conhecido como Markov chain Monte Carlo (MCMC) que, através de uma abordagem baseada em Simulações de Monte Carlo, visita amostras do espaço de busca e, dadas amostragens suficientes, pode ser usada para estimar valores como a média e desvio padrão da distribuição *a posteriori* (NASCIMENTO; REIS; YANG, 2017).

2.1.3 Sistemas de Substituição de Sequências Moleculares

Uma característica comum aos métodos mais modernos de inferência filogenética é o uso explícito de modelos ou sistemas de substituição. Seu uso surge da necessidade de estimar a probabilidade das alterações em um conjunto de sequências genéticas, com cadeias de Markov de tempo contínuo figurando como uma das abordagens mais utilizadas (YANG, 2014). Embora Felsenstein (1988) afirme que todos os métodos filogenéticos dependam de hipóteses, sejam implícitas ou explícitas, sobre os processos de substituição de nucleotídeos e proteínas, somente os métodos baseados em máxima verossimilhança (ML) e inferência bayesiana dependem explicitamente destes sistemas, que nada mais são do que formalizações matemáticas das

hipóteses do método utilizado.

$$Q = \begin{pmatrix} -\mu_A & \mu_{AG} & \mu_{AC} & \mu_{AT} \\ \mu_{GA} & -\mu_G & \mu_{GC} & \mu_{GT} \\ \mu_{CA} & \mu_{CG} & -\mu_C & \mu_{CT} \\ \mu_{TA} & \mu_{TG} & \mu_{TC} & -\mu_T \end{pmatrix} \quad (2.1)$$

Um dos componentes mais importantes em um cadeia de Markov é a matriz de transição, exemplificada na Equação 2.1 por uma matriz de transição para DNA (A,G,C,T), que, no caso dos sistemas de substituição de sequências moleculares costuma indicar a frequência com que as transições de estados ocorrem, assim como a distribuição de ocorrência de cada estado, ilustradas na Figura 3. Neste contexto, os estados são as diferentes proteínas ou nucleotídeos representados pelas possíveis bases, no caso de DNA, como demonstra a Tabela 1. Assim, um dos fatores determinantes de diferenciação entre os sistemas de substituição disponíveis são: *a*) a configuração da sua matriz de transição; *b*) a distribuição de ocorrências de estados; *c*) o número de parâmetros livres utilizados no sistema.

Dada a importância dos métodos de inferência filogenética baseados em ML e Bayes, Keane (2006) afirma que conhecer os processos de substituição e conseguir construir sistemas realísticos é um dos fatores determinantes para habilitar uma correta inferência dos relacionamentos entre espécies. Dentre uma vasta gama de sistemas disponíveis, apresenta-se na Tabela 4 uma lista dos mais populares conforme Yang (2014). Contudo, apesar dos grandes avanços na modelagem do processo evolutivo, existem uma série de falhas conhecidas pela comunidade filogenética e que devem ser cuidadosamente verificadas antes de tomada uma decisão a respeito do sistema a ser usado para qualquer tipo de análise. Keane (2006) fornece uma lista das mais reconhecidas fraquezas presentes nos sistemas:

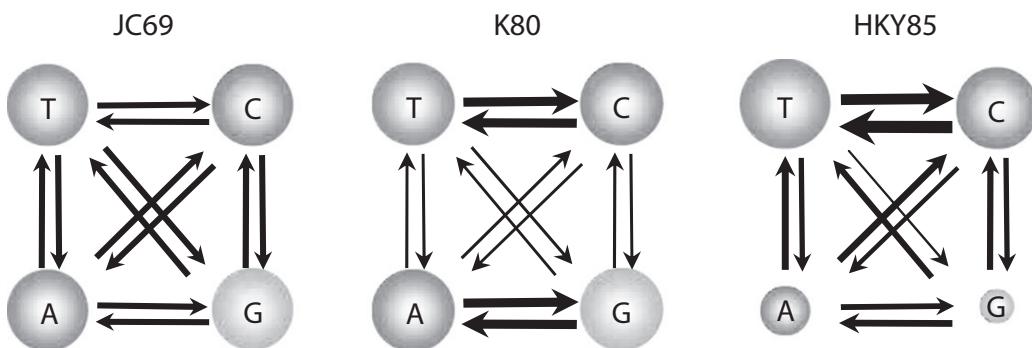
- **Atração por Longas Ramificações (LBA¹):** refere-se à tendência de organismos posicionados ao final de longas ramificações de uma árvore filogenética de parecer artificialmente perto um do outro;
- **Transferência Horizontal de Genes (HGT²):** é um processo em que um organismo transfere material genético para outro organismo que não pertence à sua descendência, o que pode produzir filogenias inesperadas, uma vez que espécies similares ao receptor (mas não envolvidas na transferência) podem ser transferidas para uma parte completamente diferente da árvore, fazendo com que tanto o doador quanto o receptor possam parecer deslocados;
- **Heterotaquia³:** denomina a ocorrência de variações ao longo do tempo na taxa evolutiva

¹do inglês, Long Branch Attraction.

²do inglês, Horizontal Gene Transfer.

³do inglês, Heterotachy.

Figura 3 – Taxas relativas de substituição para três sistemas baseados em cadeias de Markov: a grossura das linhas representa a probabilidade de transição e o tamanho dos círculos representa a distribuição de ocorrências



Fonte: Yang (2014) referenciando os sistemas Jukes e Cantor (1969), Kimura (1980) e Hasegawa, Yano e Kishino (1984), respectivamente.

considerada normal para uma posição de uma sequência molecular, confundindo sistemas que assumem uma taxa evolutiva constante, sendo esta uma premissa que não necessariamente se mantém por todo o histórico de um organismo devido às diferentes pressões evolutivas durante sua história.

Portanto, é fundamental que a escolha de um sistema de substituição para análises filogenéticas seja embasada em dados a respeito das múltiplas sequências moleculares em estudo e que respeite as características da análise, seja ela, a respeito de uma determinada região de interesse ou de toda a sequência genética do organismo; se contempla múltiplas espécies; ou ainda, se abrange um longo período de tempo. Zwickl (2006) justifica a necessidade de extremo cuidado na escolha do sistema afirmando que:

escolhendo um sistema excessivamente simples, perde-se a capacidade de capturar totalmente as nuances do processo de substituição que gerou os dados, reduzindo a precisão da análise; de modo inverso, a escolha de um sistema muito complexo traz consigo o risco de reduzir o poder da inferência no evento em que alguns parâmetros podem contemplar ruído presente nos dados ao invés de verdadeiros sinais filogenéticos; no entanto, ao trabalhar com dados biológicos reais, **qualquer sistema escolhido será inherentemente uma simplificação, de forma que a tarefa é encontrar o melhor sistema possível** ao invés do ‘verdadeiro’ sistema.

2.2 Computação de Alto Desempenho

Com o objetivo de impulsionar a pesquisa e fornecer respostas para os cada vez mais complexos problemas de variados campos do conhecimento, os supercomputadores e clusters estão na vanguarda dos projetos computacionais. Inicialmente projetados com hardware e

software específico, como era o caso dos Cray, na década de 60, ao longo dos anos os sistemas de computação de alto desempenho foram convergindo em suas tecnologias e definindo tendências nesse sentido. Recentemente, o design de supercomputadores e sistemas de alto desempenho tem sido dominado, do ponto de vista de hardware, por componentes “de prateleira”, ou seja, disponíveis para usuários finais, e por um aumento no paralelismo através de processadores com múltiplos cores e GPUs.

Embora este tipo de sistema esteja, classicamente, restrito a centros de pesquisa e universidades com grandes orçamentos, os aprendizados obtidos na construção e operação de supercomputadores têm sido compartilhados com a comunidade em geral através de métodos e algoritmos publicados em revistas científicas. Tais ensinamentos eventualmente alcançam o público em geral e passam, conforme sua utilidade, a se tornar parte da rotina do desenvolvedor de software. *Clusters* caseiros e a computação em nuvem, conforme veremos a seguir, contribuem para expandir o acesso a um alto poder computacional que deve ser explorado através de técnicas eficientes e já testadas, como as oriundas da computação de alto desempenho.

Através de técnicas de computação paralela e distribuída torna-se possível tratar problemas que transcendem a capacidade de um único computador, sendo atualmente uma das formas mais amplamente conhecidas de computação de alto desempenho. O aumento no volume de dados, acompanhado da necessidade de análises cada vez mais profundas fez com que a capacidade de processamento de um único recurso computacional não fosse mais suficiente para atender tais necessidades. A percepção desta mudança deu origem ao termo *Big Data* e fomentou a visão da empresa orientada a dados, o que, como uma profecia autorrealizadora, agravou ainda mais o problema, dando origem à projetos de software como Hadoop, BigQuery, Spark e Kafka, dentre os mais populares.

O surgimento da computação em nuvem mudou radicalmente a forma como se compra computação em um contexto comercial, influenciando, também, os rumos da pesquisa acadêmica. Através do modelo de custos baseado no uso, a computação em nuvem possibilita transformar o investimento capital, que incorre da compra de recursos como servidores, equipamentos de rede e pessoal para configuração, em custo operacional, que varia de acordo com o uso acompanhando altos e baixos no consumo de recursos computacionais. Acessando serviços remotos através da Internet, a computação em nuvem permite acesso ubíquo a recursos computacionais que podem ser rapidamente configurados e requerem gerenciamento mínimo enquanto fornecem ao provedor o benefício das economias de escala.

A elasticidade se posiciona como a principal mudança ocasionada pela computação em nuvem do ponto de vista da computação de alto desempenho, possibilitando o projeto de sistemas que se ajustam conforme a demanda, uma vez que adicionar e remover recursos é uma das funcionalidades oferecidas pelos provedores na forma de uma chamada de função e realizada em questão de minutos. Esta capacidade de auto serviço via Interfaces de Programação de Aplicações (APIs) torna possível a criação e composição de inovadoras soluções computacionais, onde uma aplicação pode alojar e desalocar os recursos necessários para executar determinada

tarefa apenas enquanto ela está sendo executada. Um exemplo deste cenário ocorre no uso de aceleradores gráficos disponibilizados pelos grandes provedores de computação em nuvem na forma de *add-ons* que podem ser atachados a uma máquina virtual (VM) mesmo depois que ela já esteja em uso, desafiando a noção clássica de que um computador contempla um conjunto fixo de recursos definidos em tempo de montagem e *boot* do sistema operacional. O desafio, portanto, passa a ser qual é a melhor maneira de tirar proveito destas capacidades, uma vez que são possíveis do ponto de vista técnico, alinhando interesses técnicos e econômicos em busca de projetos de software que possam se moldar conforme o uso.

Uma nova abordagem à virtualização utilizada pelos provedores de computação em nuvem é apresentada na forma da orquestração de contêineres, que, ao mesmo tempo que fornecem isolamento entre aplicações, são estruturas mais leves do ponto de vista de sobrecarga computacional por utilizarem o paradigma de virtualização à nível de sistema, ao invés de hardware, como é o caso das máquinas virtuais. Através do uso de sistemas de arquivos em camadas, contêineres fornecem, além do isolamento, uma camada de portabilidade ao empacotar, na forma de imagens, tanto a aplicação quanto as suas dependências.

Dentre as mais recentes inovações no contexto de computação em nuvem podemos citar o paradigma FaaS, também chamado de *serverless*, que abstrai do desenvolvedor a maior parte das camadas anteriores ao código enquanto fornece um ambiente padronizado com elasticidade automática embutida. Tal cenário é habilitado pela combinação entre elasticidade de recursos e contêineres, possibilitando o uso de FaaS para execução de códigos pontuais, geralmente limitados em termos da sua duração, e que respondem a eventos. Desta forma, obtém-se um ambiente verdadeiramente pago pelo uso, onde a cobrança só ocorre quando o código está executando, e não enquanto o recurso está disponível, como acontece nas máquinas virtuais. Devido as características como o auto grau de elasticidade e a facilidade em obtê-lo, a cobrança pelo uso e a gestão da maior parte da infraestrutura por parte do provedor de computação em nuvem, o paradigma FaaS vem ganhando espaço no âmbito comercial, embora muitos ainda o considerem imaturo.

2.2.1 Computação Paralela e Distribuída

A computação paralela vem sendo amplamente utilizada como forma de obter, em um tempo razoável, respostas para problemas considerados de alta complexidade. O aumento no volume de dados disponíveis colocou as técnicas de computação paralela e distribuída em maior evidência, uma vez que a demanda por análises cada vez mais detalhadas e descritivas traz benefícios para toda a sociedade (WILKINSON; ALLEN, 1999).

Projetos de computadores paralelos e distribuídos têm grandes demandas no que diz respeito a qualidade dos equipamentos utilizados. Hardware específico com redes confiáveis e de alta velocidade são a norma observada entre os supercomputadores da Top500⁴. Entretanto,

⁴ Top500 é um projeto que divulga os 500 supercomputadores mais poderosos do mundo.

devido aos altos custos de capital e de operação, tais supercomputadores têm sido gradualmente substituídos por *clusters* de computadores comuns (BAKER; BUYYA, 1999).

Os benefícios da computação paralela e distribuída certamente compensam as dificuldades em construir e operar um *cluster* ou um supercomputador, no entanto tais dificuldades oneram também a construção das aplicações que farão uso destes recursos. Desta forma, os supercomputadores e suas aplicações tendem a ter um alto nível de acoplamento, o que torna tanto o comportamento quanto a performance das aplicações dependente do hardware, da plataforma, das características de rede e de outras particularidades, limitando a escalabilidade (RAJAN et al., 2011).

Fornecendo uma base sólida para novos desenvolvimentos, os mais importantes padrões de projeto para computação paralela e distribuída, na visão de Wilkinson e Allen (1999), são brevemente descritos abaixo.

- **Embaraçosamente paralelos:** Caracteriza problemas que podem ser imediatamente decompostos em unidades independentes e balanceadas, requerendo pouco ou nenhum esforço para a divisão das tarefas e a completa independência entre as unidades;
- **Divisão e conquista:** Consiste em reduzir um problema em partes sucessivamente menores, de forma que estas possam ser tratadas de forma individual, com maior facilidade. Análoga a recursão em algoritmos sequenciais, a divisão e conquista pode se manifestar na *decomposição de domínio*, ou na *decomposição funcional*;
- **Pipelines:** Apropriada para problemas parcialmente sequenciais, esta técnica divide a computação em etapas que devem ser completadas uma após a outra com o paralelismo se manifestando na forma de múltiplas unidades de execução para cada uma das etapas, também chamadas de estágios do pipeline;
- **Fases paralelas:** Neste modelo, também conhecido por *Bulk Synchronous Parallel* (BSP) os vários processos que correm em paralelo devem, em um determinado momento, chegar a uma barreira onde, obrigatoriamente, devem se comunicar, compartilhando informações de estado, por exemplo, e esperar os demais processos atingirem este mesmo ponto para, só então, prosseguir com sua computação;

2.2.2 Computação em Nuvem

A computação em nuvem apresenta um novo paradigma computacional trazendo consigo uma forma diferenciada de se pensar em Tecnologia da Informação no que diz respeito a aquisição e manutenção de hardware e datacenters. Esta pode ser vista como a combinação entre um conjunto de tecnologias já estabelecidas e conhecidas, mas que juntas possibilitaram a criação de um modelo de operação mais flexível que o tradicional datacenter (ZHANG; CHENG; BOUTABA, 2010).

Através da Internet, a computação em nuvem provê acesso a infraestrutura e recursos computacionais em escala massiva e com estrutura de custos similar a uma companhia de eletricidade, realizando a ideia teorizada por John McCarthy nos anos de 1960, a computação como um serviço⁵ (ZHANG; CHENG; BOUTABA, 2010; SULEIMAN, 2012). Empresas não precisam mais investir grandes capitais em hardware e infraestrutura para implantar seus serviços e nem dos custos operacionais para mantê-los, pois através da computação em nuvem estes custos são diluídos como despesas operacionais e crescem conforme o uso (MARTIN et al., 2011).

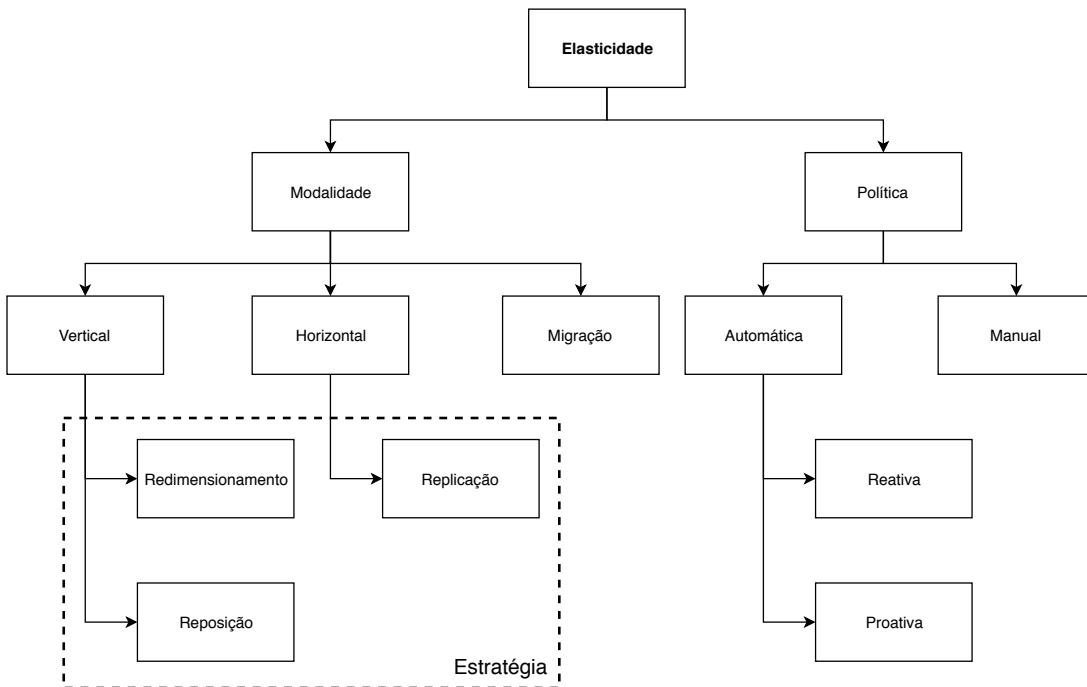
Armbrust et al. (2010), por meio de estudos em conjunto com os maiores provedores de computação em nuvem, definem as principais características deste paradigma: *a) ilusão de recursos computacionais infinitos, disponíveis sob demanda, deste modo eliminando a necessidade de provisionar capacidade; b) dispensa investimento de capital, permitindo começar com pouco e aumentar a alocação de recursos, conforme as necessidades e o ritmo de crescimento; e c) pagar pelo uso (por exemplo, servidores por hora e armazenamento por dia), recompensando, desta forma, a conservação de recursos.*

Por ser relativamente jovem e uma combinação entre tecnologias, a obtenção de uma definição amplamente aceita para a computação em nuvem foi um desafio, pois cada autor favorece as características que mais lhe convém. No entanto a elasticidade e o modelo de custos são consenso, formando a base para que o *National Institute of Standards and Technology* (NIST)⁶ a defina como:

Infraestrutura com capacidades de aprovisionamento de recursos de forma rápida e elástica, em alguns casos automática, para aumentar e diminuir o número de recursos. Para o usuário, tais capacidades muitas vezes parecem ser ilimitadas, podendo ser realizadas em qualquer quantidade e a qualquer momento.

Pode-se concluir que a computação em nuvem, por meio da combinação de tecnologias, atinge o objetivo de prover recursos computacionais como um serviço, democratizando e simplificando a computação em larga escala (AWADA; BARKER, 2017). Através da virtualização, é capaz de simular capacidade infinita e reduzir drasticamente os tempos de provisionamento, fornecendo diversos benefícios ao mesmo tempo que impõe novos desafios (ZHANG; CHENG; BOUTABA, 2010), requerendo, por exemplo, esforços de migração de sistemas, devido a natureza orientada a servidores de aplicações existentes, que torna-se uma significativa barreira de entrada para um grande número de usuários (JONAS et al., 2017).

Figura 4 – Taxonomia da Elasticidade apresentando quais são as Modalidades e Políticas disponíveis além das diferentes estratégias de elasticidade.



Fonte: Adaptado de Rodrigues (2016), Coutinho et al. (2015), Righi (2013) e Galante e Bona (2012).

2.2.3 Elasticidade

Dentre as características que definem a computação em nuvem e, talvez, o que efetivamente diferencia este paradigma frente a outras alternativas (GALANTE; BONA, 2012), a elasticidade, que diz respeito à capacidade de adicionar e remover recursos computacionais de forma automatizada e com um tempo de provisionamento na casa dos minutos, ao invés de semanas, é a propriedade que permite aos usuários casar a demanda com a alocação de recursos (ARMBRUST et al., 2010; RAVEENDRAN; BICER; AGRAWAL, 2011; IMAI; CHESTNA; VARELA, 2012; SULEIMAN, 2012).

Segundo Raveendran, Bicer e Agrawal (2011), projetos de aplicações paralelas e distribuídas tradicionalmente contemplavam um número fixo de recursos, efetivamente definindo um prazo de validade para o investimento, pois esta classe de aplicações costuma ser muito sensível às características do ambiente. Tipicamente, recursos computacionais como processamento, rede e armazenamento são finitos e estáticos, sendo inicialmente adquiridos através de uma estimativa da capacidade de pico na esperança de que a carga média fique abaixo desta estimativa. No entanto, enquanto os recursos são estáticos, a demanda é dinâmica (MARSHALL; KEAHEY; FREEMAN, 2010).

Enquanto a alocação estática tenta provisionar recursos sempre para o pior caso, correndo

⁵ Do inglês: *utility computing*.

⁶ Instituto Nacional de Padrões e Tecnologia, uma agência governamental americana.

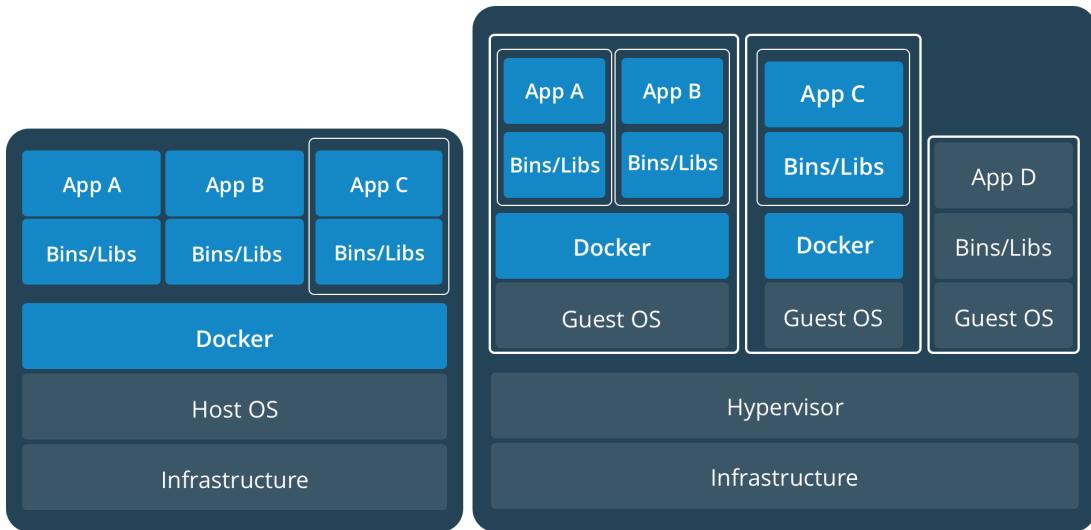
o risco de errar para mais ou para menos, a alocação elástica depende do comportamento da aplicação, definido através de métricas e níveis de serviço contratados (RIGHI, 2013). Os benefícios da alocação elástica se manifestam para todos os envolvidos, onde o usuário consegue manter o desempenho da sua aplicação a um preço justo e o administrador da nuvem otimiza o uso de seus recursos, repassando para outros usuários a capacidade ociosa (RAVEENDRAN; BICER; AGRAWAL, 2011; RIGHI, 2013; HENNESSY; PATTERSON, 2013).

Diferentes provedores e middlewares gerenciam elasticidade da forma que melhor lhes convém, o que torna necessária uma classificação destas estratégias para compreender suas diferenças e, com isso, os pontos fortes e fracos de cada estratégia. Tomando por base as classificações realizadas por Coutinho et al. (2015), Righi (2013) e Galante e Bona (2012), ilustradas na Figura 4, pode-se categorizar as estratégias de elasticidade segundo as seguintes dimensões:

- **Modalidade:** *a)* vertical: onde substitui-se um recurso por outro com atributos redimensionados, como CPU ou memória, por exemplo; ou *b)* horizontal: onde é alterado o número de recursos, incluindo novas cópias idênticas para apoiar o atual, por exemplo; ou *c)* migração: onde a VM é realocada para outro servidor com diferentes características, sendo útil para cenários de redução na carga de trabalho, consolidando recursos computacionais.
- **Política:** *a)* manual: onde as ações de elasticidade dependem de impulsos externos à plataforma; ou *b)* automática: onde a própria plataforma decide e executa as ações de elasticidade que podem ser, por sua vez, (i) reativas: através de um mecanismo de regra-condição-ação; ou (ii) proativas: que, através de análises preditivas, executa ações de elasticidade visando antecipar a carga.
- **Estratégia:** *a)* replicação: onde cópias de uma mesma máquina virtual são utilizadas para contribuir com o processamento; *b)* reposição: onde uma máquina em execução é levada para outro servidor substituto, visando, por exemplo, aumento ou redução de recursos; ou *c)* redimensionamento: onde atributos específicos da máquina virtual são modificados para influenciar no desempenho da computação.

A precificação dinâmica, um subproduto do uso elástico de recursos computacionais, possibilita a criação de mercados que negociam a capacidade ociosa dos provedores de nuvem (*spot market*) a um preço mais vantajoso, sendo desejável para uma ampla categoria de aplicações científicas e comerciais. Contudo, esta possibilidade deixa de ser utilizada por tornar ainda mais complexo o cálculo de retorno no investimento em um ambiente de nuvem e porque aplicações não estão preparadas para lidar com a preempção de recursos (que ocorre quando há exaustão da capacidade ociosa) (JONAS et al., 2017). Neste momento, o corpo de conhecimento da computação paralela e distribuída se torna um aliado, uma vez que um dos requisitos para a elasticidade é uma boa escalabilidade, ou seja, a capacidade de uma aplicação manter sua eficiência e aumentar o consumo de recursos proporcionalmente ao tamanho do processamento (GALANTE; BONA,

Figura 5 – Diagrama ilustrando as principais diferenças no que tange às estratégias de virtualização baseadas em Contêineres (esquerda) e Máquinas Virtuais (direita)



Fonte: Reproduzido de https://www.docker.com/what-container#/virtual_machines.

2012; HENNESSY; PATTERSON, 2013). Por fim fornece, também, estratégias para recuperação (*self-healing*) e alta disponibilidade.

Apesar dos benefícios promovidos pela elasticidade, realizar seu potencial é uma tarefa não trivial devido aos desafios oriundos do desenvolvimento de aplicações aptas a usufruir destas capacidades (RAVEENDRAN; BICER; AGRAWAL, 2011; LOFF; GARCIA, 2014). Embora se observe uma mudança nos padrões de aplicações distribuídas em direção a um maior uso de elasticidade (SHANKAR et al., 2018), aproveitar de maneira eficiente as vantagens dessa característica permanece um desafio mesmo para usuários sofisticados (JONAS et al., 2017).

2.2.4 Orquestração de Contêineres

Contêineres fornecem isolamento entre aplicações através de mecanismos mais leves que a virtualização completa como ocorre nas máquinas virtuais (VMs) (AWADA; BARKER, 2017). Sua grande vantagem está no fato de que um contêiner, através de uma imagem, guarda um ambiente empacotado, auto-contido e pronto para transporte e publicação de uma aplicação (PAHL, 2015), incluindo, se necessário, bibliotecas e/ou binários de *middlewares* dos quais o software é dependente (AWADA; BARKER, 2017). Essa propriedade favorece a portabilidade do código e é, conforme notado por Higgins, Holmes e Venters (2015), especialmente relevante em um contexto de computação em *grids* remotos, onde o ambiente pode carecer das dependências necessárias para suportar um determinado *job*.

Ainda assim, contêineres são um tipo de virtualização, neste caso, chamada de virtualização à nível de sistema (TOSATTO; RUIU; ATTANASIO, 2015) em contraste com a virtualização à nível de hardware, representada pelas VMs, e que incorre uma alta sobrecarga de processa-

mento. Um dos diferenciais da virtualização por contêineres, e que fornece grandes ganhos de performance, é o seu tratamento do sistema de arquivos por camadas, onde existe permissão de escrita apenas a camada do topo e as demais são composições somente para leitura das camadas restantes, agrupadas em formato de pilha (HIGGINS; HOLMES; VENTERS, 2015).

Contudo, Pahl (2015) pondera que apesar de compartilhar a categoria virtualização, estas duas abordagens solucionam problemas distintos devido aos componentes envolvidos em cada arquitetura, conforme mostra a Figura 5. Embora máquinas virtuais forneçam um ótimo isolamento e abstração do hardware, o custo necessário para atingir este objetivo é significativo, exigindo uma cópia completa do sistema operacional, a interceptação e tradução de chamadas de sistema e drivers específicos para cada virtualizador (FELTER et al., 2015). Em contrapartida, contêineres fazem uso de diretivas de isolamento e gestão de recursos nativos dos sistemas operacionais, reduzindo significativamente o *overhead* enquanto relaxam as garantias de abstração. Apesar de parecer uma troca vantajosa, em um ambiente de computação em nuvem é comum que ambas as estratégias sejam utilizadas em conjunto devido às necessidades de isolamento de recursos que os provedores devem assegurar entre diferentes clientes (BERNSTEIN, 2014).

Uma das principais implementações de virtualização à nível de sistema é o projeto Docker⁷ que vem definindo os rumos para abordagens concorrentes. Seu funcionamento é, assim como boa parte dos projetos similares, baseado em duas funcionalidades do *kernel* do Linux, nomeadas: *cgroups* e *namespaces* (PAHL, 2015). A partir da sua combinação se torna-se possível isolar aplicações, dando a impressão de que cada uma delas têm o sistema operacional dedicado para si, e limitar o consumo de recursos, especificando de antemão os limites aplicáveis a cada software que roda dentro de um contêiner (TOSATTO; RUIU; ATTANASIO, 2015). Para especificar o empacotamento é usado um arquivo chamado *dockerfile*, que detalha os elementos que compõem a imagem, assim como as instruções para sua construção.

Embora diversos provedores de computação em nuvem forneçam diretivas para ajustar a quantidade de recursos à demanda, chamadas de *auto-scaling* (MAO; HUMPHREY, 2011), seu aproveitamento e potencial de economia são limitados pelo delay com que VMs são adicionadas e removidas ao *pool* de recursos (SHANKAR et al., 2018). Contêineres, por sua vez, devido à sua portabilidade e baixa sobrecarga computacional, costumam ser utilizados para melhorar a utilização de recursos e possibilitar uma densidade de processos mais alta, o que levou à popularização de ferramentas de *deployment* baseadas em contêineres, com diversos projetos comerciais e acadêmicos se propondo a gerir ambientes baseados em contêineres.

Projetos como o Omega (SCHWARZKOPF et al., 2013) e Borg (VERMA et al., 2015) que, eventualmente, levaram ao Kubernetes⁸ (BURNS et al., 2016), todos oriundos do Google, além do Mesos⁹ (HINDMAN et al., 2011), do Twitter, são propostas para facilitar a gestão de ambientes

⁷ Disponível na URL <https://www.docker.com>, sem publicações.

⁸ Principal orquestrador de contêineres atualmente e que traça suas origens ao projeto de gestão de *datacenters* Borg, nunca publicamente revelado, mas que influenciou diversas iniciativas (como o próprio Mesos), disponível na URL <https://kubernetes.io>.

⁹ Projeto originalmente criado para gestão de *datacenters* mas que tem se articulado para tornar-se um produto centrado em contêineres, disponível na URL <http://mesos.apache.org>.

baseados em contêineres, seja localmente ou em um ambiente de computação em nuvem. Ao mesmo tempo os principais provedores anunciam, também, suas respectivas implementações, fornecendo plataformas que possibilitam aos usuários criar *clusters* e gerenciar instâncias de contêineres de maneira automatizada visando maximizar o aproveitamento de recursos como memória e CPU ou prover alta disponibilidade. Sendo os mais populares: Elastic Container Service (ECS¹⁰) da AWS, Azure Kubernetes Service (AKS¹¹) da Microsoft e o Kubernetes Engine¹² na própria Google Cloud Platform (GCP).

2.2.5 Serverless e FaaS (Function as a Service)

Considerado o próximo passo na evolução de tecnologias oriundas da computação em nuvem, o paradigma *Function as a Service* (FaaS), também conhecido como *serverless*, surgiu como iniciativa dos provedores de computação em nuvem, estreando com o AWS Lambda em 2014, ao fornecer uma nova camada de abstração focada em melhorar o comportamento de elasticidade e aumentar a granularidade dos intervalos de cobrança por recursos utilizados. O principal fator de diferenciação para esta emergente arquitetura de desenvolvimento de aplicações está, segundo Wang et al. (2018), no fato de que ela isola do usuário tudo o que diz respeito a gestão de servidores, originando o nome *serverless*.

Por ser uma abordagem relativamente nova, ainda não há consenso quanto a definição precisa de FaaS embora diversos autores concordem no que diz respeito aos pontos centrais. Este cenário pode ser ilustrado através de citações provenientes de três fontes diferentes: Wang et al. (2018), Shankar et al. (2018) e o Google Cloud Platform (GCP).

componentes pequenos, autônomos e *stateless* dedicados a lidar com tarefas específicas sendo, na maioria dos casos, um pequeno bloco de código escrito em linguagem de *scripting* que têm seus ambientes de execução e servidores gerenciados pelo provedor de computação em nuvem, que aloca recursos de forma dinâmica para garantir sua escalabilidade e disponibilidade. (WANG et al., 2018)

provê usuários com acesso instantâneo a uma grande quantidade de recursos computacionais sem o esforço necessário para a gestão de *clusters* possibilitando executar funções orientados a eventos, e *stateless* que incluem restrições sobre o uso de memória e tempo de execução sobre cada invocação. (SHANKAR et al., 2018)

¹⁰ Disponível na URL <https://aws.amazon.com/ecs>.

¹¹ Disponível na URL <https://azure.microsoft.com/en-us/services/kubernetes-service>

¹² Disponível na URL <https://cloud.google.com/kubernetes-engine>

Tabela 5 – Comparativo das principais características nas ofertas de FaaS por parte de três grandes provedores de computação em nuvem.

	AWS '18	AWS '17	Azure	GCP
Memória (MB)	$64 \times k$ ($k = 2, 3, \dots, 47$)	$64 \times k$ ($k = 2, 3, \dots, 24$)	$128 \times k$ ($k = 2, 3, \dots, 12$)	$128 \times k$ ($k = 2, 4, 8, 16$)
Tempo Limite de Execução (minutos)	15	5	10	9
CPU	Proporcional a Memória	Proporcional a Memória	Fixo	$200 \text{ MHz} \times j$ ($j = 1, 2, 4, 7, 12$)
Sistema Operacional	Amazon Linux	Amazon Linux	Windows 10	Debian 8
Espaço em Disco (MB)	512	512	500	> 512
Fatores de Cobrança	Tempo de Execução, Memória Alocada	Tempo de Execução, Memória Alocada	Tempo de Execução, Memória Consumida	Tempo de Execução, Memória Alocada, CPU Alocada

Fonte: Atualizado pelo autor, adaptado de Wang et al. (2018).

uma solução computacional leve¹³, assíncrona e baseada em eventos que permite a criação de pequenas funções de finalidade única que respondem a eventos na nuvem sem a necessidade de gerenciar servidores ou ambientes de execução¹⁴. (GOOGLE, 2018)

Comparando-se ao paradigma mais popular de computação em nuvem, o *Infrastructure as a Service* (IaaS) que é baseado no aluguel de máquinas virtuais cobradas por hora, as FaaS tem como uma das suas maiores diferenças os limites de memória, tempo de execução e, principalmente, o modelo de cobrança, destacados na Tabela 5. Apesar da computação em nuvem permitir que o hardware dedicado local seja substituído por recursos pagos pelo uso e dinamicamente alocados, Eivy (2017) destaca que a cobrança não deixa de ser baseada na alocação, ainda que por hora, ao invés do uso efetivo, fazendo com que o consumidor pague mais do que o necessário por recursos alocados, mas não necessariamente utilizados na sua completude.

Desta forma, o modelo de cobrança FaaS apresenta uma evolução ao cobrar baseado no tempo de execução de uma função (ROBERTS; FOWLER, 2016; WANG et al., 2018), geralmente arredondado para a casa das centenas de milissegundos ($100ms$), o que significa que o usuário só paga pelo tempo em que seu código está efetivamente executando. Contudo, os operadores de computação em nuvem cobram seu preço pelas funcionalidades disponíveis no modelo FaaS,

¹³lightweight

¹⁴runtime environment

com valores contendo um ágio de aproximadamente $2x$, segundo Jonas et al. (2017), quando comparados com VMs de capacidade similar, o que ainda configura um preço aceitável para este autor, uma vez que a cobrança ocorre de forma mais granular, a elasticidade e paralelismo massivos e o fato de que não é incomum encontrar *clusters* utilizando apenas 50% de suas capacidades.

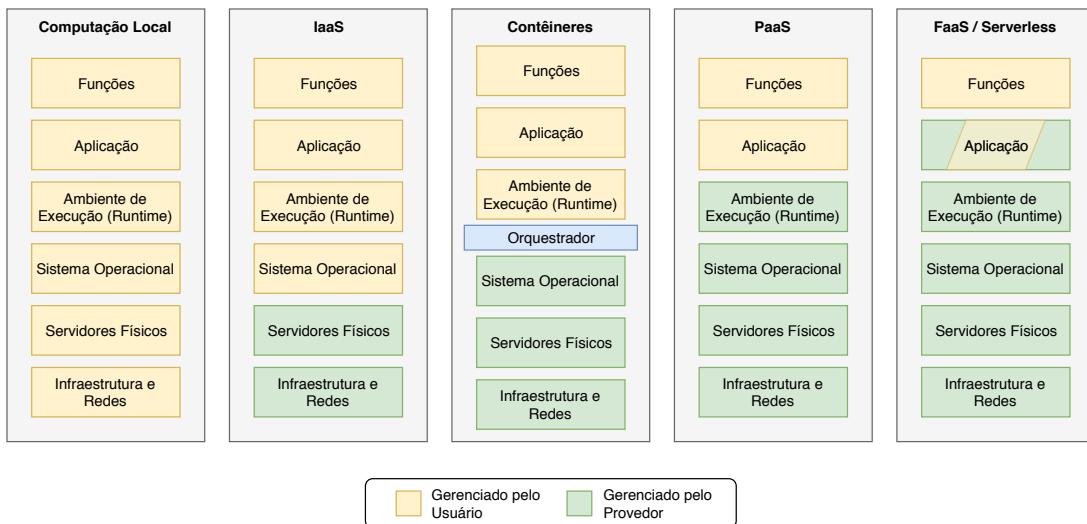
Segundo Wang et al. (2018), uma execução de FaaS geralmente ocorre em um tipo de *sandbox*, seja um contêiner de função ou uma máquina virtual, que impõem limites sobre fatores como memória, tempo de CPU e tempo máximo de execução. Desta forma, uma instância da função só será criada quando houver uma solicitação de execução e será congelada imediatamente após seu término, uma visão que é compartilhada por Roberts e Fowler (2016) e que atribuem este comportamento ao fato de que os contêineres computacionais que executam as funções são efêmeros, criados e destruídos pelo provedor puramente em função da demanda.

Um ponto amplamente divulgado pelos provedores de FaaS e endossado pela academia é o fato de que as execuções de funções são fundamentalmente elásticas, com cada invocação sendo potencialmente executada por um processo paralelo e distribuído, independente e isolado dos demais em função da natureza *stateless* das funções (SPOIALA, 2017; JONAS et al., 2017; ROBERTS; FOWLER, 2016; JONAS et al., 2017). Apesar de existirem, ao contrário do que sugere o nome, servidores rodando o código submetido, estes não são preocupações do ponto de vista do desenvolvedor (SPOIALA, 2017), que se abstém de tarefas como instalação, otimização e atualização de sistemas operacionais, *runtimes* e hardware. Desta forma a gestão do ambiente e da elasticidade é transferida para o provedor de computação em nuvem, escalando tais funções de maneira automática e dependendo apenas da quantidade de recursos disponíveis em sua plataforma e a demanda de uso por seus clientes (HAN, 2017).

Através deste comportamento inherentemente elástico torna-se possível tirar proveito da elasticidade de uma maneira simplificada, algo que continua desafiador dependendo das características de aplicações e *middlewares*, muitos desenhados originalmente para hardware dedicado (JONAS et al., 2017). Do ponto de vista da produtividade do desenvolvedor, o emprego de FaaS permite focar apenas nos seus fatores de diferenciação (HAN, 2017), sem preocupar-se com gestão de infraestrutura, *runtimes* e elasticidade, onde Jonas et al. (2017) ponderam que não é difícil imaginar um cenário em que o objetivo do usuário não é necessariamente obter a melhor performance possível, mas sim um ganho capaz apenas de superar seus recursos locais sem demandar grande esforço de desenvolvimento. Ademais, o modelo *serverless* se encaixa bem no contexto do desenvolvimento moderno de aplicações, evoluindo, como mostra a figura Figura 6, para partes reaproveitáveis cada vez menores (conhecidas como *microservices*) que, juntas, compõem a aplicação, contrastando com a construção de aplicações monolítica que se mostram difíceis de gerenciar e escalar (EIVY, 2017).

Além dos benefícios para o desenvolvimento, Jonas et al. (2017) argumenta que o modelo *serverless* possibilita a criação de sistemas de processamento distribuídos radicalmente mais simples, fundamentalmente elásticos e mais amigáveis do ponto de vista do usuário, caracte-

Figura 6 – Diagrama representando a divisão de responsabilidades sobre as diversas camadas que compõem um ambiente computacional contrastando ambientes locais com diferentes modelos de serviço habilitados pela computação em nuvem (IaaS, PaaS, Contêineres, e FaaS).



Fonte: Adaptado de [Spoiala \(2017\)](#).

rísticas estas, interessantes para a aplicação na pesquisa acadêmica. [Spillner, Mateos e Monge \(2017\)](#) afirmam, endossando esta visão, que a natureza do provisionamento verdadeiramente sob-demanda, aliada ao modelo de cobrança, torna as plataformas de FaaS atrativas para tarefas de pesquisa, apresentando ainda uma lista (complementada aqui com dados de [Roberts e Fowler \(2016\)](#)) de implementações do modelo FaaS. Dentre elas:

- **Comerciais:** AWS Lambda, Google Cloud Functions, Azure Functions, Oracle Functions, IBM OpenWhisk;
- **Código Aberto:** Docker-LambCI, Effe, OpenLambda, Fission, OpenFaaS, Kubeless, IronFunctions;
- **Middlewares:** Serverless, Claudia.js, AWS Serverless Application Model, Architect.

Outro fator relevante das FaaS está relacionado aos ganhos para o provedor de computação em nuvem, com [Eivy \(2017\)](#) afirmando que ao oferecer uma plataforma como esta o provedor tem maior controle sob o hardware e software utilizados, possivelmente unificando tudo aquilo que diz respeito às versões, sistemas operacionais, atualizações de segurança e ferramental interno. Além disso, a natureza *stateless* e de curta duração destas funções facilita a gestão e aumenta a eficiência no aproveitamento de recursos ao garantir a rotatividade necessária para multiplexar clientes e balancear a demanda à oferta de recursos, um fator primordial para a rentabilidade do provedor de computação em nuvem ([SHANKAR et al., 2018](#)).

Apesar dos grandes benefícios no que diz respeito ao modelo de cobrança, amplo paralelismo e a elasticidade nativa, [Han \(2017\)](#) realiza uma necessária análise dos pontos negativos inerentes

ao modelo FaaS, como: *a*) Redução na Transparência: assim como qualquer abstração, certas partes do sistema saem do controle do desenvolvedor, neste caso, o fato do provedor gerenciar a infraestrutura pode dificultar a investigação e solução de problemas; *b*) Dificuldade de Depuração: como toda a execução se dá no provedor de nuvem e em um ambiente restrito as opções de *debug* são limitadas e ainda muito dependente de *logs* e *traces* de execução; *c*) Elasticidade de Custo: proporcionalmente a elasticidade de recursos, os custos pelo uso de FaaS podem potencialmente sair de controle devido a sua natureza virtualmente infinita de recursos, causando surpresas na hora da cobrança.

A opacidade do modelo computacional possibilitado pelas FaaS também é destacada por Wang et al. (2018) como um impeditivo relevante para adoção por determinadas categorias de usuários, como os que trabalham em ambientes altamente regulados, devido às questões que levanta sobre a qualidade do isolamento entre *tenants*, resistência à ataques e o controle de custos. Corroborando com este último ponto, se destaca a publicação de Eivy (2017), que dedica parte significativa ao cálculo e explicação do modelo de cobrança *serverless*. No que diz respeito às dificuldades no desenvolvimento, autores como Jonas et al. (2017) expressam uma grande preocupação quanto à maturidade das ferramentas para *debug*, uma situação que só se agrava uma vez que o modelo FaaS incentiva o uso de componentes distribuídos e provê amplo paralelismo, complicando a análise de *logs* e *traces* de auditoria.

Ainda no que diz respeito a maturidade deste modelo computacional, são encontrados argumentos que sugerem um longo caminho de desenvolvimento pela frente, seja por falta de ferramental apropriado, reduzido corpo de pesquisa acadêmico (comprovado pela falta de publicações e a vasta quantidade de referências para blogs e páginas web em (WANG et al., 2018), por exemplo) e, até mesmo, pela falta de uma definição precisa e autoritativa acerca dos conceitos de FaaS e *serverless*, sendo ambos utilizados de maneira intercambiável na indústria. Atualizações frequentes, por parte dos provedores de computação em nuvem, também comprovam a existência de lacunas no paradigma, com novas funcionalidades e correções figurando frequentemente nos *releases* de versões.

Contudo, apesar das fraquezas expostas, evidências informais apontam que arquiteturas baseadas em FaaS vieram para ficar, devido aos relatos de empresas e indivíduos reportando casos de grandes economias e melhorias de performance através da sua adoção. No meio acadêmico, os trabalhos de Jonas et al. (2017) e Shankar et al. (2018) contribuem para este cenário encorajador ao atingir, respectivamente, performances na casa de 40 TFLOP/s com 2800 unidades de execução e 4 TFLOP/s com 180 unidades de execução, enquanto trabalhos como o de Wang et al. (2018) fornecem um panorama sobre as características e estratégias utilizadas pelos principais provedores de computação em nuvem nas suas ofertas de FaaS.

3 TRABALHOS RELACIONADOS

*“A fool thinks himself to be wise,
but a wise man knows himself to be a fool.”*

William Shakespeare

Neste capítulo apresenta-se um levantamento da literatura para a compreensão do estado da arte, apresentando trabalhos relacionados assim como suas contribuições para a área de bioinformática aplicada à filogenética e a computação de alto desempenho. Apresenta-se também a estratégia aplicada no levantamento da literatura, as motivações e as justificativas para a estratégia escolhida. Além de delinear as fontes utilizadas e as questões de pesquisa, destacando as palavras-chaves empregadas e os critérios utilizados para inclusão dos resultados.

Em seguida, elaborou-se uma taxonomia para melhor categorizar e classificar os trabalhos encontrados neste levantamento, para então, realizar uma análise comparativa entre eles e um comparativo estatístico. É de fundamental interesse avaliar os trabalhos em termos do emprego de técnicas de computação de alto desempenho, resultando na [Tabela 11](#), que apresenta uma visão consolidada dos trabalhos enquanto categorizados pela taxonomia e avaliados em termos das técnicas de computação de alto desempenho.

Por fim, delineou-se o cenário encontrado em termos das suas lacunas no que diz respeito à computação de alto desempenho, principalmente no emprego de computação em nuvem e elasticidade de recursos, estratégias de balanceamento de carga e uso de aceleração por GPUs.

3.1 Metodologia de Seleção de Trabalhos

Com o intuito de mapear o estado da arte, no que diz respeito a ferramentas computacionais relacionadas à filogenética, esta dissertação buscou inicialmente uma estratégia baseada em mapeamento sistemático. Contudo, particularidades da área estudada representaram grandes desafios para a execução de uma análise satisfatória, podendo ser citadas neste sentido as seguintes dificuldades:

- **Datas de publicação:** Apesar dos grandes avanços em tempos recentes, no que diz respeito a tecnologias de sequenciamento genético e algoritmos, as bases que suportam estes avanços permanecem as mesmas e muito populares entre pesquisadores, de forma que uma restrição do tipo “ano de publicação” implica em descartar uma parte relevante do corpo de conhecimento da filogenética e seus alicerces. Exemplos que corroboram com esta asserção podem ser encontrados na [Subseção 2.1.3 \(Sistemas de Substituição de Sequências Moleculares\)](#), tendo boa parte das bases estabelecidas na década de 80 e com trabalhos inciais, e até hoje relevantes, iniciando em 1969;

Tabela 6 – Palavras-chaves utilizadas na composição dos termos de busca desta revisão da literatura.

Código	Palavra-chave
K1	Filogenética
K2	Software
KO1	Inferência
KO2	Cloud

Fonte: Elaborado pelo autor.

Tabela 7 – Critérios de inclusão no conjunto de dados para trabalhos encontrados nesta revisão da literatura.

Código	Filtro
F1	Apenas documentos com texto completo disponível
F2	Publicados em revistas ou periódicos com revisão por pares
F3	Dissertações e Teses
F4	Escritos em Inglês

Fonte: Elaborado pelo autor.

- **Baixa aderência a revistas especializadas:** Outra característica da intersecção entre computação e filogenética é a baixa aderência das publicações a revistas especializadas, de maneira que projetos de software são muitas vezes publicados em revistas da biologia, como por exemplo *Systematic Biology*, *Nucleic Acids Research* e *Synthetic Biology*. De modo inverso, revistas especializadas em bioinformática acabam recebendo submissões de trabalhos que apenas utilizam softwares para este fim, sem necessariamente apresentar contribuições no ferramental. Exemplos deste cenário incluem as revistas *Bioinformatics* e *BMC Bioinformatics*. Para a computação, a situação é agravada pelo reduzido número de publicações em veículos já estabelecidos como o *IEEE* e *ACM*;
- **Ausência de divulgação das ferramentas:** Por fim, um fator adicional é a maneira como diversos artigos apresentam suas contribuições do ponto de vista computacional e ferramental, sendo este, em casos extremos, relegado a algumas frases nas seções de metodologia informando que um software foi desenvolvido para auxiliar a pesquisa. Embora este comportamento seja compreensível em situações onde as contribuições para a biologia e a filogenética sejam expressivas, não deixa de ser frustrante para o pesquisador ao conduzir um mapeamento sistemático, pois causa a rejeição de produções nesta situação logo nos primeiros estágios de filtro de relevância em função da ausência desta informação em pontos de destaque como título, abstract e até mesmo conclusão, por exemplo.

Diante destas dificuldades e o volume de artigos encontrados em estágios iniciais da abordagem sistemática¹, a estratégia para o mapeamento do estado da arte passou a ser uma revisão de literatura. Apesar da mudança na estratégia do levantamento, boa parte dos passos anteriores seguiram relevantes, dentre eles, o estabelecimento das questões de pesquisa, das palavras-chaves

¹ Utilizando-se dos bancos de dados disponíveis para a biblioteca da UNISINOS foram encontrados cerca de 7500 artigos no primeiro estágio do mapeamento sistemático utilizando o termo de busca (phylogeny OR phylogenetics OR phylogenetic) AND (software OR tool OR program OR package), adicionadamente aos filtros: somente resultados que contemplem texto completo disponível, publicados em revistas com revisão por pares, datados a partir de 1990 e com idioma inglês.

Tabela 8 – Editoras e fontes utilizadas na elaboração da presente revisão de literatura.

Fonte	Endereço
Association for Computing Machinery (ACM)	https://dl.acm.org/
BioMed Central (BMC)	https://www.biomedcentral.com/
CiteSeerX †	https://citeseerx.ist.psu.edu/
Elsevier	https://www.sciencedirect.com/
Google Scholar †	https://scholar.google.com/
Institute of Electrical and Electronics Engineers (IEEE)	https://ieeexplore.ieee.org/
Nature	https://www.nature.com/
Oxford Academic	https://academic.oup.com
Public Library of Science (PLOS)	https://www.plos.org/
Semantic Scholar †	https://www.semanticscholar.org/
Springer	https://link.springer.com/

† não publicam material original mas servem como agregadores do conteúdo de diversas fontes

Fonte: Elaborado pelo autor.

e dos demais filtros, apresentados a seguir. Buscou-se com este trabalho encontrar o estado da arte situado na intersecção entre computação de alto desempenho e bioinformática, mais precisamente estudos filogenéticos, o que resultou nas seguintes questões de pesquisa:

Questão 1: Quais técnicas de otimização são usadas atualmente para possibilitar a inferência de grandes filogenias?

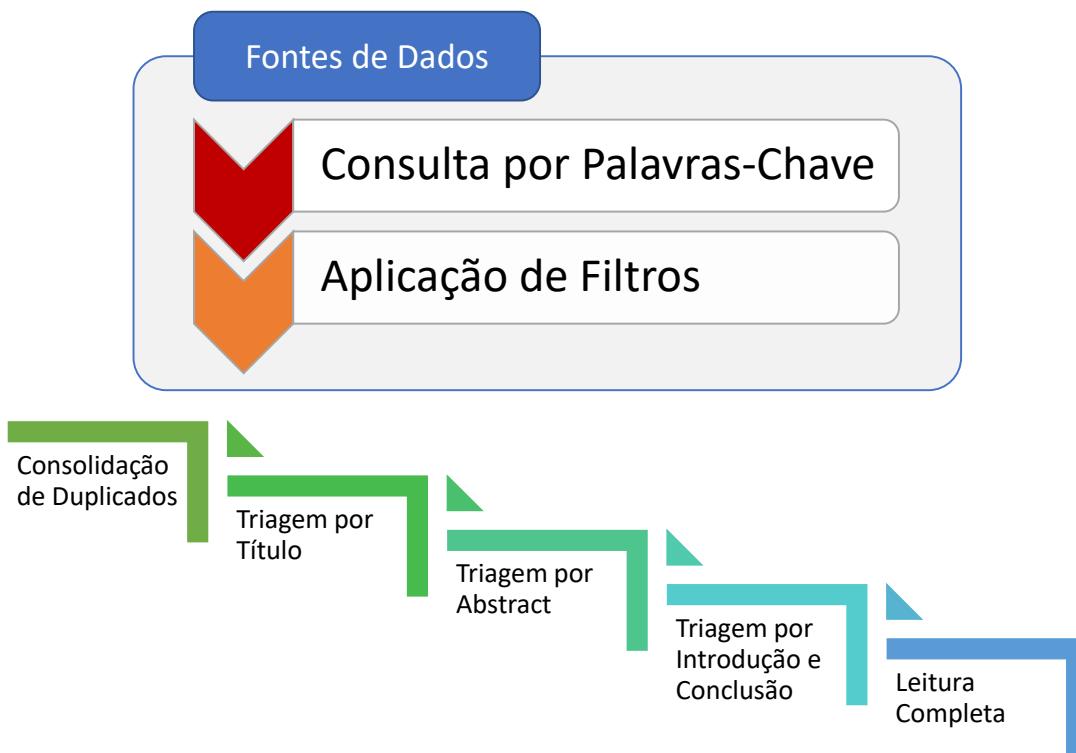
Questão 2: Qual é o cenário no que diz respeito a adoção de técnicas de computação paralela e distribuída para inferência filogenética?

Questão 3: Quais são os desafios encontrados pelos autores e as oportunidades disponíveis para avançar o estado da arte no que diz respeito a computação paralela e distribuída aplicada à algoritmos filogenéticos?

Uma vez definidas as questões de pesquisa, o próximo passo para a execução do mapeamento consiste em estabelecer as palavras-chaves mais apropriadas para esta pesquisa, além de definir filtros adicionais, também chamados de critérios de inclusão, afim de aprimorar a qualidade dos resultados e formular a *string* de busca.

No que diz respeito às palavras-chaves foram definidos três termos fundamentais e dois opcionais, demonstrados na Tabela 6. A existência de termos opcionais mostrou-se necessária em função da grande variação entre as diferentes bases de dados, conforme abordado anteriormente

Figura 7 – Diagrama representando a estrutura de busca e triagem utilizada neste trabalho para a revisão de literatura.



Fonte: Elaborado pelo autor.

nas dificuldades do mapeamento sistemático. Os filtros detalhados na [Tabela 7](#) visam estabelecer critérios de qualidade a respeito do material utilizado neste mapeamento e seguem boas práticas, com a notável omissão de um limite quanto a data de publicação. Esta ausência ocorre também em função das particularidades encontradas no mapeamento.

A execução das buscas teve como objetivo a cobertura de um amplo leque de editoras especializadas em ciência da computação e bioinformática, como: *Association for Computing Machinery* (ACM), *Institute of Electrical and Electronics Engineers* (IEEE), *BioMed Central* (BMC) e *Public Library of Science* (PLOS), assim como veículos de ampla cobertura, como Elsevier, Nature e Springer, além de agregadores como o *Google Scholar*, com a lista completa disponível na [Tabela 8](#).

A elaboração das *strings* de busca varia de acordo com as capacidades de cada plataforma, com preferência para formatos baseados em expressões booleanas compostas pela combinação das palavras-chaves básicas e opcionais por operadores de conjunção lógica (E) incluindo termos sinônimos em operadores de disjunção, por exemplo:

```

(phylogeny OR phylogenetics OR phylogenetic)
AND (software OR tool OR program OR package)
AND (inference)
AND (cloud)
  
```

Em casos nos quais a busca resultava em poucos ou nenhum resultado, as palavras-chaves opcionais eram removidas da expressão. Já a aplicação dos filtros era realizada na própria plataforma de busca, quando esta fornecia tais capacidades ou realizada de maneira manual, caso contrário.

As próximas etapas do levantamento bibliográfico consistiram na avaliação manual de cada resultado para garantir sua relevância no contexto deste trabalho, em consonância com suas questões de pesquisa. Para tanto, adotou-se uma abordagem estruturada, conforme [Figura 7](#), que compôs o *pipeline* de seleção, composta pelos seguintes passos: I) aplicação dos filtros (caso necessário); II) consolidação de duplicados; III) triagem baseada no título; IV) triagem baseada no abstract; V) triagem por leitura da introdução e conclusão; e VI) triagem por leitura completa.

3.2 Taxonomia Elaborada

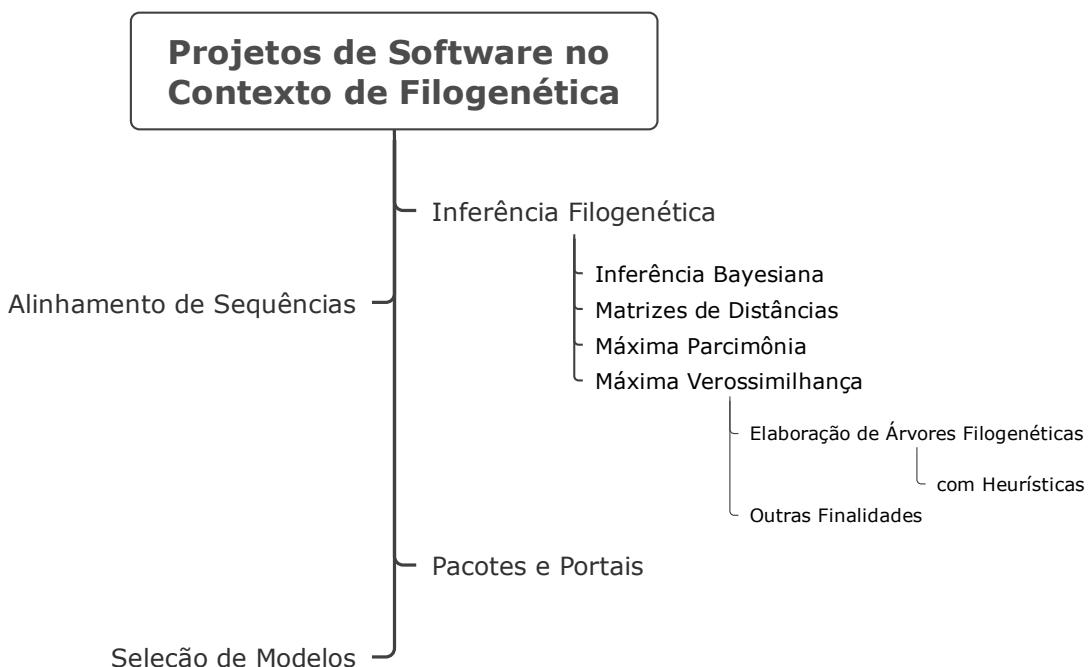
Ao analisar o corpo de dados obtido através do levantamento bibliográfico, o objetivo é responder às questões de pesquisa previamente estabelecidas. Contudo, devido à amplitude da área de conhecimento e ao elevado número de resultados, a elaboração de uma taxonomia que se proponha a ajudar na classificação, comparação e avaliação dentre os múltiplos projetos de software encontrados no campo da filogenética foi julgada relevante para o escopo desta dissertação assim como contribuição.

Embora muito se fale em taxonomias no contexto de filogenética, estas se referem, sua grande maioria, à descrição, denominação e classificação de organismos vivos ([YANG, 2014](#)), não devendo ser confundida com a proposta deste trabalho. O objetivo buscado aqui consiste em uma abordagem hierárquica capaz de representar um corpo de conhecimento de um determinado campo de estudo e que também pode ser visualizado como uma árvore ([DENNING; ROSEN-BLOOM, 2009](#)). Adicionalmente, as pesquisas realizadas durante o levantamento bibliográfico não contemplavam qualquer tipo de estrutura formal dos projetos de computação aplicados à filogenética, de maneira que a taxonomia aqui proposta pode ser considerada um esforço inicial neste sentido.

No entanto, ao invés de construir uma taxonomia teórica e então aplicá-la na classificação de projetos existentes, foi adotada uma abordagem prática: construção da taxonomia com base nos resultados obtidos através do levantamento bibliográfico e, assim como ([PERERA et al., 2013](#)), identificando características, técnicas, funcionalidades ou abordagens empregadas pelos projetos de software contemplados no levantamento. Tal abordagem é relevante uma vez que, para o melhor do conhecimento do autor, não foram encontradas nenhuma outra taxonomia ou classificação similar que contemplasse a intersecção entre filogenética e bioinformática com a computação de alto desempenho.

Diante desta situação, e conforme pode ser visto na [Figura 8](#), foi formalizada uma taxonomia que classifica os trabalhos encontrados em distintas categorias e em uma estrutura hierárquica que contempla as principais características de cada trabalho. Esta taxonomia pode ser dividida

Figura 8 – Taxonomia elaborada para a classificação dos trabalhos identificados durante a etapa de revisão de literatura.



Fonte: Elaborado pelo autor.

em quatro pilares principais: *a) Inferência Filogenética; b) Pacotes e Portais; c) Seleção de Modelos de Substituição Molecular; e d) Alinhamento de Sequências Moleculares.*

Ademais, o pilar que trata de inferência filogenética foi adicionalmente dividido em termos dos métodos utilizados e de suas finalidades, de acordo com os conceitos apresentados na [Subseção 2.1.2. Estatísticas quanto ao número de trabalhos encontrados para cada nível da taxonomia são encontradas na Tabela 9.](#)

3.3 Análise do Estado da Arte

Uma vez executado o levantamento previamente descrito, apresenta-se, nesta seção, a lista de trabalhos selecionados para análise do estado da arte. Ao todo foram encontrados 83 trabalhos relevantes, todos eles situados na intersecção entre inferência filogenética e a ciência da computação, contemplando temas como: *a) Alinhamento de Sequências Moleculares; b) Seleção de Sistemas de Substituição Molecular; c) Pacotes e Portais; e d) Inferência Filogenética.*

Do ponto de vista da filogenética é interessante notar a grande quantidade de ferramentas disponíveis para inferência filogenética, contudo há uma notável predominância de softwares que utilizam métodos baseados em ML ([Subsubseção 2.1.2.3](#)). Outra estatística que se destaca é a grande quantidade de ferramentas classificadas como ‘Pacotes e Portais’, categoria que contempla os trabalhos que não se limitam a um único método de inferência ou que também incluem funcionalidades das demais categorias.

Tabela 9 – Quantidade de trabalhos da revisão de literatura quando classificados conforme as categorias presentes na taxonomia proposta

Categoria	Método	Finalidade	Quantidade
Inferência Filogenética	Máxima Verossimilhança	Outras Finalidades	22
		Elaboração de Árvores	17
		Filogenéticas com Heurísticas	5
	Inferência Bayesiana		3
Pacotes e Portais	Matrizes de Distâncias		3
	Máxima Parcimônia		2
Total			83

Fonte: Elaborado pelo autor.

Tais projetos buscam centralizar boa parte das tarefas envolvidas no processo de análise filogenética, sendo notáveis exemplos os projetos CIPRES (MILLER; PFEIFFER; SCHWARTZ, 2010), GALAXY (AFGAN et al., 2016), MEGA (KUMAR; STECHER; TAMURA, 2016), e Phylemon (TÁRRAGA et al., 2007; SÁNCHEZ et al., 2011). Além de pacotes de software que moldaram os rumos da bioinformática no contexto da filogenética, como é o caso do PHYLIP de Felsenstein (1989), autor que consta na lista da revista Nature dos 100 mais citados (Van Noorden; MAHER; NUZZO, 2014).

Em tratando-se de características computacionais, o objetivo desta pesquisa foi realizar um mapeamento das capacidades dos softwares no que diz respeito a computação paralela e distribuída e, principalmente, a elasticidade de recursos. Desta forma, buscou-se nas publicações e, quando disponível, nas páginas e documentações dos projetos, informações que pudessem indicar o suporte à características como:

Característica 1: Suporte à diretivas de computação distribuída (excetuando-se paralelismo na própria máquina, através de processadores *multi-core*, por exemplo);

Característica 2: Suporte à elasticidade no consumo de recursos e, de maneira mais ampla, suporte a contextos de *cloud computing*;

Característica 3: Implementação de diretivas para平衡amento de carga, seja no contexto de *cloud* ou de computação distribuída;

Característica 4: Suporte à computação baseada em placas gráficas, *Graphics Processing Unit* (GPU), que costuma ser muito eficaz em contextos científicos;

Característica 5: Modelo de interação com o usuário, caso o software suporte comandos interativos ou disponha de uma interface gráfica.

De acordo com os critérios estabelecidos é possível montar um panorama do estado da arte no que diz respeito às ferramentas para filogenética inseridas no contexto de computação em nuvem e elasticidade, sumarizado na Tabela 10. Apesar do amplo suporte à interfaces gráficas ou consoles interativos, presente em praticamente 60% dos softwares encontrados, apenas 23% deles incluem algum tipo de suporte à computação distribuída, sendo fornecido, na grande maioria dos casos, por implementações baseadas em *Message Passing Interface* (MPI), com o projeto TNT, por Goloboff e Catalano (2016), destacando-se como a única exceção encontrada ao trabalhar com *Parallel Virtual Machine* (PVM), um predecessor do MPI.

Em tratando-se de técnicas mais recentes, como a computação baseada em GPU, apenas quatro trabalhos alegam suporte, sendo eles: *a*) o projeto GALAXY (AFGAN et al., 2016) através de seus *plug-ins*; *b*) os softwares BEAST (DRUMMOND et al., 2002) e mrBayes (HUELSENBECK; RONQUIST, 2001), que aceleram na GPU as computações do algoritmo Metropolis-Hastings, também conhecido como Markov chain Monte Carlo (MCMC); e *c*) METAPIGA (HELAERS; MILINKOVITCH, 2010) completa a lista, tirando proveito da GPU para executar uma heurística para a inferência filogenética baseada em algoritmos genéticos.

No campo da elasticidade de recursos e da computação em nuvem a situação é agravada, com apenas três dos trabalhos anunciando suporte. Apesar de portais como CIPRES (MILLER; PFEIFFER; SCHWARTZ, 2010), Phylemon (TÁRRAGA et al., 2007; SÁNCHEZ et al., 2011) e Phylogeny.fr (DEREPPER et al., 2008) alegarem elasticidade, não foram encontrados indícios de suporte à nível de algoritmo, sendo a elasticidade, nestes casos, referente à capacidade de adequar os recursos consumidos ao volume de usuários ativos nas plataformas, também é importante notar a abstenção de suporte à elasticidade automática, com ajustes ficando a cargo dos operadores dos portais supracitados, o que na opinião do autor anula os benefícios de um ambiente baseado na computação em nuvem.

Tabela 10 – Quantidade de trabalhos da revisão de literatura quando classificados de acordo com as características relevantes para a computação distribuída.

Projetos	Console Interativo	Interface Gráfica	Computação Distribuída	Aceleração por GPU	Elasticidade de Recursos	Balanceamento de Carga
83	45	34	19	4	3	2
	54%	41%	23%	5%	4%	2%

Fonte: Elaborado pelo autor.

Merecem destaque no que diz respeito a elasticidade de recursos os trabalhos de Keane et al. (2005) e Keane, Naughton e McInerney (2007) que através dos projetos DPRML e, posteriormente, MultiPhyl apresentam uma plataforma de inferência filogenética nos moldes de projetos como Folding@home (SHIRTS; PANDE, 2000; LARSON et al., 2002) e SETI@home (KORPELA et al., 2001), utilizando recursos ociosos de computadores pessoais. Apesar de não figurar explicitamente como computação em nuvem, devido ao uso de computadores das universidades ao invés de provedores como *Amazon Web Services* (AWS), Azure ou Google Cloud, projetos como estes enfrentam os mesmos desafios técnicos, do ponto de vista da elasticidade, de um projeto *cloud native*. Completa a lista o portal GALAXY (AFGAN et al., 2016), ao fornecer, através dos seus *add-ons* algoritmos que tiram proveito de ambientes de nuvem, além de oferecer imagens compatíveis com provedores como AWS (VARIA; MATHEW, 2017), OpenStack (SEFRAOUI; AISSAOUI; ELEULDJ, 2012) e OpenNebula (MILOJIČIĆ; LLORENTE; MONTERO, 2011). Tais imagens carregam todos os componentes do software empacotados e configurados para se beneficiar ao máximo da elasticidade presente nestes contextos, facilitando a gestão e configuração dos ambientes.

A fim de apresentar uma visão consolidada entre os temas abordados nesta seção elaborou-se a **Tabela 11**, que combina as características relevantes para a filogenética e a bioinformática, oriundas da taxonomia apresentada na **Figura 8**, com os atributos mais relevantes para a computação distribuída conforme elencados na **Tabela 10**. Ao conciliar os dados referentes a estes dois pontos de vista, filogenético e da computação distribuída, chegou-se, enfim, ao estado da arte em termos de produção bibliográfica pertencente a esta intersecção, possibilitando, em última análise, a identificação de pontos fortes e fracos desta área do conhecimento, sendo estes pontos fracos o objetivo de estudo da próxima seção.

3.4 Lacunas de Pesquisa

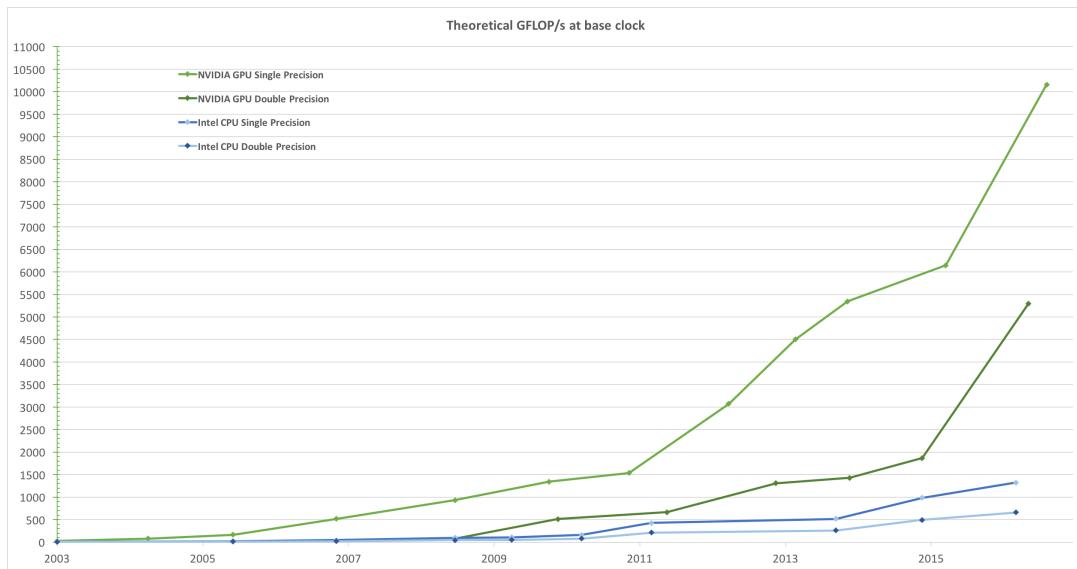
De posse do estado da arte e compreendendo melhor o contexto em que se inserem as ferramentas computacionais utilizadas na filogenética, pode-se encontrar pontos de melhoria e novas abordagens capazes de, entre outros, acelerar o processamento de dados e a quantidade de amostras, aumentando a eficiência dos programas e dos pesquisadores da bioinformática e contribuindo, em última instância, ao avanço na qualidade da produção científica. Motivados por este objetivo e munidos de conhecimentos oriundos da computação paralela e distribuída (RIGHI, 2013; RIGHI et al., 2016, 2017) foram identificadas três lacunas de pesquisa no contexto deste trabalho, nomeadamente:

Lacuna 1: Aceleração por GPUs;

Lacuna 2: Balanceamento de carga;

Lacuna 3: Elasticidade na utilização de recursos.

Figura 9 – Comparativo entre a evolução do poder computacional das CPUs versus GPUs medido em GFLOP/s.



Fonte: [NVIDIA \(2018\)](#).

Apesar do foco de investimentos em melhorias na capacidade computacional tenha permanecido por muitos anos na CPU, este cenário já não é o mesmo atualmente, com as GPUs ganhando relevância em termos de poder computacional aliado a um custo atrativo. Conforme pode ser visto na Figura 9, GPUs já estão muito a frente no que diz respeito ao poder bruto de processamento, sendo especialmente relevantes para contextos científicos (AJI et al., 2013). Dada a natureza dos cálculos de inferência filogenética, a aplicação de GPUs se apresenta como uma ótima combinação haja vista que o alto grau de paralelismo permite explorar, em múltiplas frentes de simulação, o espaço de busca da melhor árvore filogenética que representa os dados de entrada. Um problema similar é a descida de gradientes aplicada na área da Inteligência Artificial (IA) e que já tira proveito da GPU para tal, vide [Zhang et al. \(2013\)](#).

Ainda que quase um quarto dos softwares analisados disponham de iniciativas que apoiam a computação distribuída, poucos foram aqueles que mencionaram estratégias de balanceamento de carga entre seus atributos técnicos. Embora não seja estritamente necessário para obter acesso às melhorias de performance oriundas da computação distribuída, a aplicação de balanceamento de carga assegura um melhor aproveitamento dos recursos disponíveis, aumentando a eficiência e potencialmente reduzindo ainda mais o tempo de execução. Através de revisões como a apresentada por [Mishra, Sahoo e Parida \(2018\)](#), foram encontradas diversas sugestões de algoritmos e estratégias para balanceamento de carga, muitas das quais certamente poderiam ser aplicadas com resultados positivos neste contexto.

Outra abordagem com baixa penetração dentre os projetos encontrados é a elasticidade de recursos, abordada na Subseção 2.2.3, característica que permite ao software se ajustar a quantidade de recursos disponível de maneira dinâmica, seja para mais ou para menos, sem que seja necessário interromper sua execução. Menção honrosa neste sentido são os trabalhos de

Keane et al. (2005) e Keane, Naughton e McInerney (2007), que tornaram os projetos DPRML e MultiPhyl aptos a trabalharem com recursos variáveis ainda que sem estar inserido em um contexto de computação em nuvem. Especulou-se que a baixa adoção a estratégias de elasticidade deve-se em parte ao modelo de investimento onde instituições investem os recursos oriundos dos financiamentos e bolsas em *clusters*, e estes são renovados periodicamente de maneira que o modelo de uso está sempre baseado na substituição, ao invés da complementação, através da adição de novos recursos. Contudo, acredita-se que os incentivos financeiros a adoção da elasticidade, aliados à maior participação dos provedores de computação em nuvem no meio acadêmico, farão com que aumente a relevância de projetos já preparados a tirar proveito destas capacidades.

Tabela 11 – Comparativo entre trabalhos encontrados na revisão de literatura classificados de acordo com a taxonomia proposta e atributos de interesse
(continua)

Projeto	Citação	Atributo					
		Console Interativo	Interface Gráfica	Computação Distribuída	Aceleração por GPU	Elastici-dade de Recursos	Balancea-mento de Carga
Alinhamento de Sequências Moleculares							
Clustal Omega	Sievers et al. (2014)						
MUSCLE	Edgar (2004)						
segminator	Archer et al. (2010)	✓		✓			
T-Coffee	Notredame et al. (2000)						
Seleção de Sistemas de Substituição							
CoMET	Lee et al. (2006)	✓		✓			
Concaterpillar	Leigh et al. (2008)				✓		
DTModSel	Minin et al. (2003)						
jmodeltest	Posada (2008)	✓		✓			
jmodeltest2	Darriba et al. (2012)	✓		✓		✓	
kakusan	Tanabe (2011)	✓					
modelgenerator	Keane et al. (2006)						
modeltest	Posada e Crandall (1998)						
MrModeltest	Nylander (2004)						
ProtTest	Abascal, Zardoya e Posada (2005)	✓		✓		✓	
Pacotes e Portais							
ARB	Ludwig et al. (2004)	✓		✓			
Bio++	Dutheil et al. (2006)						
BOSQUE	Ramírez-Flandes e Ulloa (2008)	✓		✓			
CIPRES	Miller, Pfeiffer e Schwartz (2010)			✓		✓	
DAMBE	Xia (2017)	✓		✓			
EMBOSS	Rice, Longden e Bleasby (2000)	✓					
GALAXY	Afgan et al. (2016)	✓		✓		✓	✓
MEGA	Kumar, Stecher e Tamura (2016)	✓		✓			
mesquite	Maddison e Maddison (2018)	✓		✓			
PAL	Drummond e Strimmer (2001)						
paup*	Swofford (2002)	✓		✓			
Phylemon	Tárraga et al. (2007)	✓		✓			
Phylemon 2.0	Sánchez et al. (2011)	✓		✓			
PHYLIP	Felsenstein (1989)	✓					

Tabela 11 – Comparativo entre trabalhos encontrados na revisão de literatura classificados de acordo com a taxonomia proposta e atributos de interesse
(continuação)

Projeto	Citação	Atributo					
		Console Interativo	Interface Gráfica	Computação Distribuída	Aceleração por GPU	Elasticidade de Recursos	Balanceamento de Carga
phylo_win	Galtier, Gouy e Gautier (1996)	✓	✓				
Phylogeny.fr	Dereeper et al. (2008)		✓				
SeaView	Gouy, Guindon e Gascuel (2010)	✓	✓				
<i>Inferência Filogenética por Inferência Bayesiana</i>							
BEAST	Drummond et al. (2002)	✓	✓			✓	
mrBayes	Huelsenbeck e Ronquist (2001)	✓			✓	✓	
PhyloBayes	Lartillot e Philippe (2004)						
<i>Inferência Filogenética por Matrizes de Distâncias</i>							
BioNJ	Gascuel (1997)	✓					
QuickTree	Howe, Bateman e Durbin (2002)						
TreeFit	Kalinowski (2009)	✓	✓				
<i>Inferência Filogenética por Máxima Parcimônia</i>							
PhyloNET	Than, Ruths e Nakhleh (2008)						
TNT	Goloboff e Catalano (2016)	✓	✓		✓		
<i>Inferência Filogenética por Máxima Verossimilhança focada em Elaboração de Árvores Filogenéticas</i>							
ALIFRITZ	Fleissner, Metzler e Von Haeseler (2005)						
DNAML	Felsenstein e Churchill (1996)	✓					
DPRML	Keane et al. (2005)	✓	✓	✓		✓	✓
fastDNAml	Olsen et al. (1994)	✓		✓			
fasttree	Price, Dehal e Arkin (2009)			✓			
IQPNNI	Vinh e Von Haeseler (2004)	✓		✓			
IQ-TREE	Nguyen et al. (2015)			✓			
MultiPhyl	Keane, Naughton e McInerney (2007)			✓		✓	✓
nhPhyML	Boussau e Gouy (2006)	✓					
PhyML	Guindon et al. (2010)	✓			✓		
PHYML-aLRT	Anisimova e Gascuel (2006)	✓					
PhyML-Multi	Boussau, Guéguen e Gouy (2009)	✓					
PROCOV	Wang et al. (2007)						
RAxML	Stamatakis (2014)		✓		✓		
splitstree	Huson e Bryant (2006)	✓	✓				
treefinder	Jobb, Von Haeseler e Strimmer (2004)	✓	✓				

Tabela 11 – Comparativo entre trabalhos encontrados na revisão de literatura classificados de acordo com a taxonomia proposta e atributos de interesse
(continuação)

Projeto	Citação	Atributo						
		Console Interativo	Interface Gráfica	Computação Distribuída	Aceleração por GPU	Elasticidade de Recursos	Balanceamento de Carga	
tree-puzzle	Schmidt et al. (2002)	✓			✓			
<i>Inferência Filogenética por Máxima Verossimilhança focada em Elaboração de Árvores Filogenéticas utilizando Heurísticas</i>								
GARLI	Zwickl (2006)				✓			
METAPIGA	Helaers e Milinkovitch (2010)	✓	✓		✓	✓		
PhyloCoco	Catanzaro, Pesenti e Milinkovitch (2008)	✓		✓				
SEMPHY	Friedman et al. (2002)							
SSA	Salter e Pearl (2001)							
<i>Inferência Filogenética por Máxima Verossimilhança com Outras Finalidades</i>								
CodeAxe	Saunders e Green (2007)							
CONSEL	Shimodaira e Hasegawa (2001)							
DNArates	Maidak et al. (1994)							
EDIBLE	Massingham e Goldman (2000)							
EREM	Carmel et al. (2010)							
GZ-gamma	Gu e Zhang (1997)	✓						
HyPhy	Kosakovsky Pond, Frost e Muse (2005)	✓	✓		✓			
McRate	Mayrose, Mitchell e Pupko (2005)							
mixturetree	Chen, Rosenberg e Lindsay (2011)							
MOLPHY	Adachi e Hasegawa (1996)							
PAML	Yang (2007)			✓				
PARAT	Meyer e Von Haeseler (2003)	✓		✓				
passml	Liò et al. (1998)	✓						
PhyNav	Le Vinh, Schmidt e Haeseler (2005)							
PHYSIG	Blomberg, Garland e Ives (2003)							
PRAP	Müller (2004)	✓		✓				
rate4site	Mayrose et al. (2004)							
SeqState	Müller (2005)	✓		✓				
SIMMAP	Bollback (2006)	✓		✓				
simplot	Lole et al. (1999)	✓		✓				
SLR	Massingham e Goldman (2005)							
TipDate	Rambaut (2000)							
		Total	45	34	19	4	3	2

Fonte: Elaborado pelo autor.

4 MODELO He-LASTIC

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

Leslie Lamport

Este capítulo apresenta uma proposta capaz de preencher algumas das lacunas de pesquisa encontradas a partir do levantamento bibliográfico realizado. Batizado de He-lastic, o modelo tem como objetivo empregar elasticidade de recursos, habilitada através do uso da computação em nuvem, em uma estrutura de duas camadas, com a primeira responsável por absorver tarefas de curta duração e que demandam maior adaptabilidade à curva de carga do sistema, enquanto a segunda camada trata tarefas de média e longa duração, lançando mão de estratégias já consolidadas para o tratamento da elasticidade.

O modelo proposto se inspira no elemento da tabela periódica Hélio (He) por características que vão além do nome, traçando um paralelo ao fato de que o elemento químico em questão tem número atômico 2, ou seja, é composto por dois prótons em seu núcleo, assim como o modelo proposto que, de maneira análoga, é constituído por dois componentes centrais, suas camadas de elasticidade. Assim como os balões de gás Hélio transmitem uma sensação de leveza devido a sua densidade menor que a do ar, o modelo He-lastic busca manter esta associação através do reduzido *overhead* decorrente do seu uso.

Inicialmente será detalhada a estratégia e a motivação para escolher uma das aplicações dentre as que foram estudadas durante a análise do estado da arte (Tabela 11) e, que por sua vez, se tornará a referência sobre a qual serão aplicadas as propostas do modelo He-lastic. Em seguida são apresentadas as características do problema que ela trata, quais fatores influenciam na carga computacional, no grão de paralelismo e quais estratégias podem ser utilizadas para obter os maiores ganhos quando aplicando técnicas de elasticidade, obtendo como resultado desta análise uma lista com sete decisões que norteiam o projeto e sua arquitetura.

Na sequência, a arquitetura proposta é detalhada com base nas características e decisões tomadas, discutindo em profundidade a respeito dos elementos constituintes do modelo, sua interação com o usuário, com o ambiente computacional e sua importância no que diz respeito a

aplicação de técnicas de elasticidade de recursos computacionais, com especial atenção no que tange a motivação para a adoção de uma arquitetura baseada em duas camadas de elasticidade e os benefícios esperados pelo emprego deste arranjo.

Por fim, a metodologia de avaliação é apresentada detalhando as estratégias empregadas para comprovar a eficiência do modelo proposto em relação ao aproveitamento de recursos e o custo associado à execução das análises realizadas pela aplicação alvo, ou seja, o cálculo de adequação (*best-fit*) de sistemas de substituição de sequências moleculares.

4.1 Seleção da Aplicação

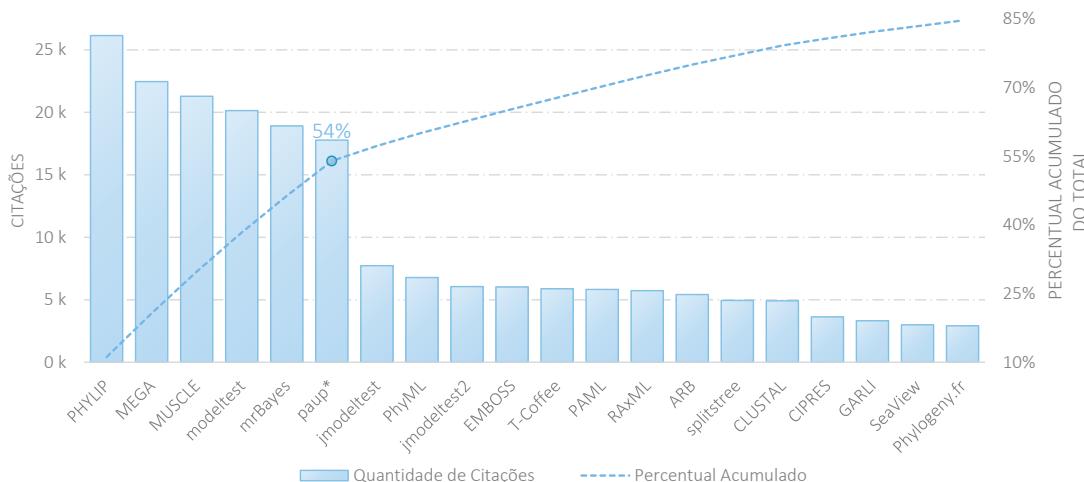
Uma vez identificado o estado da arte no que diz respeito à intersecção das áreas de computação de alto desempenho e bioinformática, mais precisamente filogenética, foi verificada a existência de lacunas, detalhadas na [Seção 3.4](#), quanto às técnicas computacionais modernas, como por exemplo: aceleração por GPUs, balanceamento de carga e o uso de recursos de computação em nuvem. Embora todas as lacunas encontradas sejam relevantes para o aumento da eficiência dos projetos de software, o modelo He–lastic estreita seu foco na questão da elasticidade de recursos computacionais. Desta forma, o presente trabalho objetiva modernizar projetos relevantes para a bioinformática, assim como, realizar ganhos de eficiência e possíveis reduções de custos, através do desenvolvimento de um modelo que seja capaz de validar a hipótese de que a computação em nuvem, através da elasticidade, pode fornecer os ganhos de eficiência e redução de custos.

Para identificar um projeto candidato dentre os 83 encontrados no levantamento bibliográfico (vide [Tabela 11](#)), recorreu-se a uma característica presente nas publicações oriundas da filogenética: o uso difundido da citação das ferramentas utilizadas como referências nas publicações da área. Embora esta não seja uma característica exclusiva das publicações da filogenética, observações empíricas pelo autor sugerem uma alta aderência a este padrão de comportamento. Como consequência desta conduta torna-se possível identificar as ferramentas mais utilizadas e de maior impacto no dia a dia do pesquisador de bioinformática, viabilizando colocar em prática os objetivos previamente mencionados.

Desta forma, foi realizado o levantamento do número de citações para cada um dos 83 trabalhos encontrados através dos agregadores utilizados no levantamento bibliográfico ([Tabela 8](#)) como Google Scholar, CiteSeerX e SemanticScholar. O resultado deste levantamento é apresentado na sua totalidade na [Tabela 42 \(Apêndice A\)](#), onde seis projetos se destacam dos demais em função de sua popularidade: *a) PHYLIP – Felsenstein (1989); b) MEGA – Kumar, Stecher e Tamura (2016); c) MUSCLE – Edgar (2004); d) modeltest – Posada e Crandall (1998); e) mrBayes – Hulsenbeck e Ronquist (2001); e f) paup* – Swofford (2002)*. Dada a grande quantidade de trabalhos oriundos do levantamento bibliográfico, foi necessário recorrer a uma variação do princípio de Pareto¹ visando reduzir a lista para contemplar somente os trabalhos

¹ O princípio de Pareto, ([PARETO, 1971](#)), é uma famosa afirmação de que 80% dos efeitos vêm de 20% das

Figura 10 – Gráfico apresentando os projetos mais relevantes para esta pesquisa segundo o critério de que somados componham até 85% do total de citações dentre os trabalhos encontrados na revisão de literatura.



Fonte: Elaborado pelo autor.

mais citados e que em conjunto acumulam 85% das citações totais, descartando a cauda longa, ou seja, trabalhos de baixa relevância bibliográfica.

A aplicação desta estratégia resultou na seleção dos vinte primeiros trabalhos ordenados de acordo com o número de citações, e pode ser vista na Figura 10. Dentre esta lista de artigos chama a atenção a presença de três variantes de softwares utilizados para o cálculo de *best-fit* de sistemas de substituição filogenética, o MODELTEST de Posada e Crandall (1998), o jModelTest de Posada (2008) e o jModelTest2 de Darriba et al. (2012), que somados ultrapassariam o projeto PHYLIP, como mais citados neste levantamento. Quando considerada a importância da correta seleção do sistema de substituição genética, conforme abordado na Subseção 2.1.3, a quantidade de citações se mostra plausível e pode, até mesmo, comprovar quão difundida é a prática entre bioinformáticos.

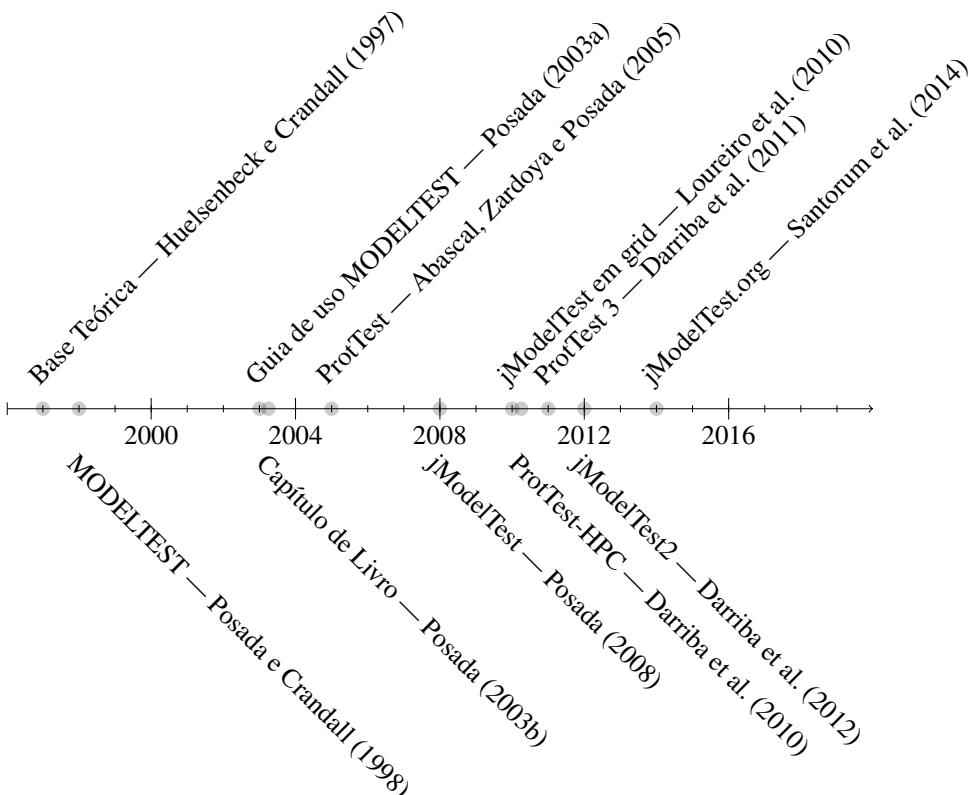
Como resultado desta análise o projeto jModelTest2 de Darriba et al. (2012) se apresenta como alvo e ponto de partida dos esforços de atualização e melhoria tecnológica no contexto da bioinformática e da inferência filogenética através do uso do modelo He–lastic. Com o objetivo de destacar para o leitor as publicações que contribuem para o entendimento das motivações e histórico dos desenvolvimentos que compõem a ferramenta, são apresentadas a seguir as origens do jModelTest2 e a relação entre autores e projetos relacionados. Desde o estabelecimento das fundações teóricas em 1997² até um portal web em 2014³, é apresentada na Figura 11 uma linha do tempo com os pilares que trouxeram o projeto até os dias de hoje, seja no quesito

causas. Este princípio é muito observado em empresas como uma estratégia para priorizar os esforços de maneira a obter os maiores retornos.

² Huelsenbeck e Crandall (1997)

³ Santorum et al. (2014)

Figura 11 – Linha do tempo do projeto jModelTest: histórico de publicações relevantes para compreensão do contexto, objetivos e evolução da ferramenta.



Fonte: Elaborado pelo autor.

autores⁴ (de Crandall para Posada e, por fim, Darriba) ou no que diz respeito a softwares⁵ (começando no MODELTEST, evoluindo para jModelTest e unindo forças com ProtTest). Daqui em diante o projeto como um todo poderá ser denominado apenas jModelTest, embora se refira especificamente à versão 2 do programa.

Programado na linguagem Java, o jModelTest2 contém uma interface gráfica acoplada ao programa e é facilmente executado em múltiplas plataformas através do *runtime* Java. O código fonte está disponível online, juntamente com o histórico de versionamento no portal [Github](#)⁶, facilitando a exploração do projeto e permitindo acompanhar sua evolução ao longo do tempo. O projeto está acompanhado de múltiplos artigos que abordam seu funcionamento e embasamento teórico, vide Figura 11, porém seu uso é simples e direto, apesar de exigir conhecimentos avançados da área de atuação para a compreensão dos resultados, ou seja, o grau de adequação dos sistemas de substituição de sequências moleculares e seus parâmetros.

⁴ Darriba parece ser o atual responsável pelo projeto, tendo herdado de Posada que o iniciou em colaboração com Crandall.

⁵ MODELTEST[†] foi o precursor direto do jModelTest que, no princípio, trabalhava apenas com um conjunto restrito de sistemas de substituição nucleotídica, no entanto ao longo da história suas funcionalidades parecem ter se fundido com o ProtTest[†] herdando a capacidade de avaliar sistemas de substituição proteica e aumentando a quantidade de sistemas disponíveis para teste de adequação.

[†]Aparentemente descontinuados e/ou obsoletos.

⁶ Disponível na URL <https://github.com/ddarriba/jmodeltest2>.

Figura 12 – Pseudocódigo retratando o cerne do fluxo de trabalho executado pelo jModelTest durante o teste de adequação de sistemas de substituição moleculares.

```

1: function MODELTEST(msa, sistemas)
2:   score_table ← DICT()
3:   etapa_score ← 0
4:   etapas ← CLUSTERINGSEARCH(sistemas)      ▷ divide os sistemas em
   etapas
5:   for e ← 1 to LEN(etapas) do
6:     fit_score ← 0
7:     subsistemas ← etapas[e]
8:     for s ← 1 to LEN(subsistemas) do
9:       sistema ← subsistemas[s]
10:      score_table[sistema] ← CALCULATEFIT(sistema, msa)      ▷
    executa o cálculo pelo PhyML
11:      if score_table[sistema] > fit_score then      ▷ guarda o melhor
12:        fit_score ← score_table[sistema]
13:      end if
14:    end for
15:    if fit_score < etapa_score then      ▷ se não há progresso, encerra
16:      break
17:    else
18:      etapa_score ← fit_score
19:    end if
20:  end for
21:  return score_table      ▷ retorna a lista completa de sistemas com seus
   respectivos scores
22: end function

```

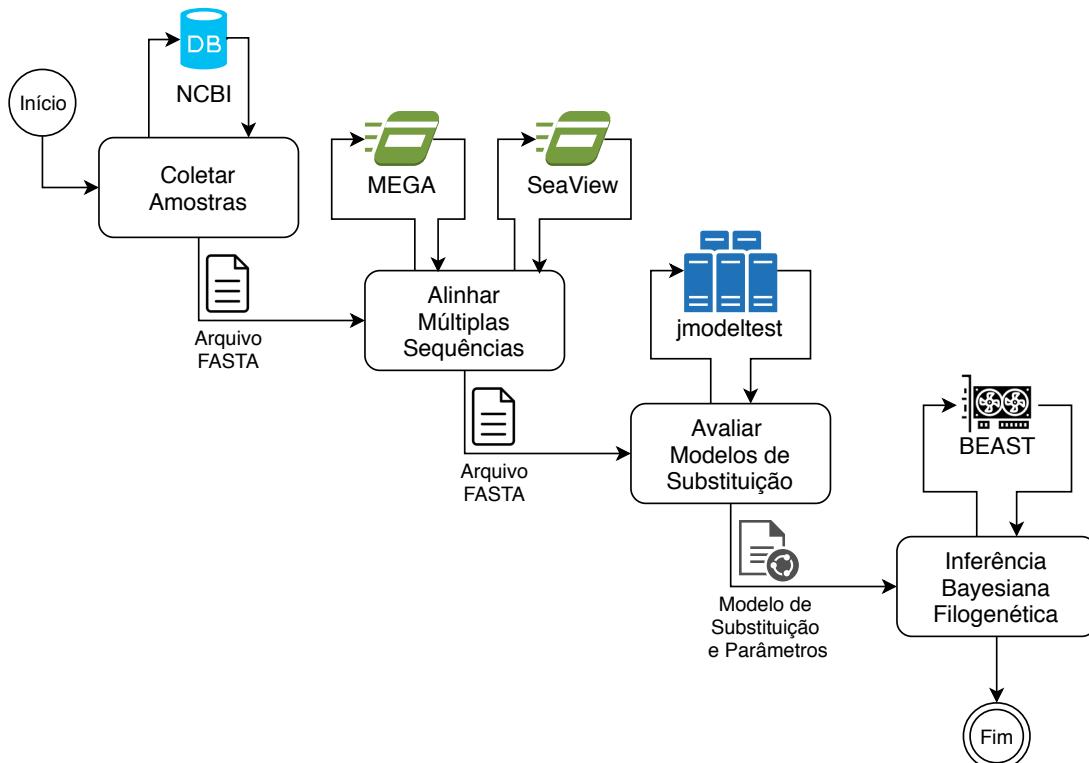
Fonte: Elaborado pelo autor.

Conforme previamente abordado na [Seção 3.3](#) e na [Tabela 11](#), o jModelTest tem suporte explícito à múltiplas CPUs, seja localmente, ou até mesmo, em ambientes MPI. Outra característica importante do programa é que a natureza do trabalho realizado permite um alto grau de paralelismo ([KEANE, 2006](#)), uma vez que o teste de cada um dos sistemas de substituição de sequências pode ser realizado de maneira independente dos demais. A [Figura 12](#) apresenta, em pseudocódigo, o bloco responsável pelo cálculo de adequação realizado pelo jModelTest no modo *Clustering Search*, detalhado na [Seção 4.4](#). Na sua estratégia de paralelismo, o programa tira proveito do fato de que o *loop* contido entre as linhas 8 e 14 não exige dados de estado ou externos ao bloco de código, o que torna trivial habilitar o paralelismo neste trecho. Estas e outras características técnicas serão abordadas em maior profundidade nas próximas seções.

Embora haja menção ao uso de computação em nuvem, através de testes em instâncias EC2⁷ e documentado em [Darriba et al. \(2012\)](#), o cenário explorado segue um modelo de recursos fixos e predefinidos, efetivamente simulando um *cluster*, e, portanto, não se beneficiando da elasticidade de recursos disponível nestes ambientes. Ainda assim, o programa consegue obter

⁷ Serviço que fornece máquinas virtuais disponibilizadas pelo provedor de computação em nuvem AWS.

Figura 13 – Exemplo minimalista do fluxo de tarefas típico para realização de análises filogenéticas no laboratório de filogenética da UNISINOS: Coleta de amostras através do banco de dados NCBI, Alinhamento de Sequências usando os projetos MEGA ou SeaView, Seleção de sistemas de substituição com a ajuda do jModelTest e, finalmente, Inferência filogenética por métodos Bayesianos com o projeto BEAST.



Fonte: Elaborado pelo autor com base em comunicação pessoal com membros da equipe de estudos filogenéticos da UNISINOS.

desempenho razoável ao fazer uso de todos os processadores disponíveis durante sua execução e se mostrar apto a trabalhar no ambiente de computação em nuvem sem a necessidade de significativas modificações em seu código ou procedimentos.

Além da popularidade do projeto, evidenciada pelo número de citações, outro fator relevante para sua escolha está no fato de que, no processo de análise e inferência filogenética, a seleção de um sistema de substituição molecular é um dos passos preparatórios à análise propriamente dita, assim como a coleta de amostras e o alinhamento de sequências moleculares. Contudo, apesar do seu caráter preparatório, este costuma ser um passo demorado em função do custo computacional da avaliação de *best-fit*, dentre os inúmeros sistemas de substituição, assim como, de suas variantes, e também o conjunto de dados a ser analisado.

De acordo com levantamento realizado junto ao laboratório de filogenética da UNISINOS, em um típico fluxo de tarefas para análise filogenética (como aquele apresentado na Figura 13), a seleção de sistemas pode levar diversas horas e, conforme o tamanho e a quantidade dos alinhamentos de sequências, nem mesmo chega a terminar, esbarrando em limitações técnicas. Esta lentidão no processo de busca do sistema mais adequado aos dados pode fazer com que pesquisadores negligenciem a importância de escolher o sistema correto, o que, segundo advertem

Minin et al. (2003) e Keane (2006), pode levar a conclusões infundadas e puramente erradas.

Sendo assim, uma possível redução no tempo de execução destes cálculos traz duplos benefícios ao: *a)* incentivar aqueles que atualmente não efetuam uma rigorosa seleção de sistemas a fazê-lo; além de *b)* conceder mais tempo livre para análises mais profundas aos pesquisadores que já utilizam o processo de seleção. Podendo, em última instância, aumentar a qualidade da produção acadêmica em ambos os casos.

4.2 Decisões de Projeto

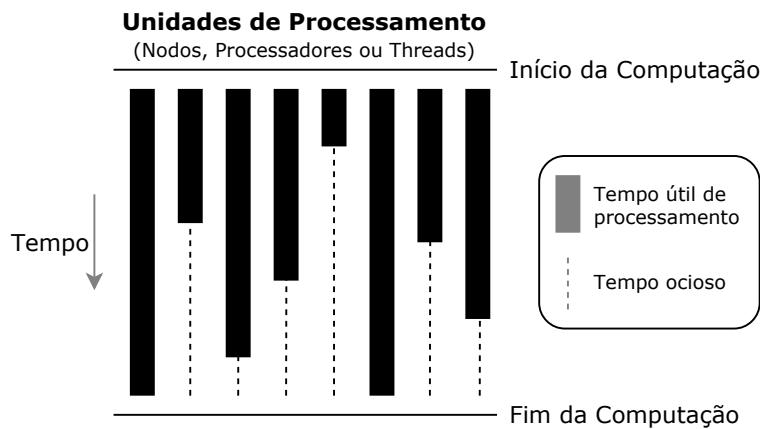
Originalmente detalhado em Posada (2008), com melhorias no campo da computação de alto desempenho, delineadas posteriormente em Darriba et al. (2014), a principal função do projeto jModelTest é encontrar o sistema de substituição molecular que melhor se encaixa ao alinhamento de sequências informado como entrada, em um processo chamado de *best-fit*. Esta tarefa é altamente receptiva ao paralelismo, uma vez que, cada sistema pode ser testado individualmente e os resultados coletados e comparados ao final do processo, elencando o sistema mais adequado, segundo algum critério de priorização ou ranqueamento (KEANE, 2006).

Contudo, apesar da aparente simplicidade da tarefa, um fator relevante no esforço computacional é a complexidade do sistema de substituição molecular. Conforme abordado na Subseção 2.1.3, nem todos os sistemas são criados iguais, com a complexidade aumentando progressivamente ao longo dos anos e acompanhando a profundidade do conhecimento acerca dos processos evolutivos. Desta forma seria intuitivo pensar que deve sempre utilizar-se o sistema mais complexo possível, uma vez que este será capaz de melhor se adaptar aos dados encontrados. No entanto, este não é o caso, pois como afirma Zwickl (2006), além de exigir maior esforço computacional, um sistema muito complexo está mais vulnerável ao risco de atribuir grande importância a algum tipo de ruído aleatório presente nos dados.

Para combater estes riscos é comum que critérios de seleção de sistemas, como os critérios de informação de Akaike (AKAIKE, 1974) e Bayesianos (SCHWARZ, 1978), penalizem a complexidade dos sistemas durante a escolha do *best-fit* (YANG, 2014). Esta política implica na avaliação de todos os sistemas disponíveis, dos mais simples aos mais complexos, para obter um panorama da adequação aos dados, o que, computacionalmente, gera tarefas com grande variação na complexidade dos cálculos e consequente carga computacional. Desta forma, ainda que paralelizável, o cálculo de adequação do sistema é uma tarefa com ampla variação no esforço necessário, o que, conforme pode ser observado na Figura 14, desafia estratégias ingênuas de computação paralela e distribuída em função do desbalanceamento da carga computacional entre os nodos disponíveis.

Dependendo das configurações selecionadas pelo usuário para início da análise o número de sistemas simulados pode variar amplamente, com possibilidades tão baixas quanto apenas três sistemas, ou tão altas quanto 1624 variações dos sistemas de evolução para teste. Quando executado em um computador de mesa, ou até mesmo, um servidor institucional, é possível

Figura 14 – Representação de carga em múltiplos processadores ou nodos durante a execução de processos com complexidade variável, exemplificando o cenário encontrado pela aplicação jModelTest durante o teste de adequação de sistemas de substituição de sequências moleculares onde parte significativa dos recursos pode permanecer ociosa enquanto aguarda a conclusão dos cálculos que exigem maior poder computacional.



Fonte: Elaborado pelo autor.

que o hardware fique subutilizado, embora seja pouco provável, uma vez que as probabilidades estejam a favor de que este se torne um gargalo, haja vista que as configurações padrão geram 88 cenários de teste, enquanto um computador de mesa conta, atualmente, com um número entre 4 a 16 *cores* de processamento. Ainda, no que diz respeito aos fatores que influenciam a carga computacional durante o cálculo, quatro são especialmente importantes no contexto deste trabalho:

- 1) Escolha dos sistemas elencados para o teste de adequação;
- 2) Complexidade inerente a cada sistema⁸;
- 3) Quantidade de sequências moleculares; e
- 4) Comprimento das sequências moleculares⁹;

A partir destes quatro elementos é possível estabelecer uma série de equações que determinam o comportamento computacional de uma execução do processo de adequação de sistemas de evolução filogenética através do software jModelTest. O cálculo do custo de uma execução do jModelTest é obtido através da Equação 4.3 (detalhada na Tabela 12) que, de acordo com os fatores previamente citados, combina o custo oriundo dos dados do arquivo de alinhamentos com o custo obtido através das escolhas de parâmetros para a execução, ambos detalhados a seguir. A Equação 4.1 detalha o custo computacional oriundo do arquivo, $C_a(i, c)$, usado como parâmetro de entrada para o programa jModelTest. Este arquivo deve conter um alinhamento de sequências moleculares compostas por amostras de uma determinada região de comprimento c de uma lista de indivíduos i .

⁸ Geralmente medida em termos da quantidade de parâmetros livres sujeitos à otimização.

⁹ Geralmente medido em termos do número de caracteres que compõem a sequência.

Tabela 12 – Detalhamento das variáveis e funções utilizadas na definição de custo de execução do jModelTest.

Variável / Função	Descrição
i	Quantidade de indivíduos / amostras contidos no arquivo de alinhamento de sequências moleculares
c	Comprimento da região molecular de interesse (geralmente uma região do código genético ou um gene específico)
s	Lista dos sistemas de modelagem do processo de evolução / substituição molecular (por exemplo os contidos na Tabela 4)
$C_a(i, c)$	Custo oriundo do arquivo de alinhamento de sequências moleculares
$C_p(s)$	Custo oriundo dos parâmetros escolhidos pelo usuário durante a execução do programa
C_s	Custo de um sistema de substituição molecular
$C(i, c, s)$	Custo total de uma execução do jModelTest

Fonte: Elaborado pelo autor.

A seguir, na [Equação 4.2](#), é demonstrado o cálculo do custo computacional em função dos parâmetros de execução, $C_p(s)$, sendo o principal elemento a variável s que representa uma lista de sistemas de substituição de sequências moleculares, onde o resultado desta equação é a soma dos custos de cada um dos sistemas escolhidos. Esta lista contém cada um dos sistemas de substituição a ser testados pelo software em busca daquele que melhor representa os dados, no processo de *best-fit*, e é obtida através da combinação dos parâmetros informados pelo usuário durante o uso do software. Embora o custo computacional do cálculo de cada sistema não seja abordado neste trabalho, ele é explorado em profundidade nas teses de [Keane \(2006\)](#), [Zwickl \(2006\)](#) e [Darriba \(2015\)](#).

$$C_a(i, c) = i \times c \quad (4.1)$$

$$C_p(s) = \sum_{k=1}^n C_s(s_k) \quad (4.2)$$

$$C(i, c, s) = C_a(i, c) \times C_p(s) \quad (4.3)$$

A proposta descrita neste capítulo adotou decisões diretamente influenciadas pelo contexto apresentado na [Seção 4.2](#), assim como, os requisitos definidos pelos objetivos deste trabalho (detalhados na [Seção 1.3](#)) e as lacunas de pesquisa encontradas durante a análise comparativa ([Seção 3.4](#)) dos trabalhos relacionados focando, todavia, na questão da elasticidade no consumo de recursos computacionais. Tais decisões formam as premissas básicas que fundamentam a arquitetura do modelo proposto, atuar como guias sobre qualquer desenvolvimento realizado. Com este cenário em mente foram estabelecidas as seguintes decisões de projeto:

Decisão 1: Deve tirar proveito das funcionalidades disponíveis em um ambiente de computação em nuvem;

Decisão 2: Deve suportar a elasticidade de recursos computacionais a nível de seus algoritmos¹⁰;

Decisão 3: Deve lidar com a variabilidade na carga computacional oriunda das características do teste de adequação de sistemas de substituição molecular;

Decisão 4: Deve permanecer agnóstico a provedores de computação em nuvem, evitando depender de funcionalidades exclusivas da plataforma;

Decisão 5: Deve manter a configuração tão simples quanto possível, idealmente limitando-se às regras de elasticidade;

Decisão 6: Deve empenhar-se em reduzir a carga operacional, delegando tarefas de gestão para o provedor de computação em nuvem; e

Decisão 7: Deve buscar uma melhor relação custo-benefício em comparação às soluções existentes.

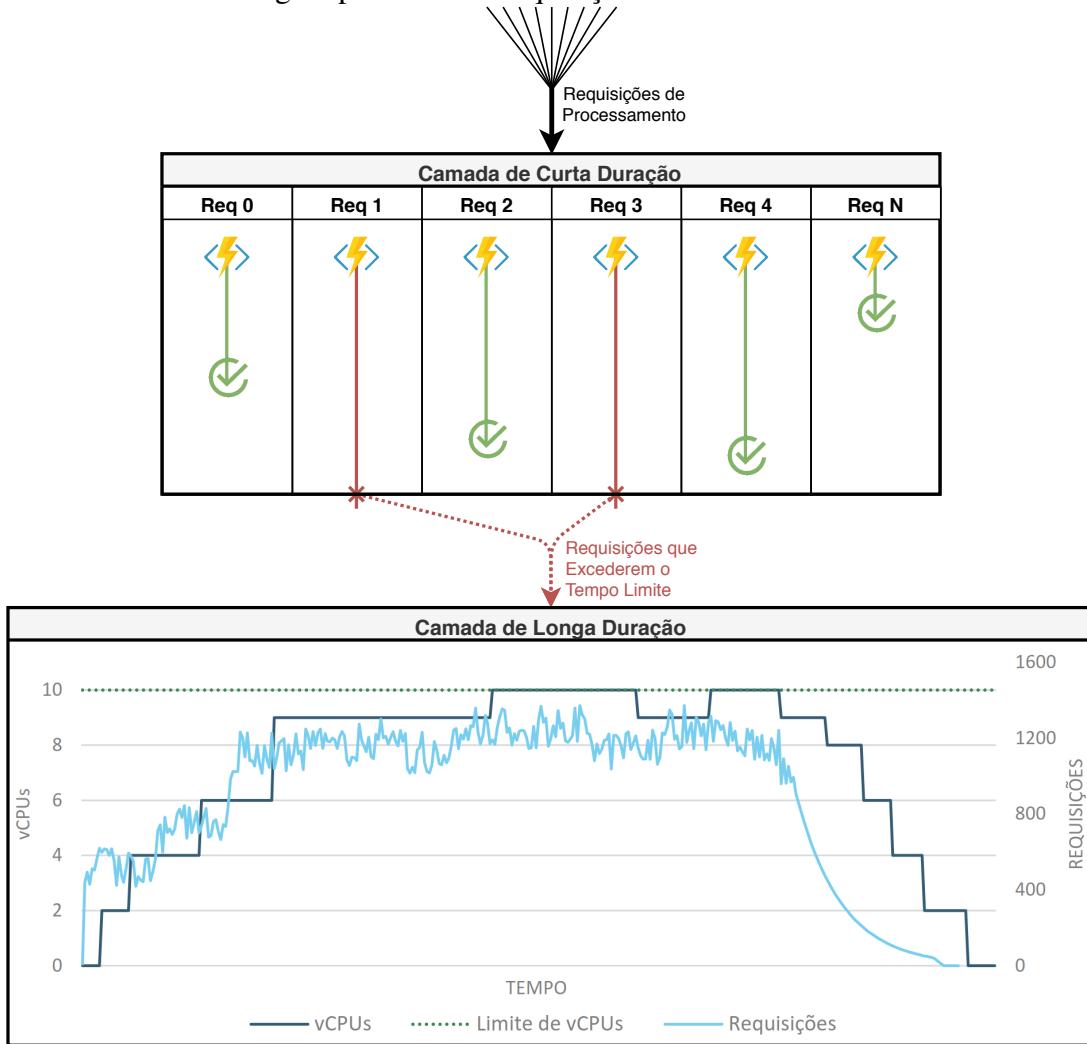
4.3 Arquitetura

O modelo elaborado, batizado de He–lastic, busca, através da elasticidade de recursos computacionais, balancear performance computacional, medida pelo tempo de execução, com custo financeiro através do uso consciente de recursos, ao lançar mão da funcionalidade mais adequada para cada tipo de processo em execução. Tal estratégia é factível em função da divisão em camadas de elasticidade proposta pelo modelo (ilustrada na Figura 15), onde cada uma apresenta características complementares e melhor adaptadas para classes distintas de processos. Portanto, é na divisão em camadas de elasticidade, abordada em detalhes na Seção 4.4, que encontra-se o ponto fundamental de diferenciação do modelo He–lastic.

Do ponto de vista do usuário da aplicação jModelTest as mudanças são mínimas, uma vez que sua execução se dá por interface gráfica ou linha de comando e o efetivo processamento já ocorre em plano de fundo, assim como é previsto com a adoção do modelo proposto. Desta forma, cabe salientar que o jModelTest oferece uma interface gráfica para uso interativo além do uso por linha de comando, embora este último seja recomendado para usuários experientes, dado que é mais confiável e apropriado para integração com demais etapas do *pipeline* de análise filogenética. No que diz respeito ao processamento, referente ao cálculo de *best-fit* de sistemas de substituição de sequências moleculares, que executa em *background*, o jModelTest oferece opções via memória compartilhada, com o uso de múltiplas *threads* de processamento, e através

¹⁰ Descartando, desta forma, estratégias de elasticidade baseadas na quantidade de usuários paralelos do sistema, como no modelo clássico de elasticidade para aplicações web e *e-commerce* com um平衡ador de carga distribuindo requisições para um conjunto de réplicas de máquinas virtuais.

Figura 15 – Visão conceitual do modelo He-lastic ilustrando como suas duas camadas de elasticidade interagem para atender requisições.



Fonte: Elaborado pelo autor.

da biblioteca paralela MPI, recomendada para ambientes de *cluster*. Uma vez que os cálculos de *best-fit*, que compõem a parte mais relevante do custo computacional, já são executados em *background*, não é esperado impacto no uso da ferramenta por parte do usuário final em função da adoção do modelo He-lastic, podendo ser visto como a implementação de um *back-end* SaaS para o jModelTest, uma vez que isola do usuário todo o trabalho de configuração e pode ser acessado como uma API.

Ainda assim, existe a necessidade de configurar alguns parâmetros de execução referentes ao ambiente de computação em nuvem e fundamentais para o bom funcionamento do modelo, conforme detalhados na Tabela 13. Neste caso, a figura de um administrador de ambientes deve ser prevista. Embora este perfil de usuário não seja mencionado nas publicações e documentações do projeto jModelTest, é natural projetar a sua existência para qualquer cenário de uso exceto os mais simples. Tal afirmação deve-se à necessidade de separar o usuário com conhecimentos em bioinformática (exerce domínio sobre conhecimentos da biologia e suas ferramentas, contudo é

Tabela 13 – Parâmetros necessários para a configuração do modelo He–lastic.

Camada	Parâmetro	Descrição
Curta Duração FaaS	Tempo Limite	Tempo máximo de processamento para os processos executando nesta camada.
	Potência	Determina o poder de processamento alocado para cada unidade de execução na camada. Geralmente se manifesta como uma medida de memória alocada, embora possa ser composta por outras métricas (vide Tabela 5).
Longa Duração Orquestrador de Contêineres	Número de CPUs	Quantidade limite de CPUs disponíveis para uso. Tem por objetivo garantir a previsibilidade de custos para esta camada.
	Limite Inferior de Carga	<i>Threshold</i> indicativo da ocupação mínima permitida para cada VM da camada. A violação deste limite causará uma ação de elasticidade para consolidação / redução de recursos.
	Limite Superior de Carga	<i>Threshold</i> indicativo da ocupação máxima permitida para cada VM da camada. A violação deste limite causará uma ação de elasticidade para replicação / incremento na quantidade de recursos.

Fonte: Elaborado pelo autor.

incapaz de executar tarefas computacionais mais avançadas) do administrador, um perfil com profundo conhecimento computacional, capaz de provisionar servidores, configurar ambientes e montar *clusters* computacionais. Cabe, também, ao administrador determinar os valores para os parâmetros de operação do modelo, uma tarefa que pode ser realizada de maneira empírica através de experiência ou observações, ou seguindo uma abordagem mais científica através de uma análise de cenários de custo–benefício, como será detalhado na Seção 6.1.

As tarefas de responsabilidade do administrador tornam-se necessárias a partir do momento em que os recursos locais do usuário bioinformata deixam de ser suficientes para suas análises, sendo este o mesmo momento em que ele potencialmente buscará por alternativas como o modelo He–lastic aqui proposto, o que leva à conclusão de que, mesmo com alguma parametrização necessária no ambiente de nuvem, este não é um fator de diferenciação entre o He–lastic e o jModelTest, principalmente em cenários em que há o uso de *clusters* MPI. Ademais, o reduzido número de parâmetros exigidos pelo He–lastic limita a profundidade do conhecimento necessário, sendo este um benefício oriundo da utilização da computação em nuvem.

Embora as camadas de elasticidade componham o cerne do modelo He–lastic, os demais componentes devem ser abordados para uma visão holística da proposta antes que se possa focar em seus detalhes, conforme abordado na Seção 4.4, onde são exploradas as características das camadas de elasticidade e seus parâmetros, assim como sua influencia sobre seu funcionamento.

A seguir, é fornecida uma descrição a respeito de todos os elementos presentes na arquitetura do modelo He-lastic, elucidando suas funções, interações e principais características, sendo esta dividida em dois grandes contextos, o do ambiente local e o de computação em nuvem. Tal divisão se origina a partir do cenário de uso do *software* jModelTest, baseado em interface gráfica, onde um pesquisador de bioinformática deseja executar seu teste de adequação de sistemas de substituição molecular a partir de sua estação de trabalho, enquanto tira proveito do poder computacional fornecido através da computação em nuvem. Visando atender este cenário a arquitetura do modelo He-lastic requer os seguintes componentes:

- **Ambiente local:** neste contexto se concentram os componentes que interagem direta ou indiretamente com o usuário final da aplicação, sendo o programa jModelTest o principal deles. Para atingir o objetivo de incrementar as funcionalidades do mesmo, enquanto possibilita o aproveitamento de recursos da computação em nuvem, é necessária a introdução de um componente denominado ‘Módulo Mediador’, que intermediará as ações computacionais assim como a comunicação com o programa jModelTest, passando a impressão de que, do ponto de vista do usuário, nada mudou. Este componente mediador é, por sua vez, composto por três subsistemas:
 - **Produtor:** traduz e transmite as solicitações de processamento geradas pela aplicação jModelTest para o ambiente de computação em nuvem, desencadeando o processamento destas;
 - **Consumidor:** coleta os resultados das execuções que ocorreram no ambiente de computação em nuvem, reportando-os de volta em um formato compreendido pelo jModelTest;
 - **Guardião¹¹:** monitora e gerencia possíveis situações excepcionais observadas durante o processamento das requisições, tratando erros, se possível, e reportando tais situações ao programa jModelTest, enquanto aborta os demais processamentos em andamento, visando evitar desperdício de recursos.
- **Ambiente de Computação em Nuvem:** agrupados neste contexto estão os componentes responsáveis pela execução dos cálculos de adequação dos sistemas de substituição molecular, podendo ser denominado como o *back-end* do modelo He-lastic por conter apenas componentes invisíveis do ponto de vista do usuário. Através do desacoplamento entre componentes é possível se manter fiel às decisões de projeto definidas na Seção 4.2, dando origem a uma arquitetura escalável e elástica enquanto suporta uma variada gama de provedores de computação em nuvem com reduzido fardo operacional para os administradores. São de fundamental importância neste ambiente os seguintes elementos:

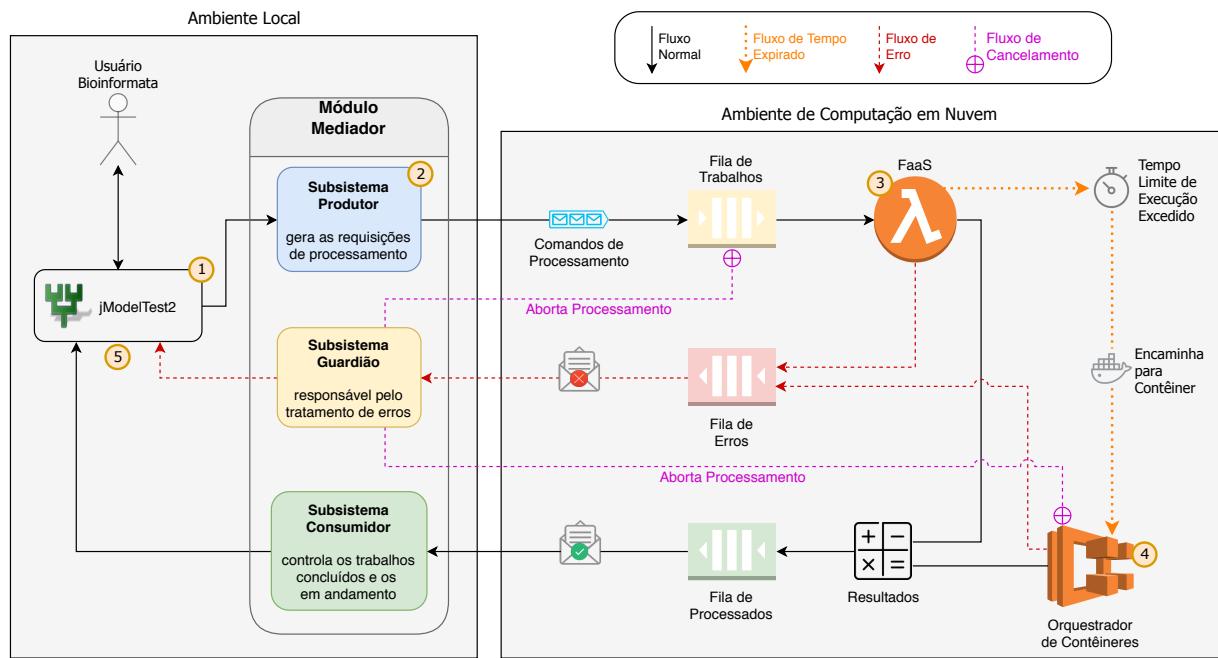
¹¹ Durante testes locais com o jModelTest foram encontradas situações de exceção que levaram ao surgimento de processos órfãos, que continuavam consumindo recursos mesmo após o cancelamento da execução. Em um contexto de computação em nuvem uma situação como essa é particularmente perturbadora, pois pode acarretar em custos financeiros indesejados, justificando a necessidade de um subsistema dedicado.

- **Filas de Mensagens:** por meio do uso de filas buscamos fornecer ao sistema os meios para se adaptar às variações de demanda e mantê-lo resiliente a falhas, uma vez que, processos com erros podem voltar para a fila ou ser direcionados para outro destino. Em se tratando de arquiteturas que visam elasticidade, tanto provedores de computação em nuvem, quanto acadêmicos como [Tran, Skhiri e Zimányi \(2011\)](#), recomendam o uso de filas e comunicação assíncrona orientada a eventos como padrão de projetos. Seu uso é difundido entre iniciativas *cloud-native* e suportado pelos principais provedores de computação em nuvem através de serviços gerenciados além de grande quantidade de alternativas *open-source*;
- **FaaS:** utilizando recursos de FaaS podemos reduzir a carga administrativa, uma vez que fornecemos apenas o código para execução e podemos tirar proveito de uma frota virtualmente infinita de processadores. Instâncias de funções são provisionadas sob demanda em segundos, escaladas elasticamente conforme a necessidade, com seu custo calculado por invocação e pelo produto do tempo de execução, de acordo com a quantidade de recursos utilizados, chegando a uma implementação quase perfeita do conceito *pay-as-you-go* ([SPILLNER; MATEOS; MONGE, 2017](#)), evitando ainda, custos com recursos inativos, pois os servidores permanecem sob gestão do provedor de computação em nuvem que é capaz de multiplexar seu uso entre clientes. Contudo, uma importante restrição das arquiteturas baseadas em FaaS é o limite no tempo de execução dos códigos e recursos computacionais relativamente restritos, o que torna esta modalidade inadequada para uma variedade de cenários;
- **Orquestrador de Contêineres:** surge como um antídoto ao limite de tempo de execução dos FaaS ao fornecer, através da conteinerização, um ambiente similar ao de um FaaS, capaz de executar os mesmos códigos. Contudo este não contam com a limitação relativa ao tempo de execução em troca de um maior fardo operacional, servindo, no modelo proposto, como um amortecedor, capaz de lidar com cálculos de longa duração e que seriam rejeitados pela camada de FaaS. Outro benefício é a redução na ocorrência de *thrashing* de recursos computacionais, uma vez que temos a garantia (pela forma como a arquitetura do modelo está estruturada) de que todas as requisições encaminhadas para o orquestrador são de média a longa duração.

Encontram-se representados na [Figura 16](#) os principais elementos do modelo, assim como, o relacionamento entre eles, com destaque para os dois grandes blocos que dividem o modelo em ambiente local e de computação em nuvem, sendo o primeiro responsável pela interação com o usuário, enquanto o segundo, tem como principal tarefa a execução dos cálculos para adequação do sistema de substituição molecular. Estão destacados na figura cinco momentos distintos durante a execução do modelo He-lastic em conjunto com o jModelTest, sendo eles:

- 1) o usuário interage com o *software* jModelTest, através de seu ambiente local, gerando requisições de processamento que serão interceptadas pelo Módulo Mediador do He-lastic;

Figura 16 – Diagrama conceitual de alto nível da arquitetura do modelo He-lastic elucidando a relação entre seus componentes assim como, a divisão em ambiente local (responsável pela interação com o usuário) e de computação em nuvem (responsável pelos cálculos), com destaque para os elementos FaaS e Orquestrador de Contêineres, que representam as unidades de elasticidade do modelo, assim como as Filas de Mensagens que coordenam a comunicação entre os componentes do modelo



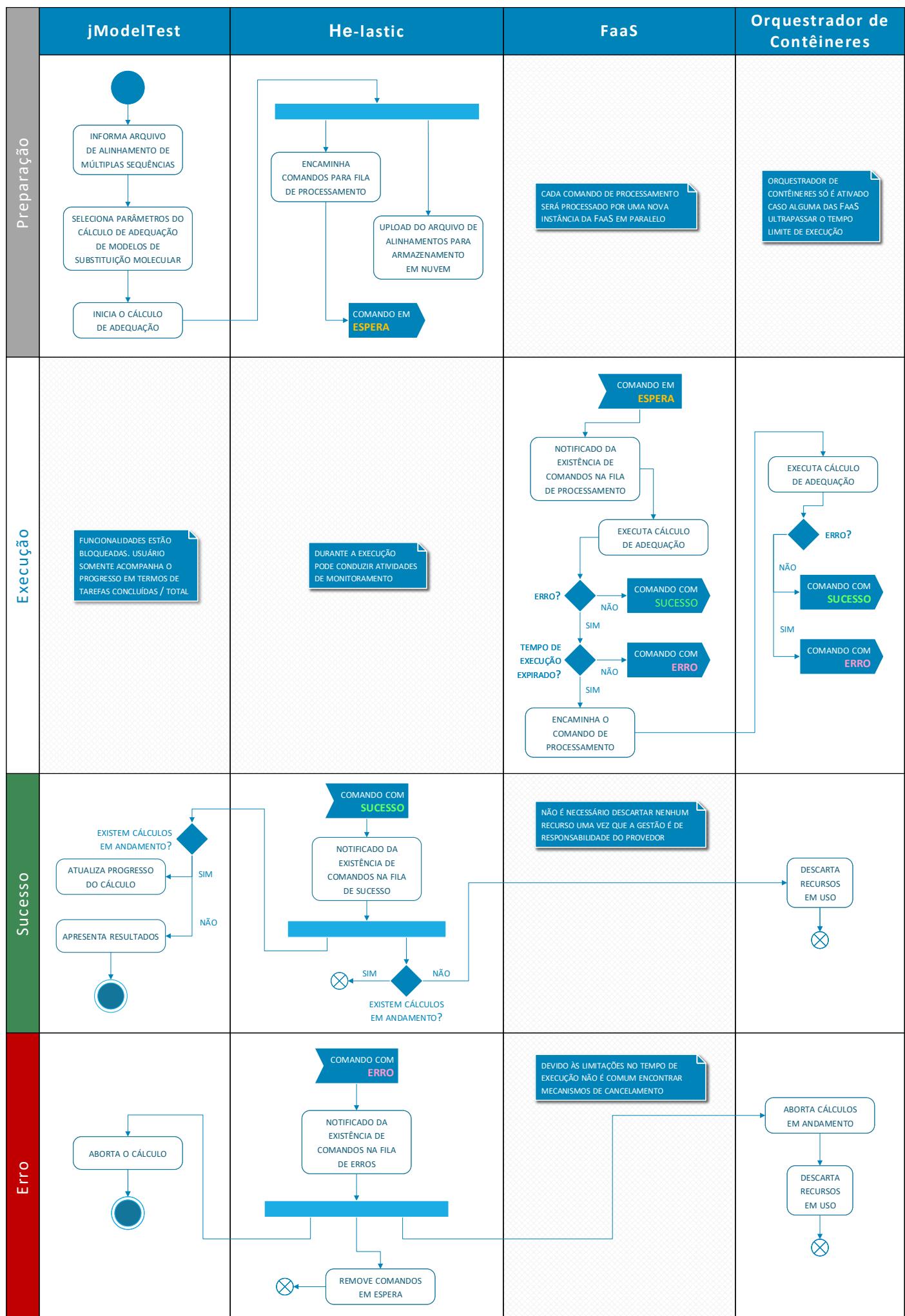
Fonte: Elaborado pelo autor.

- 2) de posse das requisições, o Subsistema Produtor traduz os comandos recebidos pelo jModelTest em requisições de processamento adequadas para o ambiente de computação em nuvem do modelo He-lastic e deposita estas mensagens em uma fila que alimenta a primeira camada de elasticidade;
- 3) cada mensagem recebida pela Fila de Trabalhos causa a ativação de uma função FaaS correspondente tirando proveito das características dessa camada, principalmente, a capacidade de absorver picos repentinos de processamento, lançando mão do paralelismo massivo habilitado por uma frota de recursos ociosos gerenciados pelo provedor de computação em nuvem;
- 4) contudo, é possível que determinadas requisições não possam ser atendidas na camada FaaS devido ao tempo limite de execução, cenário onde tais requisições serão encaminhadas para a segunda camada de elasticidade (vide Figura 15), o Orquestrador de Contêineres que representa um ambiente mais propício para execuções de longa duração e que demandam grande poder de processamento contínuo;
- 5) ao fim do processamento, todos os resultados estarão disponíveis na Fila de Processados, que é monitorada pelo Subsistema Consumidor, responsável por encaminhar os resultados

de volta ao *software* jModelTest, com o Subsistema Guardião executando função análoga com relação aos possíveis erros ocorridos durante o processo.

Uma vez detalhados os objetivos e os componentes do modelo torna-se possível a utilização de técnicas de projeto e arquitetura de *software* para fornecer uma visão de alto nível e descrever as interações entre os elementos que compõem o modelo He–lastic. Por meio da Linguagem Unificada de Modelagem (UML, no inglês), podemos representar formalmente o fluxo de controle entre os componentes do modelo, ou seja, seu comportamento, em um diagrama de atividades. Este diagrama apresenta, conforme detalhado pela [Figura 17](#), no eixo vertical as etapas de execução do jModelTest, sendo compostas por Preparação, Execução, Erro e Sucesso enquanto no eixo horizontal são descritos os principais componentes do modelo. Desta forma é possível identificar, por exemplo, a transição da camada FaaS para a camada de Orquestração de Contêineres através da leitura diagonal da esquerda para a direita e de cima para baixo, seguindo também as setas e seus possíveis locais de encaixe, que são representados pelo símbolo de evento, conforme a especificação UML Os pontos de comunicação assíncrona através de filas de mensagens realizam a mediação entre ambiente local e de computação em nuvem além de favorecer o comportamento elástico através do desacoplamento, permitindo que cada componente do modelo atinja escalabilidade independentemente dos demais.

Figura 17 – Diagrama UML de atividades do modelo He-lastic representando a interação, agrupada por etapa do processamento, entre os componentes da arquitetura.



Fonte: Elaborado pelo autor.

4.4 Estratégias para Elasticidade

A interação entre os componentes previamente citados representa o cerne do modelo He–lastic que, através das suas duas unidades de elasticidade, nomeadamente o componente FaaS e o Orquestrador de Contêineres, operando na modalidade horizontal, com política automática reativa e estratégia por replicação (vide Subseção 2.2.3), possibilita um melhor acompanhamento da demanda computacional e economia ao evitar custos ociosos. Estas duas unidades apresentam características distintas uma da outra, mas que se mostram, de certa forma, complementares. Na Tabela 14, uma comparação entre estas características é apresentada e, a seguir, uma descrição detalhada das mesmas.

Tarefas de média e longa duração são gerenciadas pelo Orquestrador de Contêineres que segue um modelo clássico de elasticidade automática reativa por replicação, distribuindo seu processamento entre uma frota de máquinas virtuais, que tem sua comunicação mediada por um平衡ador de carga (embutido no orquestrador de contêineres). Apesar da inexistência de restrições técnicas para o processamento de tarefas de curta duração, o modelo de especificação baseado em horas de uso, assim como, os *delays* de início e término das máquinas virtuais (que contribui para o fenômeno conhecido como *thrashing* (BERSANI et al., 2014)), faz com que tarefas curtas mereçam tratamento diferenciado, incrementando a complexidade de desenvolvimento e de operação.

Por este motivo o FaaS é empregado como uma solução para tarefas de curta duração, relevantes para o contexto da aplicação jModelTest, ao apresentar um modelo de especificação baseado em segundos de uso e empregar uma frota de máquinas virtuais gerenciadas e otimizadas pelos provedores de computação em nuvem, conforme abordado pela Figura 6. Neste componente, os únicos requisitos são a definição da quantidade de recursos necessários e um pacote com o código fonte a ser executado por um dos ambientes oferecidos pelo provedor.

Do ponto de vista do programador, cada execução da sua função é análoga a iniciar uma instância, executar seu código e descartá-la. Contudo, a gestão fica por conta do provedor, que é responsável pelo isolamento do ambiente e reaproveitamento de máquinas virtuais em execução. Do ponto de vista da elasticidade, o uso de FaaS dá, ainda mais do que com a elasticidade clássica, a impressão de recursos ilimitados, uma vez que, para cada novo evento, um novo executor é alocado, possibilitando lidar facilmente com cargas de trabalho que ocorrem em rajadas (*bursty workloads*). Entretanto, este comportamento só é possível devido aos limites estabelecidos no tempo de execução de cada chamada, geralmente menores que 10 minutos, o que aumenta a rotatividade dos recursos e permite que o provedor de computação em nuvem absorva picos de demanda com capacidade ociosa através da multiplexação entre usuários.

Ao mencionar o modelo de elasticidade clássico, o presente trabalho faz referência a já amplamente estudada estratégia de elasticidade automática por replicação baseada em *thresholds* superiores e inferiores que controlam a adição e remoção de máquinas virtuais dentre um *pool* de recursos que fica acessível através de um balanceador de carga, como representado

Tabela 14 – Comparativo entre as principais características das unidades de elasticidade utilizadas no modelo He–lastic (maiores informações a respeito das implementações de FaaS podem ser encontradas na [Tabela 5](#))

	Orquestrador de Contêineres	FaaS
Elasticidade	Através de mecanismos Regra — Condição — Ação (automática reativa)	Virtualmente infinita, uma nova instância para cada requisição (orientada a eventos)
Requisitos	Imagen do Ambiente de Execução	Pacote com Código Fonte
Carga Operacional	Maior	Menor
Provisionamento	Em torno de Minutos	Em torno de Segundos
Tempo de Execução	Ilimitado	Limitado em poucos Minutos
Precificação	Por Hora	Por Segundo
Ociosidade	Incorre Custos	Não Incorre Custos
Adequação	Processos Longos	Processos Curtos

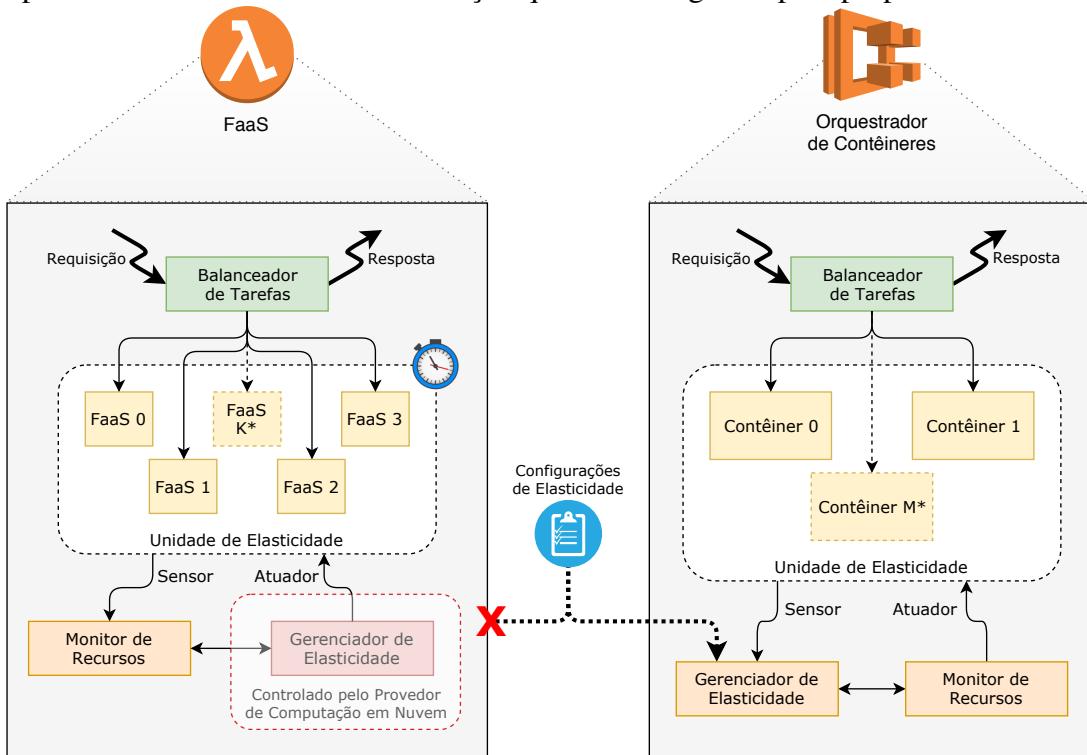
Fonte: Elaborado pelo autor.

pela [Figura 49 \(Apêndice A\)](#). Este modelo é muito popular entre aplicações comerciais como *e-commerce*, portais e *blogs*. Todavia, seu uso no mundo acadêmico se mostra limitado quando confrontado com modelos de paralelismo como Fases Paralelas e Pipelines, conforme abordado em [Aubin e Righi \(2015\)](#), [Righi et al. \(2016\)](#) e [Righi et al. \(2017\)](#). Ainda assim, apesar da semelhança estrutural, as camadas de elasticidade FaaS e Orquestrador de Contêineres, ilustradas na [Figura 18](#), apresentam características distintas e complementares, principalmente, em função das particularidades a respeito da virtualização e tempo limite de execução, podendo ser interpretadas até mesmo como especializações do modelo clássico de elasticidade.

Tal estratégia contrasta com a proposta original de paralelismo do *software* jModelTest ao introduzir não somente elasticidade, como aplicando também uma divisão em camadas para melhor se adequar às variações na demanda computacional. Conforme previamente abordado na [Seção 4.3](#), o jModelTest propõem acelerar a computação através do uso de *threads* ou por intermédio da biblioteca paralela MPI, sendo essa estratégia mais utilizada em casos onde existe grande demanda computacional. A [Figura 19](#) apresenta uma visão conceitual sobre a estratégia de paralelismo adotada pelo jModelTest quando executando com o auxílio da biblioteca MPI, onde quatro pontos são especialmente relevantes para o entendimento do modelo He–lastic:

- 1) o programa recebe de entrada um arquivo contendo um conjunto de múltiplas sequências moleculares alinhadas, referente à parcela do custo de processamento detalhado na [Equação 4.1](#);
- 2) a partir dos parâmetros selecionados pelo usuário é gerado um conjunto de tarefas para

Figura 18 – Diagrama ilustrando as principais diferenças entre os componentes das camadas de elasticidade: Apesar de compartilharem a estrutura de alto nível, o FaaS tem importantes limitações no que diz respeito ao tempo de execução e capacidade de processamento, o que permite ao provedor de computação em nuvem otimizar a gestão da elasticidade e multiplexar o uso de recursos entre usuários; em contrapartida, o orquestrador de contêineres permite que o usuário determine o comportamento da elasticidade e abdica da limitação no tempo de execução em troca de um maior fardo operacional e um ambiente de execução que deve ser gerido pelo próprio usuário.



Fonte: Elaborado pelo autor.

execução do teste de adequação dos sistemas de substituição moleculares, referente à parcela do custo de processamento detalhado na [Equação 4.2](#);

- 3) através do uso da biblioteca MPI, as tarefas são distribuídas entre um número fixo de recursos que compõem os nodos disponíveis no *cluster* computacional;
- 4) cada recurso do *cluster* recebe e processa um único teste de adequação para um sistema de substituição molecular, tendo como base os parâmetros informados e o arquivo de alinhamentos recebido ([Equação 4.3](#)), retornando uma lista de parâmetros numéricos que determina quão bem o sistema em questão representa os dados e os parâmetros que fizeram parte do teste de adequação.

Enquanto a estratégia adotada pelo jModelTest é muito eficaz na obtenção de altos níveis de performance, através da distribuição de tarefas via MPI, ela sofre com o desperdício de recursos computacionais em função da heterogeneidade no custo computacional do cálculo de adequação de sistemas de substituição de sequências moleculares. Este cenário é agravado quando há a

Tabela 15 – Configuração do processamento efetuado pelo jModelTest durante o uso de *Clustering Search* mostrando a quantidade e os sistemas de substituição molecular de acordo com a etapa.

Etapa	Quantidade	Sistemas
1	8	GTR, GTR+G, GTR+I, GTR+I+G, SYM, SYM+G, SYM+I, SYM+I+G
2	120	001234 $\times v$, 010234 $\times v$, 011234 $\times v$, 012034 $\times v$, 012134 $\times v$, 012234 $\times v$, 012304 $\times v$, 012324 $\times v$, 012334 $\times v$, 012340 $\times v$, 012341 $\times v$, 012342 $\times v$, 012343 $\times v$, 012344 $\times v$, TVM $\times v$, TVMef $\times v$
3	80	001203 $\times v$, 010203 $\times v$, 011203 $\times v$, 012003 $\times v$, 012103 $\times v$, 012203 $\times v$, 012300 $\times v$, 012301 $\times v$, 012302 $\times v$, 012303 $\times v$
4	48	001200 $\times v$, 010200 $\times v$, 011200 $\times v$, 012000 $\times v$, 012100 $\times v$, 012200 $\times v$
5	24	000100 $\times v$, 011000 $\times v$, 011100 $\times v$
6	8	JC, JC+G, JC+I, JC+I+G, F81, F81+G, F81+I, F81+I+G
Total	288	onde $v = \{ +F, +G, +G+F, +I, +I+F, +I+G+F, +I+G \}$

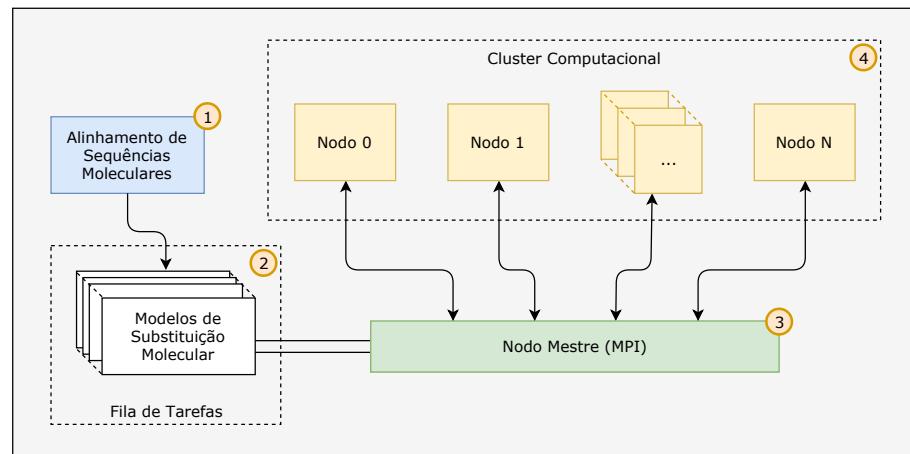
Fonte: Elaborado pelo autor.

aplicação da estratégia de *Clustering Search*, onde o total de sistemas de substituição é dividido em 6 grupos de teste de acordo com critérios de particionamento, que buscam, através de uma busca gulosa, testar até 288 sistemas, do total de 1624 suportados pelo jModelTest, em troca de uma precisão levemente pior, segundo afirmam Darriba et al. (2012) e Darriba (2015).

Embora a estratégia de *Clustering Search* represente uma heurística útil, sua implementação se dá através de 6 etapas de execução que lembram o modelo Bulk Synchronous Parallel (BSP) de computação, o que, por si só, não configura um problema, mas causa, na implementação do jModelTest, um aumento na ociosidade de recursos computacionais em função da alta variabilidade na quantidade de sistemas compreendidos em cada uma das etapas, conforme mostra a Tabela 15. Em um cenário hipotético de um *cluster* computacional com 288 *cores*, menos da metade seria totalmente ocupado durante a execução de uma análise através do jModelTest, sendo que somente em um cenário com 120 *cores* haveria completo uso dos recursos disponíveis. Contudo, tal uso ocorreria somente na etapa dois das seis totais, enquanto nas demais etapas, parte significativa destes recursos alocados estaria ociosa.

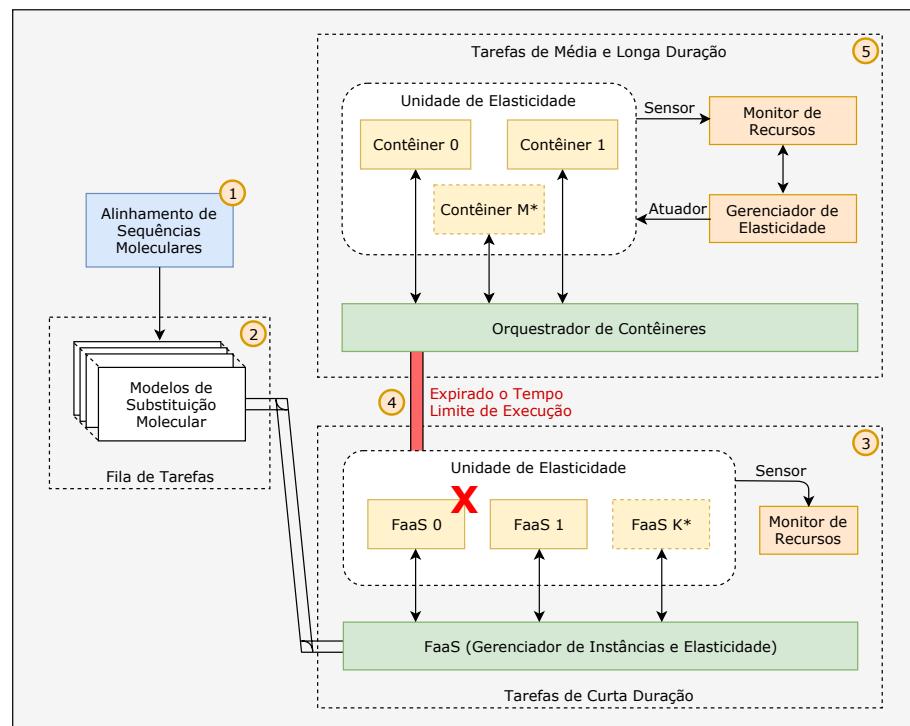
Ao assumir um cenário de execução do jModelTest com um *cluster* moderado para padrões atuais, contendo 64 *cores* de processamento, haveria, em uma mesma execução, contenção por falta de recursos nas etapas dois e três, enquanto na etapa quatro, o ambiente teria uso de 75% e nas demais apresentaria ociosidade significativa. Através deste exemplo fica evidente o ganho que uma estratégia de elasticidade pode prover ao efetuar um casamento mais apropriado entre demanda e recursos disponíveis. Desta forma, o modelo He–lastic lança mão da sua

Figura 19 – Estratégia de paralelismo adotada pelo programa jModelTest: um arquivo com múltiplas sequências moleculares alinhadas serve como entrada para um número de tarefas de avaliação (uma para cada sistema de substituição selecionado) e que ficam, por sua vez, em uma fila de tarefas e são processados conforme existam recursos disponíveis dentre um conjunto fixo preestabelecido.



Fonte: Adaptado de Darriba et al. (2014).

Figura 20 – Estratégia de paralelismo adotada pelo modelo He-lastic: um arquivo com múltiplas sequências moleculares alinhadas serve como entrada para um número de tarefas de avaliação (uma para cada sistema de substituição selecionado) que ficam, por sua vez, em uma fila de tarefas sendo primeiramente processadas na camada FaaS e encaminhadas para a camada de Orquestração de Contêineres em caso de falha por tempo de processamento expirado.



Fonte: Elaborado pelo autor.

estratégia dividida em duas camadas de elasticidade para, ao mesmo tempo, otimizar para: *a*) a heterogeneidade na complexidade computacional, e consequente tempo de execução, para o cálculo de adequação de sistemas de substituição molecular; e *b*) a ampla variação na quantidade de recursos requeridos durante execução quando empregando o método de *Clustering Search*; e *c*) manter um baixo impacto na eficiência (*overhead*) comparado com a performance original do jModelTest.

Comparada a abordagem original do jModelTest, representada pela Figura 19, a proposta do modelo He–lastic adiciona, na camada FaaS (de curta duração), um *buffer* ou colchão, capaz de absorver uma parcela das requisições de forma paralela, enquanto evita sobrecarregar a camada de longa duração com alta rotatividade de tarefas. Esta abordagem é ilustrada pela Figura 20, onde encontram-se destacados cinco pontos de especial atenção, que detalham as semelhanças e distinções entre a abordagem jModelTest, sendo eles:

- 1) o programa recebe de entrada um arquivo contendo um conjunto de múltiplas sequências moleculares alinhadas, assim como no jModelTest;
- 2) a partir dos parâmetros selecionados pelo usuário são geradas um conjunto de tarefas para execução do teste de adequação dos sistemas de substituição moleculares, assim como no jModelTest;
- 3) através de filas de mensagens o componente FaaS, primeira camada de elasticidade e responsável pelas tarefas de curta duração, é notificado de que há processamento aguardando, o que serve como gatilho para que o provedor de computação em nuvem aloque os recursos necessários de maneira elástica, de forma que cada recurso seja responsável por processar uma única requisição;
- 4) no caso em que um dos processamentos exceda o tempo limite de execução configurado para a camada FaaS, sua requisição é encaminhada para a segunda camada de elasticidade, o Orquestrador de Contêineres, onde são executadas as tarefas de média e longa duração;
- 5) cada nodo da unidade de elasticidade recebe e processa múltiplas requisições, sendo elas testes de adequação de sistemas de substituição molecular, onde um gerenciador de elasticidade configurável pelo usuário/administrador dá conta de alocar e desalocar recursos, conforme os parâmetros de *threshold* e a demanda, retornando uma lista de parâmetros numéricos que determina quão bem o sistema em questão representa os dados e os parâmetros que fizeram parte do teste de adequação.

4.5 Métricas de Avaliação

Uma vez definido o modelo He–lastic por meio das decisões que nortearam seu projeto, sua arquitetura e, principalmente, sua estratégia de elasticidade, resta detalhar como poderá ser avaliada sua performance e eficiência. Esta seção dedica-se a esta tarefa, ao estabelecer e

justificar métricas consideradas relevantes ao modelo e que possibilitam uma análise comparativa frente a possíveis alternativas e ao *software* jModelTest.

Dado que o modelo He–lastic se origina de uma necessidade real baseada nas deficiências do projeto jModelTest no que diz respeito ao seu aproveitamento de recursos, é natural intuir que a análise de desempenho se dará através de execuções reais do *software*, sendo este, efetivamente o caso. Toda a avaliação do modelo He–lastic se dá por meio de execuções reais e que poderiam plenamente ser usadas por pesquisadores da bioinformática nas suas análises de inferência filogenética.

Contudo, ao contrário de abordagens que se apoiam em *benchmarks* sintéticos, como o cálculo de aproximação para integrais ou o LINPACK de Dongarra, Luszczek e Petitet (2003), o uso de avaliações reais impõem restrições no que diz respeito à definição do grão de paralelismo computacional. Trabalhos como o de Kwiatkowski (2002) comprovam a importância da escolha adequada da granularidade em cenários de computação paralela e distribuída, podendo este ser ajustado, no caso de avaliações sintéticas, facilmente através da regulagem dos parâmetros do algoritmo usado na computação. Abordagens como a de Shankar et al. (2018), que explora o paralelismo massivo das FaaS, mencionam especificamente a necessidade de ajustar a granularidade de tarefas para otimizar a computação:

contanto que seja escolhida a granularidade de tarefas de forma que a maioria das tarefas possam ser concluídas com sucesso no intervalo de tempo alocado, não foi observada uma penalidade muito grande de desempenho pelo encerramento de um processo por conta do tempo limite de execução (*timeout*)

Embora não seja possível, no caso do modelo He–lastic, controlar explicitamente o nível de granularidade, existem formas de fazer esta regulagem através dos parâmetros de execução do cálculo de adequação de sistemas de substituição de sequências moleculares. Para este fim, são de grande valia o conjunto de equações detalhadas na Seção 4.3 que abordam o custo computacional em função do conteúdo do arquivo de múltiplos alinhamentos de sequências e os parâmetros da execução do teste de adequação, culminando na Equação 4.3. Desta forma, para os fins deste trabalho, define-se a granularidade do paralelismo conforme:

Premissa 1 — da Granularidade: a granularidade de uma execução do modelo He–lastic e, consequentemente, do *software* jModelTest, pode ser determinada através da combinação entre o conteúdo do arquivo de múltiplos alinhamentos de sequências moleculares, fornecido como parâmetro de entrada, e os parâmetros escolhidos pelo usuário que determinarão, em última instância, quais sistemas de substituição farão parte do teste de adequação.

De posse da definição de granularidade, útil tanto para o modelo He–lastic quanto para o jModelTest, há, ainda, a necessidade de estabelecer as métricas de performance. Iniciando pelo

jModelTest e seu modelo de computação distribuída de recursos fixos, o uso de uma formulação já existente se torna possível e recomendável com vistas a garantir a comparação com resultados similares. Tomando por base as definições de Energia e Custo, regularmente utilizadas nas publicações oriundas do Programa de Pós–Graduação da UNISINOS, é possível especificar o Custo da execução do jModelTest, e por consequência, de recursos fixos, como sendo:

$$Custo_{fixo}(t, j) = t \times j \quad (4.4)$$

onde t representa o tempo total de execução e j representa a quantidade de *cores* de processamento disponíveis. A ausência de comportamento elástico garante uma formulação simples e direta, capaz de representar o custo do ponto de vista energético. Uma vez que o presente trabalho também se interessa pelo custo financeiro da computação é relevante definir também a Equação 4.5 que se utiliza da Equação 4.4:

$$Financeiro_{fixo}(m, t, j) = m \times t \times j \quad (4.5)$$

onde m representa o custo financeiro por unidade de tempo t .

Visto que a elasticidade é central para a proposta do modelo He–lastic, devem ser definidas métricas análogas aquelas das Equações 4.4 e 4.5. Tendo em vista que a elasticidade é a capacidade de adicionar e remover recursos de maneira dinâmica de um ambiente de computação em nuvem, é necessária uma maneira de representar estes custos ao longo do tempo, onde a métrica de Energia adota uma abordagem análoga a um histograma, ao representar, ao longo de unidades de tempo pt_e , que também pode ser interpretado como número de amostras, a quantidade de recursos disponíveis j , no caso deste trabalho adotado como *cores de processamento*, resultando na seguinte fórmula:

$$Energia(i, s) = \sum_{i=j}^s (j \times pt_e(j)) \quad (4.6)$$

sendo i e s , respectivamente, os limites mínimo e máximo de *cores* de processamento alocáveis durante a execução. A Equação 4.6 se transforma, por sua vez, na métrica de custo com a introdução tempo total de execução da aplicação:

$$Custo_{elastico}(t, i, s) = t \times Energia(i, s) \quad (4.7)$$

onde, por fim, através da introdução do componente m , que representa o custo financeiro por unidade de tempo t , obtém-se a equação para o custo financeiro em cenários elásticos:

$$Financeiro_{elastico}(m, t, i, s) = m \times Custo_{elastico}(t, i, s) \quad (4.8)$$

Embora as equações 4.6, 4.7 e 4.8 deem conta do cenário usual de elasticidade, elas se mostram insuficientes quando confrontadas com o cenário de elasticidade via FaaS, fundamental

Tabela 16 – Comparativo entre as principais variáveis que compõem a equação de Custo (Equação 4.9) para o modelo FaaS de acordo com os limites impostos por três grandes provedores de computação em nuvem.

	AWS ‘18	AWS ‘17	Azure	GCP
y	$2 \leq y \leq 47$	$2 \leq y \leq 24$	$1 \leq y \leq 12$	$y \in \{2, 4, 8, 16\}$
F_{mem} (MB)	64	64	128	128
c	y	y	1	$\begin{matrix} c \in \\ \{1, 2, 4, 7, 12\} \end{matrix}$
F_{cpu} (MHz)	102	100	2400	200
t	≤ 9000	≤ 3000	≤ 6000	≤ 5400
F_{tempo} (ms)	100	100	100	100

Fonte: Atualizado pelo autor, adaptado de [Wang et al. \(2018\)](#).

para a compreensão do modelo He–lastic. Tendo por referência as características elencadas na [Tabela 5](#), apresentada na [Subseção 2.2.5](#), são necessários alguns ajustes para obter uma equação capaz de formalizar os conceitos de energia e custo no cenário FaaS, detalhados a seguir.

Em um primeiro momento é necessário descartar o conceito de Energia, uma vez que, no modelo FaaS, as execuções são contabilizadas individualmente e existe a alocação fracional de recursos, o que gera uma incompatibilidade com a definição de Energia Elástica. Em seguida, é necessário remodelar a definição de Custo, baseada na [Equação 4.6](#), para que se ajuste à realidade observada durante o uso de FaaS, resultando em:

$$Custo_{faas}(y, c, t) = ((y \times F_{mem}) \times (c \times F_{cpu}) \times (t \times F_{tempo})) \quad (4.9)$$

onde y e c são inteiros, ou então, serão arredondados para cima, representando, respectivamente: alocação de memória, alocação de CPU e tempo de execução. Uma vez que cada um dos principais provedores de computação em nuvem estabelece limites e modelos de especificação diferentes, se torna relevante a visualização dos elementos contidos na [Equação 4.9](#) na forma de uma tabela, como visto na [Tabela 16](#). Assim como a equação de custo financeiro para o cenário elástico se originou da definição de custo, o mesmo procedimento pode ser adotado no cenário de FaaS, resultando na seguinte equação:

$$Financeiro_{faas}(m, y, c, t) = m \times Custo_{faas}(y, c, t) \quad (4.10)$$

onde m , tal qual na [Equação 4.8](#), determina um componente de custo financeiro por unidade de tempo, podendo assumir um valor monetário como R\$ 5,25, 1,5€ ou \$ 2,75, por exemplo.

Contudo, por convenção e padronização quanto às práticas de precificação adotadas pelos principais provedores de computação em nuvem, a unidade monetária adotada para m ao longo deste texto será o Dólar Americano (\$), salvo indicação do contrário.

Determinar uma métrica de avaliação unificada torna-se uma tarefa não trivial, considerando as particularidades, no que tange ao cálculo de Energia e Custo, conforme os modelos de execução por recursos fixos, com uso de elasticidade e com uso de FaaS. Desta forma, mantendo-se fiel ao conceito de *utility-computing* habilitado pela computação em nuvem, a métrica referente ao custo financeiro foi escolhida como fator de equalização entre os diferentes modelos de execução e será usada no restante deste trabalho. Uma desvantagem em relação à métrica de custo financeiro está na sua variabilidade ao longo do tempo e entre provedores de computação em nuvem, uma vez que preços tendem a se alterar conforme são disponibilizadas novas funcionalidades e novos tipos de hardware.

Premissa 2 — da Métrica de Avaliação: nos cenários que contemplem a composição entre múltiplos modelos de execução (fixo, elástico e FaaS), a métrica Financeira será utilizada como baliza para determinar as relações de eficiência. Tal definição se apoia na associatividade de custos, definida por [Armbrust et al. \(2010\)](#), ao afirmar que, em um ambiente de computação em nuvem usar 1000 máquinas por uma hora equivale a usar uma máquina por 1000 horas.

Contudo, para os fins deste trabalho, esta desvantagem foi anulada através do uso de um único provedor de computação em nuvem e uma mesma família de hardware para as máquinas virtuais, decisões que são abordadas em detalhes na [Seção 5.3](#). Ainda assim, a métrica de custo financeiro, além de ser um forma eficaz de prover uma linha justa de comparação entre as três abordagens, é resiliente por naturalmente conter em sua composição uma análise de energia e custo, ainda que realizada pelo provedor. Resultando, enfim, nas definições 2 e 3:

Premissa 3 — do Provedor de Computação em Nuvem: para os fins do presente trabalho fica estabelecida a restrição referente ao uso de um único provedor de computação em nuvem. Esta restrição não é imposta pelo modelo He–lastic, que é indiferente a uma possível implantação *multi–cloud* em função da sua arquitetura desacoplada através de componentes independentemente escaláveis, contudo tem por objetivo manter uma comparação justa entre os diferentes modelos de execução, evitando discrepâncias no que tange à performance dos recursos computacionais subjacentes e à precificação dos mesmos.

4.6 Considerações Parciais

Assim, elevando as capacidades da estratégia original do projeto jModelTest ([Figura 19](#)), o modelo proposto neste trabalho ([Figura 20](#)) promove uma dupla camada de elasticidade: (i) com o componente FaaS em um primeiro nível, absorvendo os picos de demanda e a parcela de requisições de curta duração, enquanto delega a gestão do ambiente e, principalmente, da elasticidade para o provedor de computação em nuvem evitando, assim, custos com ociosidade; e

(ii) com o Orquestrador de Contêineres no segundo nível, provendo a capacidade necessária para a execução de tarefas de média e longa duração, enquanto reduz o esforço operacional e a complexidade da escolha dos parâmetros de elasticidade em função do seu uso reduzido, dado que parte dos cálculos poderá ser absorvida pela camada anterior.

Representando a principal contribuição do modelo proposto para o estado da arte, a divisão do tratamento de elasticidade em duas camadas ([Figuras 16, 17, 18 e 20](#)) permite a adaptação às características computacionais do teste de adequação de sistemas de substituição de sequências moleculares, nomeadamente a irregularidade no esforço computacional dos cálculos (representado pela [Figura 14](#)), enquanto reduz significativamente os custos com infraestrutura em comparação com a abordagem clássica de computação distribuída (baseada em *clusters* ou *grids*), conforme utilizada pelo projeto jModelTest (vide [Figura 19](#)).

Além disso, o modelo delega a gestão de tarefas secundárias¹² consequentemente reduzindo o fardo operacional enquanto se mantém agnóstico a provedores de computação em nuvem, permitindo amplas possibilidades de adoção tanto no meio acadêmico quanto comercial. Como forma de possibilitar a comparação entre as abordagens do projeto jModelTest, assim como a elasticidade das camadas FaaS e de Orquestração de Contêineres, a métrica de avaliação selecionada neste trabalho é o custo financeiro, uma vez que ele se apresenta como denominador comum à todos os serviços de computação em nuvem. A seguir o trabalho prossegue para especificar a metodologia de avaliação, fornecendo detalhes a respeito dos cenários de teste, as características do protótipo desenvolvido, assim como informações a respeito da aplicação e a infraestrutura de computação em nuvem utilizada.

¹² a expressão *undifferentiated heavy-lifting* costuma ser usada no contexto de computação em nuvem para caracterizar tais esforços operacionais, sendo alguns exemplos, a configuração e manutenção de redes, *firewalls* e segurança, aplicação de *patches* e atualizações à nível de sistema operacional, entre outros.

5 METODOLOGIA DE AVALIAÇÃO

“Software maintenance is not ‘keep it working like before’.
It is ‘keep being useful in a changing world’.”

Jessica Kerr

Neste capítulo é apresentada a metodologia empregada para avaliar o modelo proposto neste trabalho. Por basear-se em um projeto de *software* já desenvolvido e amplamente utilizado, o jModelTest, foi adotada uma abordagem de comparação entre o cenário atual, ou ‘o antes’, com os cenários habilitados pelo modelo proposto, o ‘o depois’. Para tanto são definidos diversos cenários de avaliação capazes de exercitar as diferentes características dos projetos sob análise, assim como os parâmetros utilizados na execução. Também são descritas neste capítulo as principais características da aplicação jModelTest e que influenciaram nas escolhas da metodologia de avaliação, assim como o ambiente de computação em nuvem e detalhes da implementação do protótipo desenvolvido neste trabalho.

Para realizar a avaliação do modelo He-lastic e futura comparação com o *software* jModelTest foi desenvolvida uma aplicação à título de protótipo, que contemplasse na sua implementação as principais ideias e diferenciais propostos pelo modelo. Todo código utilizado pelo protótipo, assim como o código do jModelTest, é publicamente acessível e encontra-se disponibilizado no portal GitHub, conforme apresentado na Tabela 17.

O protótipo foi desenvolvido utilizando a linguagem Python, na sua versão 3.6, no ambiente do provedor de computação em nuvem AWS, utilizando, portanto, suas APIs e serviços disponíveis como blocos de construção. A abordagem adotada para replicar o funcionamento do jModelTest foi através da repetição de *traces* de execução, o que exigiu pequenas modificações¹ no projeto original. Estas decisões são detalhadas a seguir.

5.1 Etapas e Cenários

Visando fornecer uma rigorosa análise a respeito do modelo He-lastic, é necessário estabelecer uma base comum para que esta possa servir de comparação às futuras conclusões no que diz respeito ao modelo, seus modos de execução e as métricas detalhadas no Capítulo 4. A escolha dos dados utilizados para os testes é particularmente relevante para o cenário de comparação com o jModelTest em função dos fatores determinados na Seção 4.2 e principalmente da Equação 4.3.

De posse dos dados, que servirão como bases aos cenários de testes, é possível iniciar na montagem e configuração de ambientes e na definição dos cenários de testes, tanto para o

¹ As alterações realizadas para viabilizar a geração de *traces* não alteraram o comportamento da aplicação e podem ser vistas no comparativo disponível no endereço: <https://github.com/mateusaubin/jmodeltest2/compare/8242fc..master?diff=split>.

Tabela 17 – Repositórios contendo o código-fonte dos projetos utilizados na avaliação do modelo He-lastic.

Repositório	Finalidade	Acessível em
modeltest-lambda	Protótipo do modelo He-lastic	https://github.com/mateusaubin/modeltest-lambda
modeltest-loadexerciser	Scripts de execução dos cenários de avaliação e parsers para coleta de estatísticas	https://github.com/mateusaubin/modeltest-loadexerciser
jmodeltest2	Versão modificada para geração de <i>traces</i> de execução	https://github.com/mateusaubin/jmodeltest2
	Fontes originais do jModelTest	https://github.com/ddarriba/jmodeltest2
PhyML	Motor de cálculo de adequação de sistemas de substituição molecular	https://github.com/stephaneguindon/phym

Fonte: Elaborado pelo autor.

projeto jModelTest, quanto para o modelo He-lastic. Por fim, os esforços são direcionados à coleta e análise dos dados obtidos através dos cenários de teste executados, sem deixar de lado a preocupação quanto a qualidade dos resultados obtidos, que deve ser avaliada por meio de análises estatísticas e de consistência, garantindo a pureza e relevância dos dados.

Este protocolo resulta em quatro etapas que dizem respeito ao processo de avaliação do modelo He-lastic, com a notável exceção sendo o desenvolvimento do protótipo que será abordado em detalhes nas próximas seções. Abaixo são descritas as quatro etapas da avaliação e seus respectivos cenários:

5.1.1 Parâmetros e Conjuntos de Dados

Esta etapa foi caracterizada por encontrar bases de dados padronizadas e amplamente utilizadas para testes no âmbito da filogenética, com o objetivo de estabelecer uma base comparativa que possa ser reproduzível por pesquisadores externos e que permita ampla avaliação do modelo proposto. Estudos preliminares indicavam a existência de tais *datasets*, conforme citam os trabalhos de [Hordijk e Gascuel \(2005\)](#), [Keane \(2006\)](#), [Stamatakis \(2005\)](#) e [Stewart et al. \(2001\)](#), contudo alguns destes autores levantam dúvidas quanto a qualidade e padronização dos *datasets*.

Através de comunicação pessoal por e-mail com Stamatakis e Darriba foi possível concluir que, de fato, não existem *datasets* amplamente aceitos pela comunidade no que diz respeito à performance de testes de adequação de sistemas de substituição filogenética. A recomendação recebida foi por utilizar os alinhamentos de exemplo que acompanham os binários do software jModelTest, assim como elencar um subconjunto dos arquivos coletados em um repositório disponível no portal GitHub². Embora exista uma quantidade significativa de alinhamentos

² Disponível na URL <https://github.com/stamatak/test-Datasets>.

Tabela 18 – Conjuntos de dados utilizados na execução dos cenários de testes para o modelo He-lastic e o jModelTest.

#	Arquivo	Sequências	Comprimento	Quantidade de Sistemas de substituição	Tempo Total de Execução
01 – aP6		6	631	288	0:00:40
02 – rodents		8	1078	256	0:03:20
03 – example		10	1000	288	0:04:42
04 – 18S_insects2		7	902	256	0:04:52
05 – HIVpol.groupM		8	3009	256	0:11:58
06 – Hex_EF1a		9	1092	256	0:20:12
07 – primate-mtDNA		12	898	288	0:24:23
08 – HIV_vpu.ref2		35	392	280	0:45:38
09 – gusanos16S.mafft		43	492	256	1:44:52
10 – Birds		9	14043	208	2:11:50
11 – gusanosCOI.mafft		44	561	256	2:58:49
12 – stamatakis-59		59	6951	128	14:51:54

* Tempo de execução obtido através da média das execuções do software jModelTest realizando o teste de adequação para 288 sistemas de evolução em uma máquina com 2 processadores.

Fonte: Elaborado pelo autor.

presentes no repositório indicado, sua utilização se mostrou proibitiva em função do tempo de execução exigido e, consequentemente, a necessidade de um grande *pool* computacional à disposição.

Tendo em vista que uma das diretrizes de projeto do modelo He-lastic está na sua capacidade de lidar com amplas variações no que diz respeito a carga computacional, o uso de arquivos que ultrapassem o tempo de execução de 24 horas foi julgado não relevante, uma vez que, do ponto de vista dos resultados, tais testes não adicionariam valor por apresentar comportamento muito similar dentre este conjunto. A escolha por evitar alinhamentos de grandes dimensões também gera efeito no custo financeiro requerido para executar os cenários de testes do modelo e do jModelTest, além de contribuir significativamente no tempo necessário para executar cada cenário, de forma que a escolha por deixar tais arquivos de fora do conjunto de dados permitiu uma maior variação nos cenários de testes em função do tempo de execução reduzido.

Assim foram estabelecidos 12 arquivos com ampla variação no tempo de execução que serão usados para as execuções de avaliação do modelo He-lastic e do jModelTest. Dos 12 arquivos apenas um deles se origina do repositório indicado por Stamatakis, enquanto os demais são oriundos do próprio projeto jModelTest. Esta decisão foi motivada pelas circunstâncias expostas anteriormente e colhe apoio no fato de que favorece análises comparativas e facilita

Tabela 19 – Parâmetros disponibilizados pelo jModelTest que influenciam diretamente na quantidade de sistemas de substituição molecular que farão parte do teste de adequação.

Parâmetro	Descrição
Esquemas de Substituição	Quantidade de sistemas de substituição molecular a ser incluídos no teste de adequação [†] Opções disponíveis são {3, 5, 7, 11, 203}
+F	Controla a inclusão de sistemas com frequências base iguais ou desiguais
+I	Controla a inclusão de sistemas com ou sem uma proporção de locais invariantes em uma sequência molecular
+G _n	Controla a inclusão de sistemas com ou sem variação na taxa de substituição por local em uma sequência molecular <i>n</i> pode ser um número arbitrário que define a quantidade e categorias de variação

[†] A Subseção 2.1.3 apresenta uma discussão detalhada a respeito dos sistemas de substituição, assim como a Tabela 4 que apresenta alguns dos sistemas mais populares

Fonte: Elaborado pelo autor.

a avaliação dos resultados por terceiros. A Tabela 18 apresenta os arquivos e algumas de suas características, assim como uma referência do tempo de execução máximo obtido por execuções em uma instância com dois núcleos de processamento disponíveis.

No que diz respeito aos parâmetros de execução das avaliações de adequação dos sistemas de substituição molecular, o objetivo principal é obter uma variedade que represente bem o cenário de um pesquisador da filogenética. Conforme pode ser visto na Tabela 19, a escolha de sistemas de substituição está sob influência de quatro configurações disponibilizadas pelo *software* jModelTest, onde o usuário pode selecionar quantidade de sistemas para teste e modificadores de comportamento. Considerando este perfil de usuário foi estabelecido que os parâmetros de execução serão os mais comprehensivos possíveis, empregando o maior número de avaliações para obter, desta forma, um amplo panorama acerca das características de execução de cada sistema de substituição molecular.

No jModelTest este cenário se dá através das utilização de *Clustering Search*, conforme detalhado na Seção 4.4, que adota uma heurística capaz de reduzir os 1624 sistemas disponíveis para teste em 6 conjuntos que somados totalizam 288 sistemas (vide Tabela 15). Também embasa esta escolha o fato de que parâmetros menos comprehensivos de execução do jModelTest resultam em apenas 88, 40 ou 24 execuções paralelas, um número considerado baixo para colocar à prova as capacidades de paralelismo e elasticidade da camada FaaS do modelo He-lastic, uma vez que não é incomum encontrar servidores com mais de 16 *cores* de processamento e VMs de provedores de computação em nuvem com mais de 30 núcleos.

Cabe ressaltar, mais uma vez, que a escolha dos arquivos de alinhamentos de sequências moleculares é particularmente relevante para o cenário de comparação com o jModelTest em

função dos fatores determinados na [Seção 4.2](#) e, principalmente, através dos componentes da [Equação 4.3](#), tornando o arquivo parte fundamental do grão de paralelismo juntamente com os parâmetros do jModelTest, detalhados na [Tabela 19](#), que influenciam diretamente na escolha dos sistemas de substituição incluídos no teste dentre os 288 possíveis, como mostra a [Tabela 15](#).

5.1.2 Estratégia de Avaliação do jModelTest

Para viabilizar uma comparação justa entre a abordagem de recursos fixos adotada pelo jModelTest, com a estratégia de elasticidade empregada pelo modelo He–lastic, existe a necessidade de coletar resultados de execução em um ambiente similar, reduzindo as variações em função de possíveis versões de dependências, modelos de hardware, situações de carga do ambiente e outras características inerentes a execução de softwares distribuídos e de alta performance. Desta forma, a próxima etapa na avaliação do modelo consiste na execução do jModelTest com os arquivos e parâmetros, que determinam o grão de paralelismo, previamente selecionados em um ambiente similar ao que encontrará o protótipo do modelo He–lastic.

Este processo determinará o patamar básico de performance, também chamado de *baseline*, que atua como balizador para os resultados oriundos do modelo proposto. Apesar das diferenças no modelo de execução, as métricas coletadas nesta etapa são fundamentais para estabelecer uma linha de comparação quando confrontadas com os resultados obtidos pelo modelo He–lastic, exigindo, portanto, o mesmo rigor na execução e coleta dos resultados.

São fornecidos pelo *software* jModelTest, dois modelos de execução paralela, sendo o primeiro deles baseado em memória compartilhada e *threads* em um ambiente não distribuído, e o segundo adotando a biblioteca MPI como intermediador para a execução em *clusters* ou *grids* computacionais distribuídos. Ambos os modos de execução estão disponíveis em um único binário desenvolvido, como indica o nome, na linguagem Java com a biblioteca MPJ Express ([BAKER; CARPENTER; SHAFI, 2006](#)) constituindo a principal diferença entre eles. Além disso, o programa permite tanto a execução através de console interativo via linha de comando, quanto por uma interface gráfica, o que segundo a [Tabela 10](#) presente na análise do Estado da Arte realizada na [Seção 3.3](#), pode ser considerado um diferencial, haja vista que apenas 41% dos trabalhos estudados contemplavam esta possibilidade.

Através de uma análise do código fonte é possível perceber que as funções de cálculo de adequação dos sistemas de substituição molecular são compartilhadas entre as quatro variantes de execução apresentadas, o que permite concluir que há uma significativa equivalência no esforço computacional entre elas, ainda que seja esperado um declínio na eficiência durante a operação baseada em MPI em função do *overhead* de coordenação e comunicação entre os nodos do *cluster*.

Durante a operação normal do jModelTest é possível observar dois comportamentos de execução, variando conforme a adoção ou não, da técnica de *Clustering Search*, onde o modo que não utiliza esta técnica se comporta de maneira muito similar a um mestre–escravo ou até mesmo

um *scatter-gather*, embarrasosamente paralelo ao distribuir para cada nodo/processador um sistema de substituição de sequências para que seja executado o teste de adequação e agregando os resultados antes de devolver uma resposta ao usuário. Nos casos onde há o uso de *Clustering Search*, seu modo de operação se assemelha ao de fases paralelas (BSP), uma vez que existem seis fases de computação com barreiras, entre elas, que são utilizadas pela heurística para determinar o ponto de parada do algoritmo. Embora não haja comunicação lateral entre os nodos de execução, sendo esta limitada a uma arquitetura mestre–escravo, a existência de uma barreira de sincronização é julgada suficiente para justificar a associação ao modelo BSP. Este modo de execução pode gerar desperdício significativo de recursos em função da ociosidade ocasionada pela ampla variação no número de testes de adequação que são executados e, portanto, paralelizáveis em cada uma das 6 etapas que compõem a *Clustering Search*, uma fraqueza do jModelTest já abordada na Seção 4.4 e exemplificada na Tabela 15.

Para atender aos objetivos do presente trabalho foi selecionada a estratégia de execução baseada em *threads* por memória compartilhada. Esta decisão teve como objetivo coletar resultados do jModelTest em um cenário de *overhead* mínimo, obtendo, assim, os melhores valores possíveis para a execução de cada um dos arquivos que compõem o *dataset* de testes. Uma comparação baseada na execução via MPI teria embutida em si todos os custos adicionais de comunicação, distribuição e coordenação entre nodos de processamento, contaminando os resultados. Embora uma avaliação por este cenário possa ser desejável ao avaliar um sistema baseado em elasticidade, no contexto deste trabalho a execução de testes no cenário MPI incorreria em um elevado esforço de execução em função da configuração necessária para estabelecer um ambiente propriamente configurado e a dificuldade de automação desta tarefa, limitando, portanto, a capacidade de coleta de dados e confiabilidade dos resultados.

Outro fator que contribuiu para a escolha do cenário baseado em *threads*, deve-se ao fato de que já é rotineiro encontrar disponível nos provedores de computação em nuvem VMs com quantidade de cores superior a 30, um número considerado suficiente para avaliar o impacto da estratégia de *Clustering Search* na eficiência computacional, conforme variação no número de nodos de processamento disponíveis. Através dos testes, adotando uma curva ascendente na quantidade de recursos disponíveis será possível, ao mesmo tempo que se determina um nível base de performance, verificar o impacto da estratégia *Clustering Search* (e a variação no nível de paralelismo que ela causa, conforme a Tabela 15) na eficiência.

Conclui-se, portanto, que a avaliação do jModelTest e, consequente base comparativa para o modelo **H_e-lastic** se dará por meio da execução de múltiplas rodadas do teste de adequação em execuções em uma VM de cada vez, variando a quantidade de núcleos de processamento disponíveis, duplicando-os ou variando conforme disponibilidade do provedor de computação em nuvem, contemplando um intervalo aproximado de 2 à 40 *cores*. A Tabela 20 apresenta possíveis cenários de avaliação utilizando instâncias da família c5, disponível no provedor AWS com a nomenclatura C_{zc} , onde, z representa jModelTest e c o número de CPUs utilizadas. Por meio desta estratégia serão obtidos dados mais otimistas possíveis, ao eliminar qualquer

Tabela 21 – Cenários de avaliação para o modelo He–lastic.

Tabela 20 – Cenários de avaliação para o software jModelTest.

Cenário	CPUs	Orquestrador de Contêineres	FaaS	
			Cenário	CPUs
C_{f0}			-	1536
C_{c8}			8	-
C_{c16}			16	-
C_{c36}			36	-
C_{j2}	2	C_{m1}	8	1536
C_{j4}	4	C_{m2}	16	1536
C_{j8}	8	C_{m3}	16	1536
C_{j16}	16	C_{m4}	36	60
C_{j36}	36	C_{m5}	36	45
Fonte: Elaborado pelo autor.		C_{m6}	36	60
		C_{m7}	36	15
			36	30
			36	60
			768	60

Fonte: Elaborado pelo autor.

overhead oriundo da computação distribuída e que permitirão estabelecer uma comparação rigorosa quanto ao consumo de recursos baseados em elasticidade do modelo He–lastic, além de, avaliar a atual eficiência no uso de tais recursos. Os resultados obtidos por meio desta estratégia serão abordados em detalhes na [Seção 6.2](#).

5.1.3 Estratégia de Avaliação do He–lastic

A avaliação do modelo proposto se dará por meio da sua execução em ambiente computacional o mais próximo possível daquele onde se dará a coleta de métricas sobre o software jModelTest, mantendo configurações de provedores de computação em nuvem, tipos e variedades de instâncias/VMs e sistemas operacionais. Tais decisões visam maximizar a quantidade de variáveis controladas o que, por sua vez, reduz a incidência de ruídos ou influências indesejadas nos resultados coletados, visando comparação entre o modelo He–lastic e o jModelTest. Conforme esperado, todos os testes devem ser executados com o mesmo conjunto de dados e parâmetros previamente definidos de forma que uma comparação justa possa se estabelecer entre as abordagens comparadas.

Os resultados obtidos nesta etapa tem a tripla finalidade de determinar: 1) a variação das métricas de avaliação comparadas às diferentes execuções do jModelTest; 2) o *overhead* causado pela abordagem de camadas de elasticidade adotada pelo modelo proposto; e 3) a variação

de performance, conforme são desabilitadas as camadas do modelo. Em última instância, os dados obtidos indicarão os pontos fortes e fracos da estratégia proposta, o que permitirá uma avaliação no que diz respeito a viabilidade da sua adoção, assim como um direcionamento quanto a possíveis trabalhos futuros em relação ao modelo He-lastic.

A flexibilidade do modelo proposto, através da sua dupla camada de elasticidade, se manifesta por meio dos parâmetros de configuração previamente detalhados na Tabela 13 especificando: *a) Tempo Limite; b) Potência; c) Número de CPUs; d) threshold inferior; e e) threshold superior*, sendo os dois primeiros relevantes para a camada FaaS e os demais para a camada de Orquestração de Contêineres. Contudo, esta flexibilidade acaba tornando-se um complicador ao estabelecer os cenários de testes, devido a explosão combinatória causada pelos cinco parâmetros do modelo, o que exige uma redução no espaço de busca para viabilizar a execução das avaliações.

No caso da camada baseada em FaaS, a associatividade de custos e proporcionalidade na alocação de recursos permite que as variações sejam concentradas em um dos dois parâmetros relevantes para esta camada, desta forma, o parâmetro de *Timeout* (tempo limite de execução) será utilizado como principal elemento de variação, com a potência sendo usada para confirmar a associatividade de custos da camada FaaS. Quanto a Orquestração de Contêineres foi definido que apenas o parâmetro relativo a quantidade de CPUs será variado, fixando-se os valores de controle do comportamento de elasticidade e alinhando o número de CPUs com aqueles adotados ao estabelecer as bases de comparação. Esta decisão visa manter uma comparação justa entre os cenários ao reduzir a quantidade de variáveis que podem influenciar nos resultados e foi considerada adequada para determinar as características de comportamento do modelo He-lastic, haja vista, a grande quantidade de combinações possíveis entre thresholds.

Contudo, a variação nos parâmetros do modelo não permite detectar os níveis de *overhead* incorridos da sua utilização, requerendo uma abordagem alternativa para obtenção deste resultado. Conforme detalhado na Seção 4.3, o modelo He-lastic se diferencia em função da sua abordagem de duas camadas de elasticidade que, em função de suas características, se adéquam, em primeira análise, para diferentes tipos de aplicações, mas que, quando combinadas, podem contribuir para uma determinada classe de tarefas, como o teste de adequação de sistemas de substituição molecular. Desta forma emergem três cenários de avaliação oriundos da combinação das camadas, nomeadamente os cenários: 1) misto, contemplando ambas camadas habilitadas; 2) apenas FaaS; e 3) apenas Contêineres. Um quarto cenário, desabilitando ambas as camadas de elasticidade, embora possível, foi descartado por não ser relevante para o modelo proposto, uma vez que este é caracterizado pelo uso da elasticidade, assim, um cenário sem elasticidade forneceria uma caracterização deturpada do modelo He-lastic, além de exigir esforço de desenvolvimento e coleta de dados.

Os cenários elencados para avaliação do modelo são apresentados na Tabela 21, assim como um detalhamento a respeito dos valores atribuídos para cada um dos parâmetros relevantes à avaliação. Tal como na Tabela 20, o cenário assume trabalhar com instâncias do tipo c5 do provedor AWS e sua nomenclatura segue o formato C_{zc} , onde z pode assumir três valores {FaaS,

Contêineres, Misto} e c representa um sequencial à exceção dos cenários por Contêineres, onde representa o número de CPUs (que, por sua vez, está alinhado aos valores definidos previamente visando uma avaliação justa). Merecem destaque, os cenários C_{f0} e C_{m7} por seus objetivos, sendo, respectivamente, examinar o ponto de limite no tempo de execução de tarefas e validar a associatividade de custos na camada FaaS, assim como, a proporcionalidade na alocação de recursos. Os cenários C_c têm por objetivo avaliar o nível de *thrashing* e o *overhead* causado pela execução de tarefas de curto prazo. Por fim, os cenários C_m colaboram para obter uma caracterização do comportamento dinâmico do modelo He–lastic no que diz respeito às camadas de elasticidade, além de medir sua eficiência.

5.1.4 Análise Comparativa

Uma vez obtidos os resultados referentes aos cenários de teste do jModelTest, assim como os do modelo He–lastic, a avaliação passa a se concentrar no estudo e comparação entre os resultados obtidos. De posse dos dados brutos, conforme gerados pelo jModelTest, assim como pelo protótipo, estes serão compilados e tabulados para então passar por uma inspeção e limpeza, em busca de *outliers*, de forma a evitar a ocorrência de viéses nos dados em função de anomalias, para que então possam servir como fontes adaptadas para análise. Estas análises serão guiadas pelas métricas estabelecidas no [Capítulo 4](#), sendo de especial interesse os resultados relativos ao tempo total de execução e custo financeiro, que permitirão determinar os pontos fortes e fracos da proposta com relação à abordagem de recursos fixos adotada pelo jModelTest. Além disso, também despertam interesse às características de elasticidade e a dinâmica de relacionamento entre as camadas FaaS e de Orquestração de Contêineres propostas pelo modelo He–lastic.

5.2 Aplicação

Conforme já detalhado na [Seção 4.1](#), a aplicação que serve de base para a implementação do protótipo do modelo He–lastic é o jModelTest, um projeto já estabelecido no cenário de bioinformática e filogenética, que conta com amplo uso no meio acadêmico. Seu objetivo é o cálculo de adequação de sistemas de substituição molecular, também conhecido como *best-fit*, onde diversos sistemas de substituição são testados para determinar qual deles melhor explica as substituições moleculares que ocorreram em um determinado conjunto de dados. Como resultado deste modo de operação, foi possível formalizar na [Equação 4.3](#) o cálculo referente a complexidade, ou custo computacional, de uma execução do jModelTest. Outra característica relevante do projeto jModelTest, no que diz respeito ao protótipo, é a sua execução dividida em fases quando é utilizada a técnica de *Clustering Search*, lembrando um algoritmo baseado em BSP, como é o caso dos cenários de avaliação estabelecidos para o presente trabalho.

Durante a implementação do protótipo foi possível explorar uma particularidade do jModelTest para obter o duplo benefício de agilizar o desenvolvimento e garantir a igualdade dos

Figura 21 – Linha de comando utilizada pelo jModelTest para controlar o *software* PhyML separada em parcela fixa e variável e apresentando amostras de preenchimento da parcela variável.

Parcela Fixa:

```
-i arquivo.phy -d nt -n 1 -b 0 --r_seed 12345
--no_memory_check -s BEST -o tlr
```

Parcela Variável:

--run_id GTR	-m 012345 -f m -c 1
--run_id GTR+G	-m 012345 -f m -c 4 -a e
--run_id GTR+I	-m 012345 -f m -v e -c 1
--run_id GTR+I+G	-m 012345 -f m -v e -c 4 -a e
--run_id F81	-m 000000 -f m -c 1
--run_id 001000	-m 001000 -f 0.25,0.25,0.25,0.25 -c 1
--run_id 010200+I+G+F	-m 010200 -f m -v e -c 4 -a e
--run_id TIM1ef+I	-m 012230 -f 0.25,0.25,0.25,0.25 -v e -c 1
--run_id TVM+G	-m 012314 -f m -c 4 -a e
--run_id SYM	-m 012345 -f 0.25,0.25,0.25,0.25 -c 1
--run_id SYM+G	-m 012345 -f 0.25,0.25,0.25,0.25 -c 4 -a e
--run_id SYM+I	-m 012345 -f 0.25,0.25,0.25,0.25 -v e -c 1
--run_id SYM+I+G	-m 012345 -f 0.25,0.25,0.25,0.25 -v e -c 4 -a e

Fonte: Elaborado pelo autor.

resultados, uma vez que o projeto atua basicamente como um *driver* da aplicação PhyML de Guindon et al. (2010). Cada teste de adequação é mapeado para uma nova execução do PhyML, com os devidos parâmetros ajustados, embora mantendo uma estrutura base. Sendo assim, é possível visualizar o jModelTest como um *front-end* que dispara e gerencia múltiplas execuções paralelas do PhyML, recolhendo e compilando os resultados ao final delas. A Figura 21 apresenta a parcela fixa da linha de comando utilizada para controlar o comportamento do PhyML, assim como, exemplos da parcela variável que são, por fim, combinadas para gerar cada linha de comando que dispara a execução de um teste de adequação de sistemas de substituição molecular.

Em função da relação entre jModelTest e PhyML é possível basear a implementação do protótipo no *replay* dos comandos enviados ao PhyML, bastando para isso a coleta de *traces* de execução. Estes registros foram habilitados pelo autor por meio de uma versão levemente modificada do jModelTest, e que permitiu garantir o mesmo comportamento no que diz respeito à característica das execuções, haja vista que o cálculo de adequação, realizado pelo PhyML, é a parte mais custosa da execução do jModelTest. Os arquivos de trace encontram-se disponíveis no repositório³ que armazena o código fonte dos *scripts* de execução.

Uma das tarefas realizadas pelo jModelTest, para viabilizar a comunicação com o PhyML, é a conversão nos tipos de arquivo fornecidos como parâmetros de entrada. Uma vez que o protótipo também funcionará como um controlador do PhyML, existe a necessidade de converter os tipos

³ Disponível no endereço: <https://github.com/mateusaubin/modeltest-loadexerciser/tree/master/traces>

de dados dos arquivos utilizados no *dataset* de avaliação do modelo. Para tanto, foi utilizada a biblioteca ALTER, especializada na conversão entre tipos de alinhamentos e também utilizada pelo jModelTest, de modo que todos os arquivos presentes no *dataset* estejam em formato .phy, que é o único reconhecido pelo PhyML.

Visando obter uma maior qualidade e reprodutibilidade das execuções realizadas, a aplicação contou com automação, por meio de scripts de execução automatizada dos cenários, tanto para estabelecer as bases de comparação no jModelTest⁴, quanto para gerir e executar as execuções do protótipo do modelo He-lastic⁵. Através da automação foi possível aumentar o número de execuções de cada cenário de avaliação e, consequentemente, a confiabilidade estatística dos resultados. Outro benefício da automação foi possibilitar a execução não assistida dos cenários, o que teria comprometido seriamente a capacidade de execução, uma vez que, múltiplos cenários ultrapassavam a marca de 20 horas de execução para todos os arquivos contidos no *dataset* de testes e estes, por sua vez, deveriam ainda ser repetidos ao menos cinco vezes.

5.3 Infraestrutura

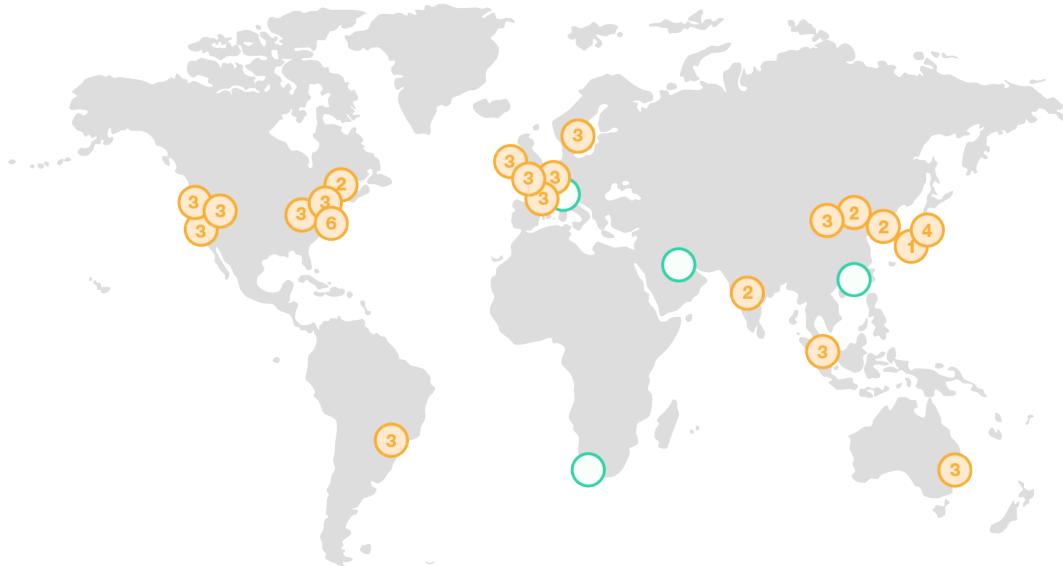
No que diz respeito a infraestrutura de computação em nuvem, a escolha foi pelo provedor Amazon Web Services (AWS), uma subsidiária da Amazon, que por sua vez, é considerada uma gigante do *e-commerce* nos Estados Unidos e já conta com alguma atuação no Brasil. A escolha da AWS se deu por este ser o maior dentro os grandes provedores de computação em nuvem (também chamados na imprensa de *hyperscalers*) e por experiência acadêmica e profissional do autor que usou e administrou nos quatro anos anteriores ambientes de nuvem baseados no provedor AWS, além de utilizá-lo como plataforma para o desenvolvimento do projeto Elastipipe, realizado como trabalho de conclusão na Graduação (AUBIN; RIGHI, 2015; RIGHI et al., 2016, 2017). A arquitetura do provedor é baseada em regiões, sendo estas geralmente divididas à nível continental onde cada região é, por sua vez, subdividida em pelo menos duas (ou geralmente três) zonas de disponibilidade, conforme mostra a Figura 22.

As zonas de disponibilidade servem como proteção contra desastres e mecanismo de redundância e alta disponibilidade, por representarem ambientes espelhados umas das outras, no que diz respeito à infraestrutura do provedor. Contudo, para tirar proveito dessas capacidades, geralmente é necessário que a aplicação seja preparada para saber lidar com esta possibilidade. Embora as regiões costumem ser parecidas, existe entre elas certa diferença no rol de serviços ofertados e, por vezes, até mesmo no hardware disponível, o que pode complicar a escolha. Um exemplo disso, foi a recente chegada do serviço *Redshift* na região situada em sa-east-1 (São Paulo, Brasil) no fim de 2016, sendo que sua oferta inicial na região us-east-1 (Virgínia do

⁴Disponível no endereço: <https://github.com/mateusaubin/modeltest-loadexerciser/blob/master/exec-old-userdata.sh>

⁵ Disponível nos endereços: <https://github.com/mateusaubin/modeltest-loadexerciser/blob/master/exec-new.py> & <https://github.com/mateusaubin/modeltest-loadexerciser/blob/master/exec-new-helper.sh>

Figura 22 – Mapa das regiões e respectivas zonas de disponibilidade fornecidas pelo provedor de computação em nuvem AWS, com os círculos verdes representando regiões em construção.



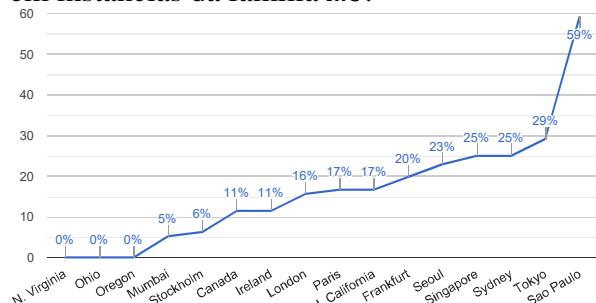
Fonte: Reproduzido de <https://aws.amazon.com/about-aws/global-infrastructure/>.

Norte, USA) ocorreu em 2012.

No contexto deste trabalho, a região escolhida foi a `us-east-2` localizada em Ohio, USA, com os principais motivadores sendo custo e funcionalidades, pois a região brasileira ainda não disponibiliza parte significativa⁶ do portfólio de serviços da AWS, além de cobrar um preço mais alto pelo uso dos recursos, resultando em um *premium* de pelo menos 50% sobre o valor base cobrado em Ohio, segundo levantamento de Marquez (2018), demonstrado nas Figuras 23 e 24. A política de preços é relevante na escolha da região, pois, embora esteja situada no Brasil, a

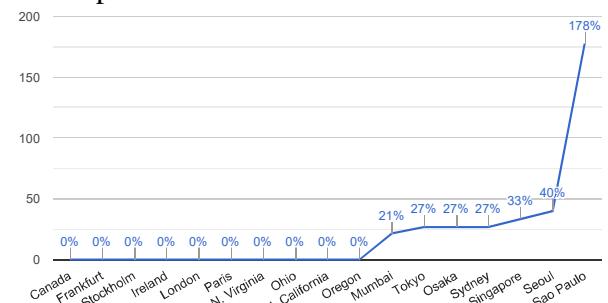
⁶ A página <https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/> apresenta a lista completa de serviços disponível por região e permite chegar a esta conclusão.

Figura 23 – Variação no custo por região do provedor AWS no uso de recursos computacionais em instâncias da família m5.



Fonte: Marquez (2018).

Figura 24 – Variação no custo por região do provedor AWS para a transferência de 1 TB de dados para a Internet.



Fonte: Marquez (2018).

cobrança dos custos oriundos desta região ainda se dá em Dólares Americanos (USD), assim como todas as demais regiões, o que nega qualquer benefício fiscal e/ou tecnológico advindo do seu uso (uma exceção seria o desenvolvimento de aplicações altamente sensíveis a latência, por exemplo).

O pioneirismo da AWS, no que diz respeito ao paradigma FaaS / *serverless*, também teve grande influência na escolha, haja vista, que foi este provedor quem efetivamente deu origem a este modelo computacional. Medições realizadas por [McGrath e Brenner \(2017\)](#) indicam que a AWS fornece as melhores métricas em termos de escalabilidade, latência de arranque frio (*coldstart*) e taxa de transferência. Além destes resultados, testes realizados por [Wang et al. \(2018\)](#) concluem que o serviço provido pela AWS é superior, uma vez que plataformas concorrentes como Azure, da Microsoft, e GCP, da Google, enfrentam dificuldades em provisionar um número suficiente de recursos subjacentes, capazes de servir requisições com alto paralelismo. Enquanto este comportamento é geralmente visto como vantagem, este resultado se dá em função da política de *bin-packing*, adotada pela AWS, que tenta manter o melhor aproveitamento de recursos servindo requisições FaaS a partir das mesmas VMs, até que não haja mais espaço, onde só então é provisionada uma nova VM. Contudo, [Wang et al. \(2018\)](#) previnem que esta política pode resultar em contenção, conforme as características no uso de recursos por parte das funções executadas, podendo prolongar artificialmente o tempo de execução.

Uma das vantagens para o provedor, no que diz respeito ao uso de FaaS (abordada na [Subseção 2.2.5](#)), está na possibilidade de manter um bom nível de utilização para hardware que esteja caindo em desuso. Havendo a possibilidade de manter um determinado patamar de performance entre diferentes VMs e famílias de instâncias, as limitações no consumo de recursos estabelecidas pelo paradigma FaaS permitem com que versões relativamente antigas de hardware sejam utilizadas sem desvantagens ao usuário, ao mesmo tempo que reduz a necessidade de compra para o provedor. Este comportamento pode ser observado na [Tabela 22](#) que apresenta a distribuição de frequência com que são utilizados determinados tipos de CPUs no AWS Lambda, a oferta de FaaS do provedor AWS, em comparação com o uso destas mesmas CPUs quando disponíveis no serviço de máquinas virtuais, o EC2. Para referência, famílias atuais do EC2 são aquelas que têm sufixo 5, como por exemplo, c5, r5 e m5, que são otimizadas para CPU,

Tabela 22 – Tipos de CPUs utilizadas no AWS Lambda e sua equivalência em termos de Famílias EC2.

Fração [†]	vCPU	Frequência	Denominação	Família
59,3 %	E5-2666	2,90 GHz	Haswell	c4
37,5 %	E5-2680	2,80 GHz	Ivy Bridge	c3
3,1 %	E5-2676	2,40 GHz	Haswell	m4

[†] Levantamento realizado através da coleta de estatística dos arquivos /proc/cpuinfo e /proc/meminfo de 20 mil VMs distintas.

Fonte: [Wang et al. \(2018\)](#) com complementos do autor.

memória e uso geral, respectivamente.

A ampla seleção de serviços disponibilizados pela nuvem da AWS faz com que o exercício e definição de arquitetura se assemelhe a montagem de LEGO⁷ onde é possível deixar a cargo do provedor diversas das funcionalidades fundamentais de um projeto *software*. Visando garantir o entendimento da arquitetura do protótipo descrita a seguir, uma lista com os serviços mais relevantes utilizados é apresentada abaixo, juntamente com uma breve descrição de suas características e funcionalidades.

- **Lambda:** é a implementação da AWS para o paradigma FaaS;
- **Batch:** serviço para execução de tarefas em modo *batch* de processamento, provendo mecanismos de filas, prioridades e ambientes de computação, além de gerenciar automaticamente ações de elasticidade conforme a demanda;
- **S3:** provê armazenamento de objetos/arquivos com alta capacidade de requisições e *throughput*, enquanto fornece garantias de replicação e durabilidade de 11 níveis;
- **DynamoDB:** banco de dados não relacional (NoSQL) orientado a tuplas chave–valor com latência na casa de um dígito de segundo ($0 \sim 9s$);
- **SNS:** filas de mensagens no modelo *publish–subscribe* suportando alta vazão e durabilidade por meio de servidores distribuídos, fornecendo política de entrega ‘pelo menos uma vez’(*at least once delivery*);
- **ECS:** é a implementação da AWS para o paradigma de Orquestração de Contêineres, sendo utilizado como recurso subjacente pelo Batch, para viabilizar a execução de qualquer tipo de aplicação em um ambiente isolado e com todas as dependências embutidas por meio de contêineres Docker;
- **Auto-Scaling:** mecanismo de elasticidade fornecido pela AWS, que possibilita a definição de regras para elasticidade horizontal automática e reativa por replicação de VMs (Figura 4), sendo utilizado pelo ECS no controle da elasticidade;
- **EC2:** base para a grande maioria dos demais serviços providos pela AWS, fornecendo, também para usuários finais, Máquinas Virtuais configuráveis em uma ampla variedade de dimensões como processamento, memória, endereços de IP e placas de rede, armazenamento e imagens de sistemas operacionais;
- **CloudWatch:** coleta, gestão e relatórios de métricas operacionais geradas pelos demais serviços AWS e pelo próprio usuário;

⁷ Brinquedo infantil caracterizado por suas peças que se encaixam e permitem montar uma infinidade de objetos.

- **CloudFormation:** gerencia a montagem, atualização e remoção de ambientes computacionais definidos por meio de uma linguagem de programação, aplicando o conceito conhecido como *infrastructure as code*;
- **IAM:** gestão e permissionamento de usuários, grupos e papéis.

5.4 Protótipo

Desenvolvido na linguagem de programação Python, o protótipo do modelo He-lastic tira proveito do fato de que o jModelTest atua como *front-end* para o PhyML e adota uma estratégia baseada na reexecução de *traces* coletados a partir de uma versão do jModelTest modificada para gerar tais registros. Conforme mencionado anteriormente, o uso do PhyML como efetivo executor dos cálculos necessários para o teste de adequação de sistemas de substituição molecular é a tarefa mais custosa computacionalmente, de forma que, o uso de *traces* não foi considerado uma vantagem para o protótipo, por não tomar, no jModelTest, tempo suficiente de cálculo, para ser significativo na avaliação de tempos de execução. Além disso, o uso do PhyML provê um segundo benefício, ao garantir que os cálculos executados pelo protótipo são exatamente os mesmos do que aqueles desempenhados pelo jModelTest, validando os resultados e assegurando que as diferenças nos custos e tempos de execução se originam, exclusivamente, das diferenças entre o modelo He-lastic (e seus modos de execução) e do jModelTest.

Como pode ser visto no trecho de código do arquivo `modeltest.py`, apresentado na Figura 25, o fluxo de execução segue um formato *decode–fetch–execute–update*. Inicialmente o comando recebido é inspecionado para garantir sua validade, para que então sejam obtidos os dados de entrada, neste caso um arquivo de múltiplos alinhamentos de sequências moleculares, o que permite executar o PhyML através do comando `subprocess.run()`, na linha 22. Por fim há um bloco dedicado ao tratamento de erros e, caso o processamento conclua com sucesso, submeter os resultados e definir o processamento como completo. Conforme anteriormente mencionado, repositórios de código–fonte contendo todos os artefatos desenvolvidos estão livremente disponíveis nos links da Tabela 17, onde o repositório `modeltest-lambda` contém o código executado nos cenários, e o repositório `modeltest-loadexerciser` contém `scripts` que gerenciam a execução dos cenários de avaliação.

5.4.1 Implementação

Uma das diretrivas que guiou o desenvolvimento foi a adoção do maior número possível de serviços de computação em nuvem, onde fosse possível e apropriado, resultando em um desenvolvimento efetivamente *cloud-native*, buscando maximizar a performance e a obter tolerância a falhas necessária para aplicações distribuídas e de alto desempenho. Esta decisão está alinhada com as decisões de projeto detalhadas na Seção 4.2 e tem como subproduto uma simplificação na configuração, assim como, a consequente redução na carga operacional do administrador

Figura 25 – Trecho de código do arquivo `modeltest.py` responsável por coordenar a execução do cálculo de adequação de sistemas de substituição molecular através da biblioteca PhyML, emulando o comportamento do jModelTest por meio de *traces* de execução.

```

1  # parse
2  sns_result = aws.SNS(record['Sns'])
3
4  # download
5  s3_result = aws.S3Download(sns_result.file_info, sns_result.jmodel_runid)
6
7  cmdline_args = [os.path.join(os.getcwd(), 'lib', 'phyml'), ]
8  cmdline_args.extend(['-i', s3_result.local_file])
9  cmdline_args.extend(sns_result.payload.split())
10
11 trace_file = os.path.join(
12     s3_result.tmp_folder,
13     "{}_phyml_trace_{}.txt".format(
14         sns_result.jmodel_runid,
15         sns_result.jmodel_modelname)
16 )
17
18 logging.info("PhyML starting...")
19 phyml_start = timer()
20
21 with open(trace_file, "w") as file:
22     result = subprocess.run(cmdline_args,
23                             stdout=file,
24                             stderr=subprocess.STDOUT)
25
26 phyml_duration = (timer() - phyml_start)
27 logging.warn("PhyML took {} secs".format(phyml_duration))
28
29 # bail out if phyml error'd
30 if result.returncode != 0:
31
32     logging.critical("PhyML.ReturnCode={}".format(result.returncode))
33
34     # log trace file
35     with open(trace_file, 'r', encoding='UTF-8') as file_stream:
36         file_contents = file_stream.read()
37         logging.error(file_contents)
38
39     raise subprocess.SubprocessError("Error calling PhyML")
40
41 # phyml succeeded, go ahead
42 result_files = [x for x in
43                 os.listdir(s3_result.tmp_folder)
44                 if x != sns_result.jmodel_runid]
45
46 logging.debug("Phyml produced = {}".format(result_files))
47
48 # upload
49 s3_up = aws.S3Upload(s3_result.tmp_folder, result_files, sns_result)
50
51 logging.info("Uploaded = {} to {}://{}//{}".format(
52     list(s3_up.files.values()),
53     sns_result.file_info['bucket'],
54     s3_up.jmodel_runid)
55 )
56
57 # mark as done
58 aws.DynamoDB(sns_result.jmodel_runid, sns_result.jmodel_modelname)

```

Fonte: Elaborado pelo autor.

de ambientes. No que diz respeito a portabilidade entre provedores de computação em nuvem, todos os serviços fundamentais encontram concorrentes com funcionalidades análogas, enquanto serviços secundários, nomeadamente CloudWatch, CloudFormation e IAM, também têm concorrentes, contudo, sua adaptação requer intervenção manual devido às diferenças e características de cada plataforma de computação em nuvem.

Um dos destaques do ponto de vista da implementação do protótipo do modelo He-lastic, e que pode ser observado nos repositórios públicos de código fonte listados na Tabela 17, foi o alto índice de reuso de código, uma propriedade que emergiu em função da utilização das duas camadas de elasticidade baseadas em FaaS e Contêineres, onde o código utilizado como função na camada FaaS foi inteiramente compartilhado com a camada de Contêineres, na forma de uma biblioteca de código fonte. Embora a camada de Contêineres não faça uso ou refencie da camada FaaS, o código fonte compartilha exatamente o mesmo artefato que foi enviado para a camada FaaS, necessitando apenas de um *wrapper* capaz de traduzir as estruturas de dados para que fiquem no formato esperado pela chamada da função. Os módulos desenvolvidos no protótipo, assim como, seus artefatos de código podem ser vistos na Tabela 23, onde também é possível visualizar o alto nível de reuso de código por meio do uso do PhyML, da biblioteca de funções `aws.py` e do executável `modeltest.py`.

Embora fossem esperados apenas dois programas, sendo um para cada camada do modelo, houve a necessidade durante o desenvolvimento de produzir três programas, para que o protótipo pudesse funcionar conforme especificado na arquitetura do modelo, seguindo as definições da Seção 4.3. A diferença deu-se pela necessidade de introduzir um módulo responsável por traduzir as requisições originalmente enviadas para a camada FaaS (Lambda), em requisições que seguissem o formato especificado para a camada de Contêineres (Batch), originando o módulo *forwarder*. Este módulo tem seu objetivo estritamente definido pelo encaminhamento de requisições, não realizando qualquer tipo de cálculo ou computação útil do ponto de vista do teste de adequação de sistemas de substituição molecular, contudo, cumprindo um papel necessário

Tabela 23 – Artefatos de código presentes nos três módulos do protótipo, com o reuso destacado em tons de azul.

FaaS		Orquestrador de Contêineres
forwarder	modeltest	
src/aws.py	src/aws.py	src/aws.py
src/forwarder.py	lib/phym	lib/phym
	src/modeltest.py	src/modeltest.py
		src/dockerentrypoint.py
		requirements.txt

Fonte: Elaborado pelo autor.

para o funcionamento do protótipo. Em contrapartida, realizam processamentos úteis, do ponto de vista funcional, os módulos *modeltest* na camada FaaS, e o respectivo módulo da camada Orquestradora de Contêineres que, conforme mencionado, reutiliza totalmente código do módulo anterior.

5.4.2 Ferramentas

Nesta subseção são brevemente descritos dois projetos de *software* que tiveram um papel de grande relevância no desenvolvimento do protótipo e avaliação dos cenários. Embora não sejam bibliotecas ou *frameworks*, que por característica são utilizados diretamente no código da aplicação, tais ferramentas auxiliam na execução e automação de diversas tarefas relacionadas aos processos operacionais de execução dos cenários de avaliação. Desta forma, apesar de não serem componentes obrigatórios para o modelo, cabe mencionar os projetos Docker e *serverless framework* pelas contribuições no decorrer deste trabalho, detalhadas a seguir.

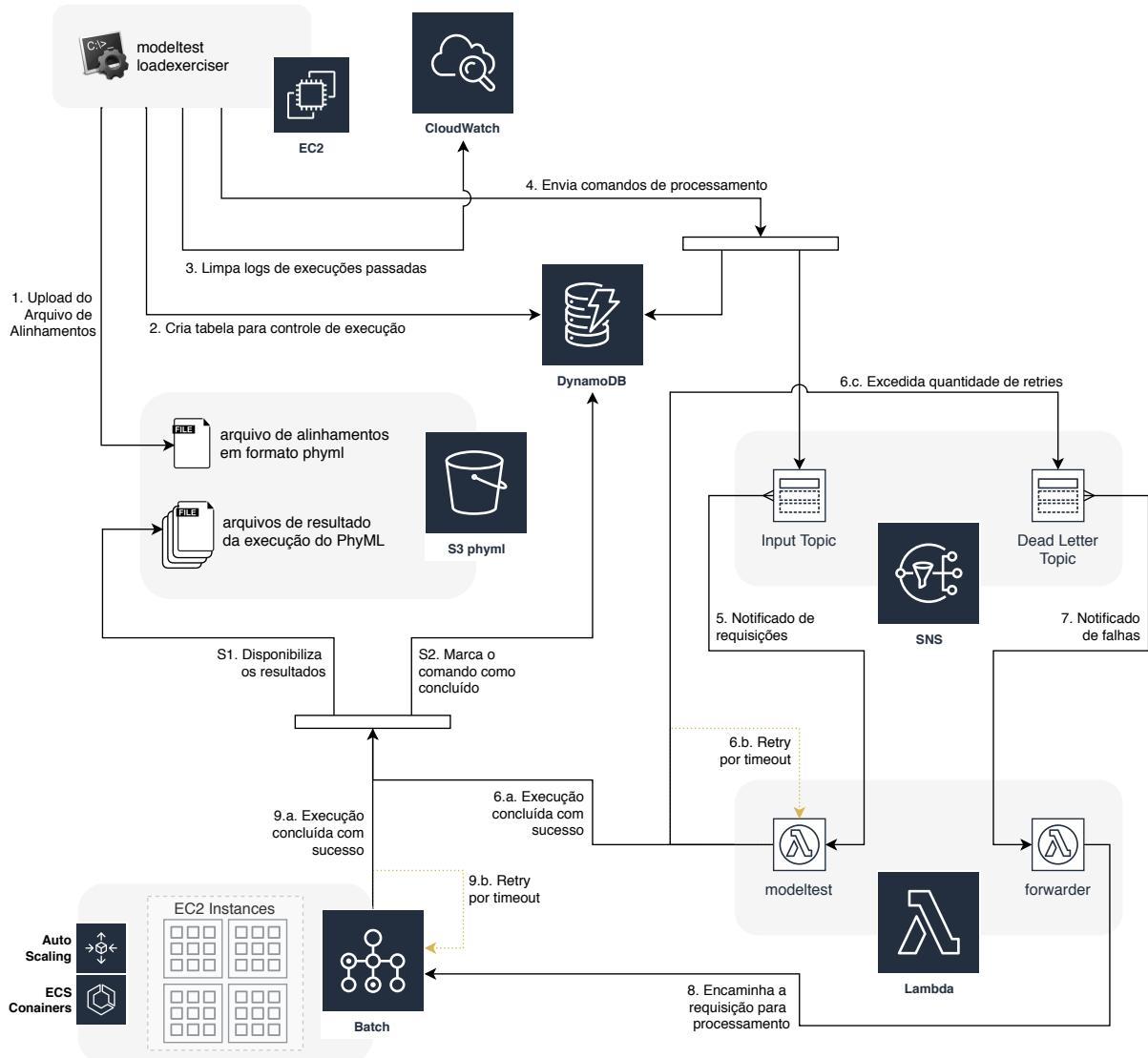
Dentre o ferramental adotado para o desenvolvimento do protótipo, o projeto Docker contribuiu de maneira significativa para garantir a testabilidade do protótipo ao permitir a especificação de um ambiente isolado e bem definido contendo os artefatos de código, *runtimes* e dependências empacotadas em uma única imagem, que pode ser utilizada no ambiente de Orquestração de Contêineres (Batch) e localmente. Através de um arquivo chamado *dockerfile* é possível especificar todos os passos necessários para a construção do contêiner e obter, depois de compilado, uma imagem pronta para implantação em qualquer ambiente que suporte imagens Docker. Além disso, por meio de projetos como o *docker-lambda*⁸ é possível obter localmente um ambiente que reproduz as características da camada FaaS (Lambda), o que também contribuiu para a testabilidade e o desenvolvimento do protótipo.

O projeto *serverless framework*⁹, que fornece uma visão unificada a respeito das capacidades de FaaS disponíveis nos principais provedores de computação em nuvem, também teve papel fundamental no desenvolvimento do protótipo e possibilitou mantê-lo agnóstico de provedores. Através do arquivo *serverless.yml* foi possível especificar todo o ambiente computacional necessário para a execução do protótipo, contemplando as duas funções FaaS e suas configurações, assim como o Orquestrador de Contêineres e suas respectivas configurações. Esta capacidade está intimamente ligada ao fato de que o *serverless framework* permite ao usuário embutir comandos CloudFormation no processo de implantação (*deployment*) do projeto, de maneira que o arquivo *serverless.yml* define tudo o que diz respeito à criação, atualização e remoção do ambiente requerido pelo protótipo do modelo He–lastic, permitindo com que simples comandos como *s1s deploy* e *s1s remove* montem e desmontem todos os componentes necessários para a execução, assim como suas dependências.

⁸ Disponível em <https://github.com/lambci/docker-lambda>.

⁹ Disponível em <https://serverless.com/framework>.

Figura 26 – Diagrama de arquitetura da implementação do protótipo do modelo He-lastic descrevendo as principais etapas no fluxo de execução.



Fonte: Elaborado pelo autor.

5.4.3 Arquitetura do Protótipo

A arquitetura do projeto desenvolvido para o protótipo do modelo He-lastic pode ser vista na Figura 26 que, além dos componentes e suas relações, apresenta também um fluxo de execução para referência. Cabe ressaltar, que o papel das duas camadas de elasticidade está dividido entre os componentes Batch, que assume as funções referentes à camada por Orquestração de Contêineres, e o componente Lambda, mais precisamente através da função `modeltest` que assume responsabilidade da camada FaaS. Embora não seja evidente na figura, é importante reforçar que a camada FaaS é caracterizada por sua capacidade de escalar rapidamente para atender centenas de requisições, sendo esta propriedade especialmente relevante para a implementação do `modeltest`.

A ausência de comunicação lateral entre tarefas da aplicação jModelTest e, consequentemente do protótipo, favorece o cenário implementado ao adotar a técnica de comunicação indireta por meio de armazenamento, neste caso o S3, de maneira muito similar a estratégia adotada por [Shankar et al. \(2018\)](#). Os pilares da implementação estão na idempotência, ou seja, a capacidade de aplicar várias vezes a mesma computação e obter sempre o mesmo resultado, independentemente de resultados anteriores e sem efeitos colaterais, e na definição de que uma tarefa só será removida do armazenamento de estado, papel desempenhado pelo DynamoDB, depois que sua computação estiver realizada e os resultados disponibilizados. Através destas estratégias, o protótipo não precisa de garantias firmes quanto as capacidades de outros elementos da arquitetura, como por exemplo, as filas de mensagens implementadas pelo SNS, que oferecem apenas garantias *at least once* (pelo menos uma vez), ao invés da garantia mais complexa de *exactly once* (exatamente uma vez), favorecendo a portabilidade e aumentando a confiabilidade do modelo e, por consequência, do protótipo.

Também se tornam aparentes, em função das escolhas arquiteturais, as propriedades de tolerância a falha e recuperação presentes no modelo He–lastic. Assim como em [Jonas et al. \(2017\)](#), a garantia de operações atômicas fornecida pelo armazenamento de estado (DynamoDB) é suficiente para garantir o sucesso de uma execução, embora, para um nível extra de garantias, este desfecho possa ser validado através da checagem dos arquivos de resultado presentes no armazenamento do S3. Desta forma, não há necessidade de gerenciar ou recuperar tarefas que podem ter sido abandonadas durante o processamento, uma vez que é seguro consultar diretamente o armazenamento de estado e de objetos para determinar todas as tarefas não concluídas, reduzindo a complexidade da implementação de tolerância a falhas do modelo He–lastic a um problema de recuperação de tarefas abortadas. Contudo, devido a sua arquitetura em duas camadas, a recuperação pode acontecer por meio de três mecanismos distintos: *a) retry* de tarefas na camada FaaS; *b) retry* de tarefas na camada de Orquestração de Contêineres; e *c) encaminhamento de tarefas falhadas da camada FaaS para a camada de Orquestração de Contêineres*. Sendo esta última estratégia um dos diferenciais do modelo e do protótipo aqui descritos.

As decisões de arquitetura, tanto do modelo quanto do protótipo, favorecem o comportamento independentemente escalável e elástico dos seus componentes, favorecendo não somente a execução de alta performance, como a progressividade nos custos associados a execução de aplicações baseadas no modelo proposto. Esta característica é particularmente relevante em contextos de baixos recursos financeiros ou de pequena escala, que não podem se dar ao luxo de gastar mais do que os já apertados orçamentos. O modelo He–lastic também pode ser uma alternativa viável para instituições maiores e que possuem recursos computacionais. Contudo, conforme afirma [Shankar et al. \(2018\)](#), tais recursos tradicionalmente se mostram de difícil acesso devido às longas filas de esperas por alocação de tempo de computação ou encontram-se impedidos por grandes volumes de processos e burocracias, desencorajando seu uso.

5.4.4 Desafios e Dificuldades

Dentre as dificuldades encontradas no desenvolvimento do protótipo, duas delas tem impacto significativo nos resultados e devem ser abordadas. A primeira dificuldade se origina do fato de que, ao contrário de outros projetos científicos e de boa parte dos *benchmarks* sintéticos não há um controle explícito sobre o grão de paralelismo que será computado. Esta ausência de uma unidade de medida para o nível de granularidade exigiu a formalização da [Equação 4.3](#), como uma maneira de determinar a granularidade e, assim, incluir no *dataset* de testes, arquivos capazes de representar níveis variados de granularidade computacional, conforme mostra a [Tabela 18](#).

A ausência de controle sobre o número de *retries* executada nas funções Lambda, da camada FaaS, se manifestou como a segunda dificuldade relevante para este trabalho. Visando aumentar a confiabilidade das operações e sendo uma decisão justificável, quando adotado o ponto de vista original a respeito das funções lambda, que objetivam execuções de curta duração orientadas a eventos, o estabelecimento por parte da AWS de um número fixo de 3 *retries* combinado com espera por *backoff* exponencial, tornou-se um problema para o protótipo do modelo He–lastic. Devido à proporcionalidade entre o *timeout* da função e o tempo de *backoff*, a espera em caso de erro injeta um *delay* significativo no processamento, sendo especialmente nocivo em cenários de *timeout* longo e onde parte significativa dos cálculos de adequação de sistemas de evolução não conseguem ser absorvidos pela camada FaaS. O resultado se manifesta como um tempo de espera, onde não é realizada nenhuma computação útil nos cenários em que justamente haveria maior necessidade de agilidade, devido ao alto número de requisições que a camada FaaS (Lambda) não consegue atender e que entrarão na fila para processamento, via elasticidade reativa na camada de Orquestração de Contêineres (Batch).

Devido a ausência de parâmetros para configurar o comportamento de *retry* do Lambda, iniciaram-se esforços em busca de uma solução para contornar este cenário, no entanto, por ser intimamente ligado ao serviço, não foi possível encontrar uma solução capaz de burlar completamente o comportamento de *retry* e o impacto no tempo de execução e no custo que traz consigo. Como estratégia para redução dos tempos de espera foi embutida na função *forwarder* da camada FaaS, a capacidade de disparar ações preparatórias de elasticidade no serviço Batch, representante da camada de Orquestração de Contêineres. Esta forma de heurística usa dados a respeito da fila de execuções do Batch para determinar um número apropriado de vCPUs para requisitar, de forma a evitar a espera pelo *ramp-up* orgânico das ações de elasticidade sem, no entanto, requisitar um número excessivo de recursos, que podem ser rapidamente descartados, o que origina o comportamento de *thrashing*, enquanto respeita os limites superior e inferior, quanto ao número de recursos disponíveis. Desta forma, cada vez que uma nova tarefa que falhou na camada FaaS chega para encaminhamento para a camada de Orquestração de Contêineres, a função *forwarder* avalia a quantidade de tarefas pendentes no Batch (Orquestrador de Contêineres) e, no caso de uma variação significativa entre tarefas

pendentes e recursos disponíveis, solicita uma readequação. Esta ação permite burlar o tempo necessário para que as primeiras ações de elasticidade entrem em efeito e comecem a aumentar a quantidade de recursos até o equilíbrio, podendo ser encarada (com a devida licença poética) como ‘pré-aquecimento do forno’, contudo, sem desperdício, uma vez que a ação só é realizada quando há de fato tarefas sendo encaminhadas para o Orquestrador de Contêineres.

Uma evolução desta heurística pode se basear na existência de recursos ativos na camada do Orquestrador de Contêineres e na probabilidade de uma tarefa exceder o tempo limite na camada FaaS, para realizar o *pass-through*, ou seja, encaminhar esta requisição diretamente para o Batch, ao invés de permitir seu processamento pelo Lambda. Uma forma simplificada desta heurística poderia tomar como entrada uma taxa de requisições com falha sobre o total, definindo que, caso a taxa de falha ultrapasse um determinado *threshold*, as demais tarefas serão encaminhadas diretamente para a próxima camada. Embora mais simples, esta versão da heurística corre o risco de obter uma alta taxa de ineficiência quando confrontada com tarefas heterogêneas ou com ampla variação na complexidade, como é o caso da aplicação adotada neste trabalho. Estas estratégias, contudo, poderiam mitigar os problemas com a espera ociosa em função do *retry* de tarefas e impactariam positivamente no custo, evitando repetir três vezes na camada FaaS, uma computação que já está fadada ao fracasso. Entretanto, devido à dificuldade em determinar a probabilidade de uma tarefa exceder o tempo limite na camada FaaS, ou as possíveis deficiências da versão simplificada, estas heurísticas não foram implementadas no protótipo do modelo He-lastic, sendo mantidas como indicação para trabalhos futuros.

5.4.5 Parâmetros

Em função das decisões de projeto e arquitetura do protótipo do modelo He-lastic, uma simplificação dos parâmetros do modelo foi possível. Conforme abordado na Tabela 13, o modelo He-lastic contempla em seu projeto cinco parâmetros que determinam o comportamento das camadas de elasticidade, sendo, dois deles, a respeito do FaaS e, os demais, sobre a camada de Orquestração de Contêineres. A diferença se dá em função do uso do serviço Batch para gerenciar a camada de Orquestração de Contêineres, onde os *thresholds*, ou seja, os limites que controlam a agressividade do comportamento das ações de elasticidade, são gerenciados de

Figura 27 – Parâmetros de configuração do protótipo conforme especificados no arquivo serverless.yml e suas equivalências sobre as definições da Tabela 13.

```
custom:
  runner: "ap6"          // Opcional - Tag para relatórios de análise de custos
  batch:
    cluster-cpus: 36    // Orquestrador de Contêineres - Número Máximo de CPUs
  lambda:
    power: 1536         // FaaS - Potência
    timeout: 60          // FaaS - Tempo Limite
```

Fonte: Elaborado pelo autor.

maneira automática e com base na quantidade de itens em espera.

Esta diferença permitiu reduzir de cinco para apenas três os parâmetros do modelo e não foi considerada relevante do ponto de vista de avaliação e testes do modelo He–lastic, haja vista que os cenários de avaliação delineados na [Subseção 5.1.3](#) não previam variações nos *thresholds* de elasticidade. Embora os parâmetros sejam significativos para uma otimização fina em cenários produtivos em que o modelo seja implementado, o principal objetivo da atual avaliação está em determinar o impacto da estratégia e a interação entre as duas camadas de elasticidade. Desta forma, na relação entre conveniência e controle apresentada pelo uso do AWS Batch contra a gestão manual da camada de Orquestração de Contêineres, por meio dos *thresholds* de elasticidade, foi favorecido o uso do serviço Batch devido à significativa redução no esforço de desenvolvimento e impacto negligenciável na estratégia de avaliação.

Em função da adoção da estratégia de *infrastructure as code*, habilitada através do *serverless framework* e o serviço CloudFormation é possível exercer controle sobre todos os parâmetros do protótipo a partir de um ponto central de configuração presente no arquivo `serverless.yml`. A [Figura 27](#) apresenta uma amostra do arquivo de configurações, detalhando a definição dos parâmetros de configuração. Desta forma, toda a criação e remoção do ambiente necessário para execução dos cenários de avaliação pode ser realizada com apenas dois comandos fornecidos pelo *serverless framework*, garantindo a reproduzibilidade dos testes e a ausência de artefatos oriundos de execuções anteriores e que poderiam influenciar nos resultados.

5.5 Considerações Parciais

O presente capítulo detalhou os tópicos relacionados a avaliação do modelo proposto, contemplando desde características das etapas e cenários de avaliação, assim como particularidades da aplicação, da infraestrutura de computação em nuvem e detalhes de implementação do protótipo. A respeito do protótipo chamam a atenção características como o amplo uso de serviços de computação em nuvem e a relação entre o Orquestrador de Contêineres e o componente FaaS, o que permitiu amplo reaproveitamento de código. O uso de ferramentas como *docker* e o *serverless framework* permitiram que o protótipo adotasse uma abordagem de infraestrutura como código, onde toda a criação do ambiente necessário para execução é descrita em código–fonte, facilitando a criação e remoção de ambientes e, consequentemente, melhorando a reproduzibilidade dos cenários executados. A arquitetura do protótipo se manteve próxima daquela especificada na proposta do modelo com a notável diferença que o ambiente local, e por consequência a interação com o *software jModelTest*, não foi implementado.

Foram descritas, também, as principais características da infraestrutura de computação em nuvem, abordando questões como serviços utilizados, o conceito de zonas de disponibilidade e especificação empregados pelo provedor AWS. No que diz respeito a aplicação utilizada como referência para o desenvolvimento, o *jModelTest*, foi abordada, neste capítulo, a íntima relação entre o programa *jModelTest* e o *PhyML*, o que permitiu ao protótipo explorar características

desta relação para implementar uma abordagem que, ao mesmo tempo que reduz o esforço de desenvolvimento, mantém as mesmas garantias quanto a completude e confiabilidade dos testes realizados, assim como seus resultados. Também foram especificados arquivos contidos no *dataset* de testes assim como os parâmetros de execução e os cenários de avaliação, tanto para o jModelTest, quanto para o modelo proposto, que determinam as características dos cenários de avaliação e serão os insumos das etapas seguintes de limpeza e análise dos dados.

6 RESULTADOS

“The purpose of life is not to be happy — but to matter, to be productive, to be useful, to have it make some difference that you have lived at all.”

Leo Rosten

Neste capítulo são apresentados os resultados obtidos por meio da execução da metodologia de avaliação sobre o *software* jModelTest e o protótipo do modelo He–lastic. Dentre os objetivos deste capítulo é possível citar a compreensão da proporcionalidade de custos entre as camadas de elasticidade do modelo, a caracterização do comportamento do projeto jModelTest, assim como o estudo das características do modelo proposto, seja na sua execução com apenas a camada FaaS, ou somente Orquestrador de Contêineres e, por fim, de ambas as camadas em conjunto.

Como etapa inicial rumo a caracterização dos resultados, uma análise de custo–benefício é realizada utilizando dados artificiais para exercitar diferentes conjuntos de parâmetros possíveis para as camadas de elasticidade do modelo He–lastic. Através destes dados será possível determinar se características como a associatividade de custos se aplica às camadas de elasticidade, assim como prever o impacto nos custos das mudanças nos parâmetros de execução. Esta análise, ainda que realizada com dados simulados, se mostra muito útil para realçar as similaridades e diferenças do modelo proposto frente a outras técnicas computacionais, salientando as propriedades únicas oriundas da combinação entre camadas de elasticidade.

No que tange aos resultados referentes à execução dos cenários de avaliação do jModelTest o objetivo é caracterizar, da maneira mais fiel possível, seu comportamento de escalabilidade assim como os tempos de execução. Através da análise de registros de utilização de CPU, assim como os tempos de execução dos cenários, será possível verificar o grau de eficiência do programa e avaliar as decisões tomadas durante seu desenvolvimento.

Em seguida poderão ser analisados os resultados obtidos através da execução dos cenários de testes do modelo He–lastic, onde poderão ser avaliados de maneira isolada, ou já traçando paralelos com os resultados do jModelTest, obtidos anteriormente. Neste caso a avaliação será dividida em três etapas correspondentes às configurações das camadas de elasticidade, sendo elas: somente FaaS, somente Orquestrador de Contêineres, e camadas combinadas. É de especial interesse uma comparação entre as características da execução com ambas as camadas versus cada camada individualmente, pois assim será possível determinar quais conjuntos de características o modelo proposto herdou de cada camada, sejam elas características positivas ou negativas para as métricas de tempo de execução e custo financeiro.

Por fim serão contrastados os cenários de avaliação especificamente no que diz respeito aos custos financeiros, uma análise que visa determinar qual das abordagens é mais eficaz na utilização de recursos. Também é de grande interesse verificar o impacto que a transição entre camadas das tarefas em execução tem no custo financeiro associado ao cenário de teste. O uso

deste tipo de avaliação permite estabelecer uma visão consolidada a respeito das características de performance e custo financeiro, que permite aos usuários do modelo avaliar a viabilidade do uso conforme seus próprios contextos, seja para utilizar o modelo He–lastic ou mesmo manter a utilização do jModelTest.

6.1 Estudo de Custo–Benefício

A estratégia de dupla camada de elasticidade adotada pelo modelo He–lastic, apoiada na divisão entre tarefas de curta duração (processadas pela camada FaaS através de alto nível de paralelismo) e tarefas de longa duração (processadas pelo Orquestrador de Contêineres), introduz a necessidade de que se estabeleça uma linha de corte que define a fronteira entre ambas camadas. Por meio de uma avaliação preliminar com dados sintéticos é possível verificar a relação entre as camadas de elasticidade, compreendendo em maior profundidade as implicações associadas a cada um dos parâmetros de configuração exigidos pelas camadas FaaS e de Orquestração de Contêineres. Sendo assim, nas próximas subseções são apresentadas as características de especificação de cada uma das camadas de elasticidade, assim como do seu uso misto.

6.1.1 Camada FaaS

Com vistas a um melhor entendimento do funcionamento da camada de curta duração, implementada através de FaaS, e por intermédio das definições prévias a respeito das métricas de avaliação do modelo He–lastic é apresentado, na Tabela 24, um estudo a respeito das escolhas de parâmetros da camada adotando o provedor AWS como referência. Através dos resultados apresentados neste estudo, é possível notar que a propriedade de associatividade de custos se mantém ao analisar as duplas em destaque referentes aos cenários {3, 4}, {5, 7} e {10, 11}, evidenciando a relação entre o tempo efetivo de execução, o tempo limite de execução e a medida de potência, definida, no estudo da Tabela 24, como a alocação de memória, conforme adotado pela AWS.

Embora o modelo de execução baseado em FaaS evite custos ociosos na ausência de requisições, obsoletando a necessidade de recursos que podem permanecer ociosos por longos períodos de tempo aguardando requisições (com *web-servers* e servidores de banco de dados com baixa utilização figurando como exemplos evidentes), ele, em contrapartida, penaliza operações que ficam em espera como requisições remotas, conexão com bancos de dados ou qualquer operação que demande alto volume de I/O ou tráfego de rede.

Desta forma, lançar mão de FaaS para atuar como *load-balancer*, por exemplo, embora seja atualmente uma tarefa que exige poucos recursos computacionais, se mostra uma péssima ideia ao exacerbar alguns dos pontos negativos do modelo de execução, nomeadamente: *a)* espera pelo resultado de terceiros, uma vez que o *load-balancer* apenas encaminha e aguarda resposta de um recurso de *back-end*; *b)* alta comunicação de rede, uma vez que o papel de um *load-balancer*

Tabela 24 – Análise do comportamento das métricas Custo e Financeiro aplicados no modelo de computação FaaS conforme parâmetros requeridos pelo modelo Heuristic onde as cores identificam linhas com o mesmo custo total apesar de parâmetros distintos.

#	Memória	CPU [†]	Tempo Efetivo	Tempo Máximo	y	t	$Custo$	$Financeiro$
1	128	8%		00:11,250	2	113	225	0,000023
2	256	17%		00:22,500	4	225	900	0,000094
3	384	25%		00:45,000	6	450	2700	0,000281
4	768	50%	00:22,500	01:30,000	12	225	2700	0,000281
5	768	50%		03:00,000	12	1800	21600	0,002251
6	1536	100%		01:00,000	24	600	14400	0,001500
7	1536	100%		01:30,000	24	900	21600	0,002251
8	1536	100%		03:00,000	24	1800	43200	0,004501
9	1920	125%		06:00,000	30	3600	108000	0,011253
10	2560	170%		12:00,000	40	7200	288000	0,030007
11	3008	200%	10:12,766	15:00,000	47	6128	288000	0,030007
12	3008	200%		15:00,000	47	9000	423000	0,044073

[†] Se refere à alocação proporcional de CPU onde 200% equivale à completa alocação de duas vCPUs

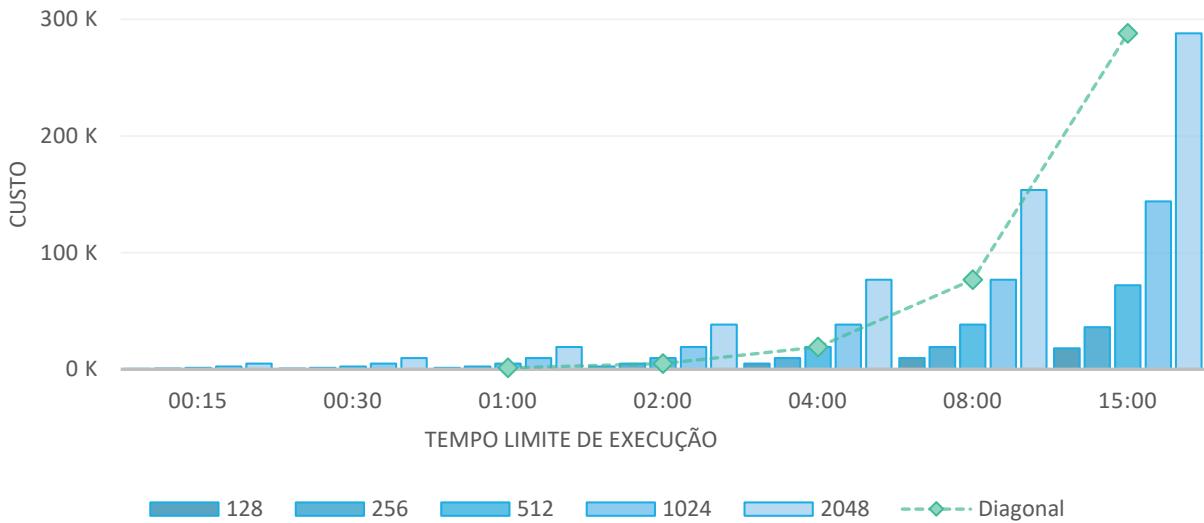
* Para as demais variáveis das equações de Custo (4.9) e Financeiro (4.10) foram adotados os seguintes valores: $c = y$ e $m = 1,04191e-7$

Fonte: Elaborado pelo autor.

é atuar como mediador entre quem faz a requisição e quem efetivamente atende a mesma; e, finalmente c) tempo de execução, que causaria requisições perdidas e sem resposta quando excedesse os limites estabelecidos. Causando, por fim, uma grande ineficiência ao incorrer custos em um cenário onde há pouca computação e muita espera.

No que diz respeito a memória, considerando dados das Tabelas 5 e 16, as principais implementações de FaaS se apresentam flexíveis o suficiente para uma ampla gama de *workloads*. Contudo, no que diz respeito a cálculos com grande exigência de memória, os limites estabelecidos inviabilizam seu uso. Considerando que a razão *Memória (GB) ÷ Cores* nas FaaS se mantém entre $1,5x$ e $2x$ nos principais provedores, ela se mostra baixa quando comparada a ofertas de instâncias (VMs) otimizadas para memória, com famílias como as $x1$ e $r4$ da AWS oferecendo, respectivamente, razões de $15,25x$ e $7,625x$. Situação similar poderia ser observada ao comparar a performance computacional das FaaS contra instâncias otimizadas para processamento, conforme evidência informal publicada por (GOYAL, 2018). Contudo, tais análises não representam o caso de uso projetado para as FaaS uma vez que ignoraram fatores como a ausência de custos em função de ociosidade, a granularidade da cobrança e a

Figura 28 – Análise da evolução do custo de execução no modelo de computação FaaS conforme escolhas de parâmetros de Tempo Limite de Execução (t) e Memória Alocada (y) com a linha ‘Diagonal’ mostrando o custo resultante da duplicação em ambas as dimensões (t e y)



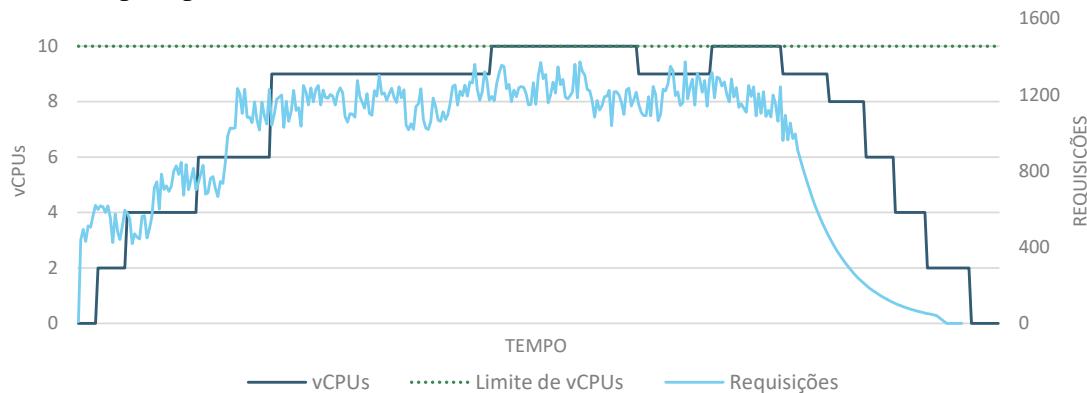
Fonte: Elaborado pelo autor.

possibilidade de paralelismo massivo.

Entretanto, ainda que a camada FaaS seja usada em cenários favoráveis compostos por alto processamento, alto paralelismo e baixa comunicação, como é o caso no modelo He-lastic, uma característica do seu cálculo de custo, conforme detalhado pela Equação 4.9, exige cuidados especialmente relevantes para a análise de custo benefício. O fato de que os parâmetros de memória, CPU e tempo limite, y , c e t , respectivamente, são multiplicados entre si, faz desta métrica uma unidade composta, tornando o cálculo de *Custo* e *Financeiro* contra-intuitivo por configurar uma função polinomial de grau três. O resultado desta característica pode ser observado na Figura 28 indicando uma possível explosão no custo em função da relação entre os parâmetros de memória e tempo limite de execução no cenário, que é simulado adotando o modelo de cobrança do provedor AWS ($c = y$, conforme Tabela 16).

Em casos de uso produtivo das FaaS é esperado que o custo observado seja inferior aos custos apresentados aqui, haja vista de que os estudos realizados adotam os limites máximos como referência, ignorando o fato de que a cobrança se dá por frações de segundos na maioria dos provedores de computação em nuvem. Ainda assim, para o modelo He-lastic e sua arquitetura baseada em duas camadas de elasticidade, tais estudos representam uma análise relevante em função de que é aceito e esperado que uma parcela das requisições encaminhadas não seja atendida pela camada FaaS em função do tempo limite de execução configurado. Desta forma, uma análise quanto ao teto de gastos é fundamental para que sejam determinados os parâmetros do modelo ao levar em conta as propriedades do problema onde o modelo He-lastic será implantado, considerando, entre outros, frequência de requisições, grau de heterogeneidade da carga computacional e orçamento disponível.

Figura 29 – Simulação do comportamento da camada de longa duração do modelo He–lastic mostrando a relação entre a quantidade de requisições e a disponibilidade de recursos para processamento.



Fonte: Elaborado pelo autor.

6.1.2 Camada do Orquestrador de Contêineres

No que diz respeito ao estudo de custo benefício da segunda camada do modelo He–lastic, previsibilidade e regularidade são fatores primordiais. Através do uso de *thresholds*, limites superiores e inferiores de carga computacional, o usuário administrador do ambiente de nuvem pode determinar a agressividade com que recursos serão adicionados e removidos. Ademais, o modelo prevê a existência de um parâmetro que determina a quantidade máxima de CPUs à disposição nesta camada, atuando como limitador de custo. Através da Figura 29 é possível visualizar o comportamento da elasticidade de acordo com a quantidade de requisições e o limite estabelecido para a quantidade de vCPUs disponíveis. Embora não estejam visíveis na figura, os *thresholds* determinam o quanto o gerenciador de elasticidade vai atuar para manter a ocupação das VMs dentro dos níveis estabelecidos.

A Tabela 25 apresenta um estudo contemplando quatro cenários em que *Custo* e *Financeiro* permanecem inalterados mesmo empregando recursos em quantidades variadas. Embora não seja abordada em detalhes como na camada FaaS, a camada baseada na orquestração de contêineres apresenta os usuais comportamentos de elasticidade, estudados em profundidade em trabalhos como Rodrigues (2016), Aubin e Righi (2015), Coutinho et al. (2015), Righi (2013), Galante e Bona (2012) e Armbrust et al. (2010).

Através do uso de um orquestrador de contêineres o modelo He–lastic alivia a carga administrativa e operacional, sendo capaz de delegar a gestão e configuração de máquinas virtuais para o provedor de computação em nuvem e focando seus esforços em preparar somente o *runtime* e o próprio código-fonte da aplicação, resultando, conforme definido nas Decisões de Projeto na Seção 4.2, em ganhos de eficiência através da redução do custo total do ambiente, contrastando com um cenário análogo onde fossem utilizados *clusters* ou *grids* computacionais. Ainda que o tempo de provisionamento de recursos em um cenário de orquestração de contêineres seja similar ao observado quando usadas apenas máquinas virtuais, a vantagem que se obtém é um

Tabela 25 – Análise do comportamento das métricas Custo e Financeiro aplicados no modelo de computação por Orquestração de Contêineres evidenciando a associatividade de custos.

#	Amostras (t)	vCPUs	Custo da Amostra	$Custo_{elastico}$	$Financeiro_{elastico}$
1	10	16	160	160	6,80
	2	2	4		
	2	4	8		
2	2	8	16	160	6,80
	2	16	32		
	2	32	64		
	2	16	32		
	2	2	4		
	2	16	32		
3	2	32	64	160	6,80
	1	48	48		
	2	8	16		
4	5	32	160	160	6,80

* Para as demais variáveis das equações de Custo e Financeiro foram adotados os seguintes valores: $i = 0$, $s = 48$ e $m = 4,25e-2$ por hora.

Fonte: Elaborado pelo autor.

ambiente de execução agnóstico ao hardware e sistema operacional, evitando a preocupação com os mesmos e permitindo colher os frutos das otimizações realizadas pelos provedores sem a necessidade de reestruturar o código de aplicação.

Diferentemente da camada FaaS, onde não há controle sobre o hardware subjacente, o uso de orquestração de contêineres permite a escolha de famílias de instâncias, dando conta de cenários como os mencionados anteriormente, na Subseção 6.1.1, onde existe uma forte demanda por uma única dimensão computacional como memória, CPU e I/O, seja de rede ou armazenamento. Através desta característica é possível ajustar o modelo He-lastic para que seja capaz de atender a uma ampla gama de cenários, mostrando sua flexibilidade e capacidade de generalização para situações além do cálculo de adequação de sistemas de substituição de sequências moleculares.

6.1.3 Camadas Combinadas

Uma vez que o cerne do modelo He-lastic está no uso de suas duas camadas de elasticidade, uma análise prévia de custos se apresenta como uma ferramenta de grande relevância para a definição dos parâmetros do modelo, conforme estabelecidos pela Tabela 13. Contudo, dada a dificuldade em determinar de antemão o tempo efetivo de execução de uma chamada na camada

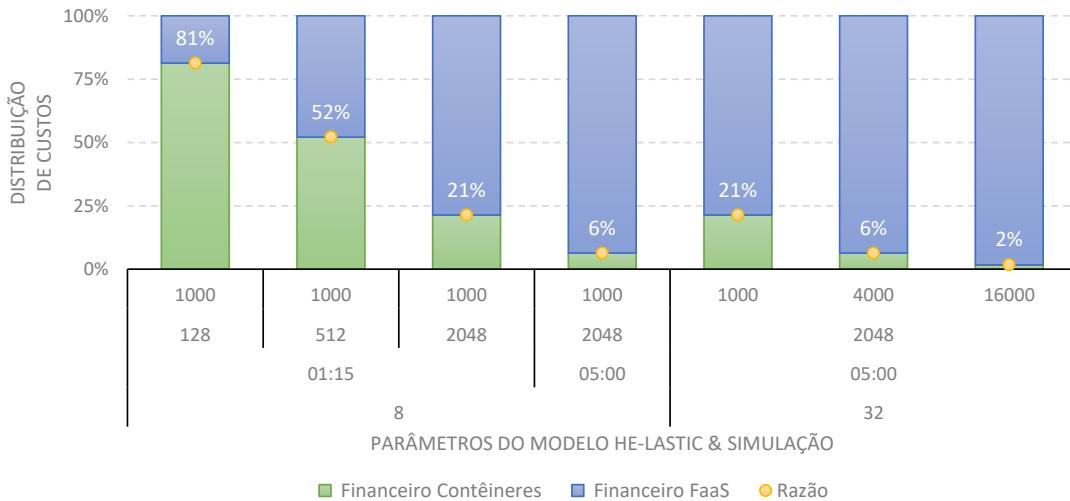
Tabela 26 – Análise de custo para o modelo He–lastic ressaltando a relação entre os custos financeiros originados nas camadas FaaS e Orquestrador de Contêineres.

vCPUs (s)	Parâmetros He–lastic		Parâmetros da Simulação		FaaS		Elástico		Total	
	Timeout (t)	Potência (y)	Requisições	Absorvidas pela Camada FaaS	Custo ($\times 10^6$)	Financeiro	Custo	Financeiro	Financeiro	Distribuição
8	01:15	128	1000	75%	1,125	0,12	8	0,34	0,46	26 : 74
				50%	0,75	0,08	8	0,34	0,42	19 : 81
				25%	0,375	0,04	8	0,34	0,38	10 : 90
	01:15	512	1000	75%	4,5	0,47	8	0,34	0,81	58 : 42
				50%	3	0,31	8	0,34	0,65	48 : 52
				25%	1,5	0,16	8	0,34	0,50	31 : 69
8	01:15	2048	1000	75%	18	1,88	8	0,34	2,22	85 : 15
				50%	12	1,25	8	0,34	1,59	79 : 21
				25%	6	0,63	8	0,34	0,97	65 : 35
	05:00	2048	1000	75%	72	7,50	8	0,34	7,84	96 : 04
				50%	48	5,00	8	0,34	5,34	94 : 06
				25%	24	2,50	8	0,34	2,84	88 : 12 [†]
32	05:00	2048	1000	75%	72	7,50	32	1,36	8,86	85 : 15
				50%	48	5,00	32	1,36	6,36	79 : 21
				25%	24	2,50	32	1,36	3,86	65 : 35 [†]
	05:00	2048	4000	75%	288	30,01	32	1,36	31,37	96 : 04
				50%	192	20,00	32	1,36	21,36	94 : 06
				25%	96	10,00	32	1,36	11,36	88 : 12
32	05:00	2048	16000	75%	1152	120,03	32	1,36	121,39	99 : 01
				50%	768	80,02	32	1,36	81,38	98 : 02
	05:00	2048		25%	384	40,01	32	1,36	41,37	97 : 03

* Para as demais variáveis das equações de Custo e Financeiro foram adotados os seguintes valores: FaaS = { $c = y$, $m = 1,04191e-7$ } e Elástico { $i = 0$, $t = 1h$, $m = 4,25e-2$ }

Fonte: Elaborado pelo autor.

Figura 30 – Análise de custo por camada para o modelo He-lastic em um cenário de absorção de 50% das requisições pela camada FaaS com base nos dados da Tabela 26.



Fonte: Elaborado pelo autor.

FaaS, assim como, a dificuldade em estimar o impacto dos *thresholds* na elasticidade da camada de Contêineres sem o profundo conhecimento do problema em mãos, ou mesmo um conjunto de resultados experimentais, a análise de custo benefício pode ser realizada por meio de cenários pessimistas, adotando os limites máximos como referência.

A decisão por uma abordagem pessimista favorece a análise de custo benefício ao reduzir para apenas três o número de parâmetros contemplados, sendo eles: (i) Tempo Limite da camada FaaS; (ii) Potência da camada FaaS; e (iii) Número de CPUs do Orquestrador de Contêineres. Os dados contidos na Tabela 26 são o resultado da análise de custo benefício por camada do modelo He-lastic ao variar os parâmetros do modelo e da simulação de carga. Através destes resultados é possível visualizar um crescimento não linear nos custos associados à camada FaaS no cenário onde há o aumento no número de vCPUs de 8 para 32 (marcado na tabela com o símbolo †), causando, em situação de carga simulada para a camada FaaS de 25%. Embora o esperado para esta situação seja um aumento de 4x no custo relacionado ao upgrade de 8 para 32 vCPUs, os dados observados são da ordem de 2,9x, com a distribuição de custo saindo de 88 : 12 para 65 : 35. Tal comportamento é suportado pela análise de custo realizada na Subseção 6.1.1 e pode ser visualizada tanto nos dados da Tabela 24, quanto no gráfico da Figura 28.

De maneira a facilitar a visualização deste comportamento, a Figura 30 apresenta um panorama sobre os dados oriundos da Tabela 26, quando limitados ao cenário que contempla uma divisão equalitária entre o atendimento das requisições disparadas. Ao analisar o gráfico, fica evidente a estreita relação entre o custo da camada FaaS e a quantidade de requisições, uma situação que não se repete na camada baseada em Orquestração de Contêineres, uma vez que seu modelo de custo considera apenas o tempo de uso dos recursos. Embora não exista, na especificação, a relação entre requisições e tempo de uso, este é um fator relevante na construção dos cenários de análise de custo benefício do modelo He-lastic, visto que, requisições que

consumem muitos recursos causarão um aumento no tempo necessário para seu processamento e resultando em um ajuste na variável t do componente $Custo_{elastico}$, conforme apresentado na Equação 4.7. A situação inversa também é relevante para análise pois mostra que o custo oriundo do Orquestrador de Contêineres pode dominar o custo total em cenários de baixo volume de curtas requisições, o que pode resultar em baixa utilização dos recursos disponíveis alocados por esta camada em função da ociosidade.

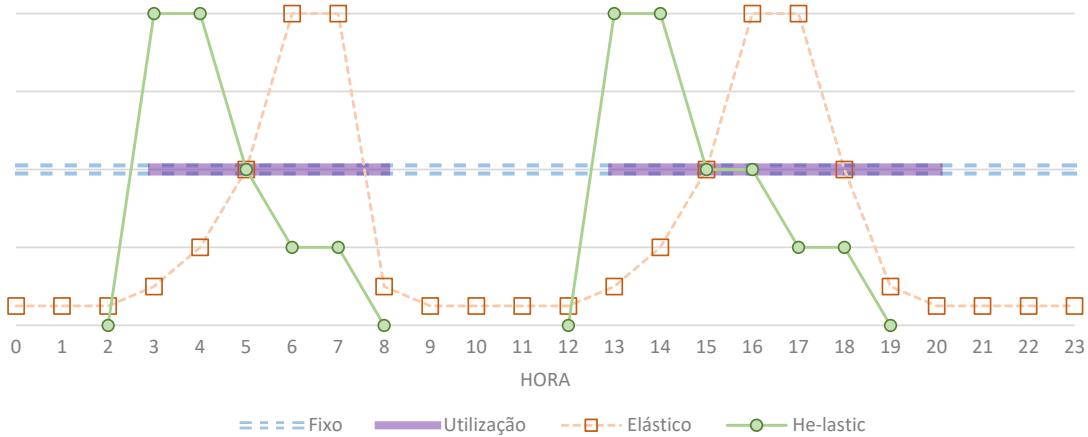
Contudo, um dos diferenciais do modelo He-lastic se mostra difícil de mapear apenas através de análises de custo em situações de carga. Devido a forma como as duas camadas de elasticidade são projetadas, uma das propriedades do modelo proposto é sua capacidade de manter custo zero em períodos de ociosidade, diferentemente de abordagens anteriores. A existência de uma camada de elasticidade baseada em FaaS fornece os mecanismos necessários para ativar, sob demanda, a segunda camada baseada em Orquestração de Contêineres, de forma que, através da combinação de camadas, torna-se possível manter inativa também a segunda camada, contudo livre do risco de rejeitar requisições por falta de servidores.

Outra característica habilitada pela abordagem em duas camadas do modelo He-lastic, é a capacidade de disponibilizar um grande número de recursos computacionais de maneira quase instantânea através da camada FaaS, evitando uma das fraquezas de abordagens baseadas puramente em elasticidade onde é característico um comportamento de *ramp-up*, ou seja, uma crescente na quantidade de recursos disponibilizados até que se obtenha uma situação de equilíbrio. Com a ajuda da camada FaaS este comportamento é atenuado em casos onde uma parcela das requisições pode ser atendida pela camada FaaS, deixando tempo hábil para que o Orquestrador de Contêineres tenha o tempo necessário para disponibilizar seus recursos.

Tanto a capacidade de manter custo zero, quanto a alteração no perfil de disponibilização de recursos, podem ser vistas na Figura 31 que utiliza um cenário simulado para ilustrar as características do modelo. A situação retratada se estende por um período de um dia, ou seja, 24 horas e contempla dois momentos onde são geradas requisições, respectivamente nas horas 3 e 13. Em uma situação de recursos fixos, como um servidor dedicado ou até mesmo um *cluster* computacional, são observados períodos significativos de ociosidade até que se configure a utilização total de seus recursos, causando grande desperdício relativo ao período ocioso. Em um cenário de elasticidade existe uma grande melhora no aproveitamento de recursos devido à capacidade de reduzir o número de processadores ativos enquanto não há trabalho a ser feito. Contudo, neste modelo de operação ainda não é possível zerar completamente a quantidade de recursos devido a ausência de gatilhos que permitam inicializar operações de elasticidade. Além disso, há o comportamento característico da elasticidade reativa baseada em *thresholds*, onde a quantidade de recursos deve progressivamente se ajustar às características da demanda, resultando em uma ascendente no consumo de recursos durante as primeiras amostras e, em alguns cenários, uma descendente ao final do processamento.

Por meio dos dados apresentados até aqui é possível concluir que a análise de custo benefício do modelo He-lastic é capaz de estabelecer apenas os limites superiores no que diz respeito ao

Figura 31 – Comportamento esperado sobre o uso de recursos ao longo do tempo conforme modelos de execução baseado em recursos fixos, elasticidade e He-lastic, evidenciando as diferenças no que diz respeito à capacidade ociosa e a evolução no aproveitamento de recursos ao longo do tempo.



Fonte: Elaborado pelo autor.

consumo de recursos e, portanto, o custo financeiro que decorre da execução do modelo com um determinado conjunto de parâmetros. Embora forneça uma orientação relevante sobre como devem ser definidos os parâmetros do modelo, tal análise permanece insuficiente para a completa determinação de seus valores conforme apresentados na Tabela 13. Este posicionamento se justifica por fatores como a cobrança por tempo efetivo de execução oriunda da camada FaaS e a dependência de dados específicos da aplicação em execução para a determinação da capacidade em termos de requisições por hora, por exemplo, da camada baseada em Orquestração de Contêineres.

Conclui-se, portanto, que a análise de custo benefício e, por consequência, a escolha dos parâmetros de execução para o modelo He-lastic requer insumos oriundos do problema sobre o qual está sendo aplicado, tornando-se uma tarefa de natureza empírica. Esta conclusão obtém apoio do artigo elaborado pelo banco espanhol de investimentos *Banco Bilbao Vizcaya Argentaria* (BBVA) que, em sua análise do modelo *Serverless* de computação, publicado em Rodríguez et al. (2018), constrói toda sua análise de custo com a ajuda de uma variável denominada ‘fator de throughput’ da aplicação. Ainda assim, prova-se, mediante os estudos realizados aqui, que é possível estabelecer o caráter de custo para um conjunto de parâmetros, determinando os limites superiores no que tange ao custo financeiro oriundo da execução do modelo e, portanto, servindo como uma base formal que auxilia nesta tarefa.

6.2 Avaliação do jModelTest

Visando obter resultados de alta confiabilidade, uma vez que estes formariam a régua sobre a qual se apoiam as medidas do modelo proposto, foram realizadas no total quase mil execuções dos testes de adequação de sistemas de substituição molecular segundo os arquivos,

Tabela 27 – Quantidade de amostras coletadas na avaliação do jModelTest para cada arquivo do *dataset* conforme o cenário.

Arquivo	Cenário / vCPUs					Total
	C_{j2}	C_{j4}	C_{j8}	C_{j16}	C_{j36}	
01-aP6	19	23	24	11	11	88
02-rodents	20	23	17	11	11	82
03-example	20	23	16	11	11	81
04-18S_insects2	20	23	22	11	11	87
05-HIVpol.groupM	20	23	17	11	11	82
06-Hex_EF1a	20	23	17	11	11	82
07-primate-mtDNA	20	23	17	11	11	82
08-HIV_vpu.ref2	20	23	19	11	11	84
09-gusanos16S.mafft	20	23	18	11	11	83
10-Birds	20	23	17	11	11	82
11-gusanosCOI.mafft	20	23	18	11	11	83
12-stamatakis-59	11	19	17	10	10	67
Total	230	272	219	131	131	983

Fonte: Elaborado pelo autor.

Tabela 28 – Coeficiente de Variação (CV) na avaliação do jModelTest contemplando tempos de execução obtidos por arquivo e cenário de avaliação.

Arquivo	Cenário / vCPUs				
	C_{j2}	C_{j4}	C_{j8}	C_{j16}	C_{j36}
01-aP6	2%	8%	18%	10%	23%
02-rodents	1%	2%	10%	0%	7%
03-example	1%	2%	10%	6%	5%
04-18S_insects2	0%	2%	14%	5%	5%
05-HIVpol.groupM	0%	1%	8%	1%	1%
06-Hex_EF1a	0%	1%	8%	1%	2%
07-primate-mtDNA	0%	1%	8%	1%	1%
08-HIV_vpu.ref2	0%	1%	9%	1%	1%
09-gusanos16S.mafft	0%	1%	8%	0%	1%
10-Birds	1%	1%	9%	0%	1%
11-gusanosCOI.mafft	0%	1%	10%	0%	1%
12-stamatakis-59	1%	1%	8%	0%	0%

Fonte: Elaborado pelo autor.

parâmetros e cenários definidos na Seção 5.1. Estas execuções foram distribuídas em quase cem (96) máquinas virtuais distintas contemplando instâncias do serviço EC2 da AWS nos seguintes modelos: `c4.2xlarge`, `c5.large`, `c5.xlarge`, `c5.2xlarge`, `c5.4xlarge`, `c5.9xlarge`. Os dados contidos na Tabela 27 apresentam os resultados da coleta de dados baseada na execução do jModelTest, totalizando, após filtragem de anomalias (*outliers*), 983 execuções por arquivo–cenário. Para obter a quantidade de chamadas à ferramenta PhyML basta multiplicar os dados da Tabela 27 pela Tabela 18, mais especificamente a quantidade de sistemas de substituição molecular contemplados por arquivo quando é seguida a estratégia de *Clustering Search*, resultando em aproximadamente 250 mil (249.184) execuções por arquivo–sistema–cenário.

A respeito da quantidade de amostras cabe ressaltar que a variação no número de coletas do cenário C_{j8} se deu em função de uma limitação do provedor AWS onde inicialmente a conta de usuário utilizada para os testes não tinha acesso à recursos computacionais da família `c5`, de última geração, de forma que, na época, a estratégia adotada foi prosseguir com os testes utilizando recursos da família `c4`, da geração anterior. O impacto destas restrições estabelecidas pelo provedor AWS também se manifesta na pequena redução da quantidade de execuções para o arquivo 12-stamatakis-59 que alocava todas as VMs disponíveis por longos períodos de tempo, principalmente nos cenários com menos recursos como C_{j2} e C_{j4} , afetando a execução dos demais cenários. Após contato com o provedor foi esclarecido que a ausência destes recursos era um mecanismo de proteção contra abusos visto que tratava-se de uma conta recém aberta. Uma vez detalhados os objetivos da requisição e o contexto de uso, os recursos `c5` foram liberados e o limite ampliado, possibilitando dar seguimento às avaliações dos cenários.

Contudo, ao detectar uma variação atípica nos tempos de execução, que se manifesta no desvio padrão e no coeficiente de variação (CV¹) das amostras coletadas, conforme evidências da Tabela 28, foi decidido por aumentar o número de execuções até obter uma variação em linha com os demais cenários e arquivos. Outro padrão que emerge da análise do coeficiente de variação por arquivo e cenário é o alto nível de oscilação apresentado pelo arquivo 01-aP6. Este comportamento está relacionado à sua curta duração e se estende para outros arquivos conforme aumenta a quantidade de recursos disponível, como mostra, por exemplo, o CV de 5% para o arquivo 04 no cenário C_{j36} . Através da análise da Tabela 28, desconsiderando-se o cenário C_{j8} pelos motivos previamente citados, é possível vislumbrar uma diagonal imaginária que abrange os dados de arquivo e cenário: {03– C_{j2} , 04– C_{j4} , 04– C_{j16} , 06– C_{j36} }.

Dentre as razões que motivam a alta variabilidade nos tempos de execução é possível citar a curta duração absoluta, medida em termos do tempo total de execução e que pode ser observada

¹ Coeficiente de Variação é uma métrica estatística amplamente utilizada para indicar a precisão, repetitividade e reproduzibilidade de avaliações, atuando também como indicador de dispersão de uma distribuição por incorporar em seu cálculo a média e o desvio padrão, sendo definido pela fórmula:

$$CV = \frac{\text{desvio padrão}}{\text{média}}$$

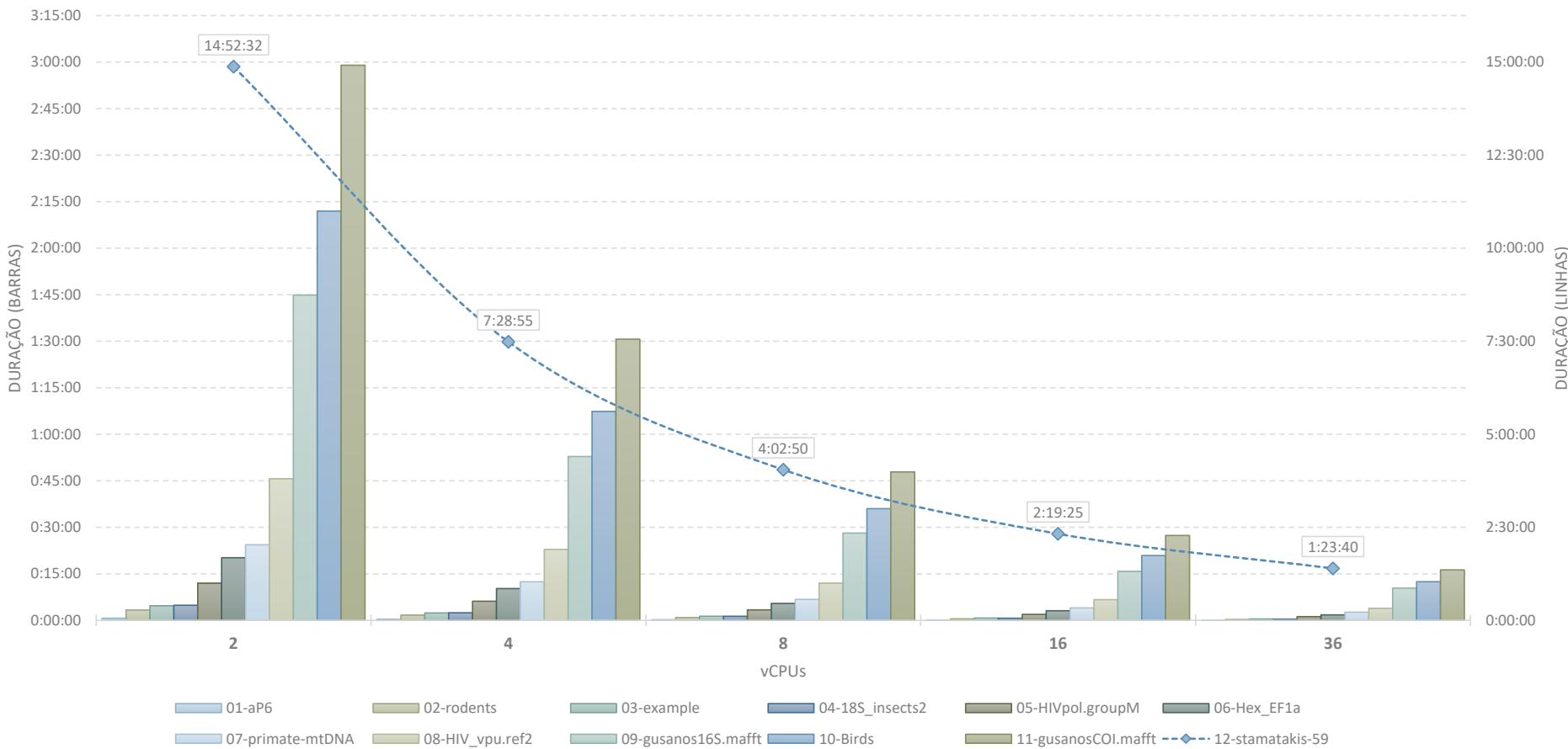
Tabela 29 – Média e Mediana dos tempos de execução observados no jModelTest por arquivo e cenário de avaliação.

Arquivo	Cenário / vCPUs									
	C_{j2}		C_{j4}		C_{j8}		C_{j16}		C_{j36}	
	Média	Mediana	Média	Mediana	Média	Mediana	Média	Mediana	Média	Mediana
01-aP6	0:00:40	0:00:40	0:00:20	0:00:21	0:00:11	0:00:11	0:00:06	0:00:06	0:00:04	0:00:04
02-rodents	0:03:19	0:03:20	0:01:40	0:01:41	0:00:54	0:00:57	0:00:29	0:00:29	0:00:17	0:00:17
03-example	0:04:41	0:04:42	0:02:22	0:02:23	0:01:19	0:01:22	0:00:44	0:00:45	0:00:26	0:00:27
04-18S_insects2	0:04:51	0:04:52	0:02:26	0:02:27	0:01:18	0:01:23	0:00:40	0:00:41	0:00:22	0:00:22
05-HIVpol.groupM	0:11:58	0:11:58	0:06:09	0:06:10	0:03:22	0:03:34	0:01:55	0:01:55	0:01:11	0:01:11
06-Hex_EF1a	0:20:11	0:20:12	0:10:14	0:10:16	0:05:27	0:05:45	0:03:05	0:03:05	0:01:45	0:01:44
07-primate-mtDNA	0:24:22	0:24:23	0:12:25	0:12:26	0:06:45	0:07:07	0:04:00	0:04:01	0:02:37	0:02:37
08-HIV_vpu.ref2	0:45:39	0:45:38	0:22:52	0:22:56	0:11:58	0:12:39	0:06:40	0:06:41	0:03:52	0:03:52
09-gusanos16S.mafft	1:44:53	1:44:52	0:52:49	0:53:06	0:28:08	0:29:46	0:15:46	0:15:47	0:10:24	0:10:24
10-Birds	2:11:57	2:11:50	1:07:22	1:07:34	0:36:00	0:37:38	0:20:53	0:20:53	0:12:27	0:12:28
11-gusanosCOI.mafft	2:58:59	2:58:49	1:30:39	1:31:03	0:47:51	0:50:14	0:27:21	0:27:22	0:16:15	0:16:15
12-stamatakis-59	14:52:32	14:51:54	7:28:55	7:30:39	4:02:50	4:15:20	2:19:25	2:19:30	1:23:40	1:23:34

* Valores apresentados em formato hh:mm:ss.

Fonte: Elaborado pelo autor.

Figura 32 – Representação gráfica da Tabela 29 mostrando os tempos de execução por arquivo e cenário conforme a mediana.



Fonte: Elaborado pelo autor.

na [Tabela 29](#) onde são detalhadas as métricas de média e mediana dos tempos de execução para cada tupla de arquivo e cenário de avaliação. Além de amplificar as diferenças relativas, medida em termos do CV, as tarefas de curta duração são especialmente sensíveis a interferências e/ou irregularidades no processamento que podem originar de uma série de fatores como: política de *scheduling* de processos e *threads*, consumo dos demais processos em execução, I/O de rede e disco, além de fatores típicos de uma ambiente de nuvem como o impacto do virtualizador e o efeito popularmente conhecido como *noisy neighbor* em que os processos em execução por outro *tenant*, que divide o mesmo recurso físico, podem afetar a performance das demais VMs.

Entretanto é possível afirmar que as métricas de média e mediana encontram-se, de maneira geral, bastante alinhadas entre si, apresentando pequenos deltas, o que, combinado com os dados de CV, indica uma distribuição estatística equilibrada e com uma tendência central bem definida. Apesar da relativamente alta variação encontrada no cenário C_{j8} quando comparado com o restante dos resultados, a afirmação anterior encontra suporte na métrica de Variância², que apresenta valores maiores do que zero em apenas quatro ocorrências, sendo elas: *a*) 16 segundos no arquivo 12-stamatakis-59 do cenário C_{j8} ; *b*) 2 segundos no arquivo 12-stamatakis-59 do cenário C_{j2} ; *c*) 1 segundo no arquivo 12-stamatakis-59 do cenário C_{j4} ; e *d*) 1 segundo no arquivo 11-gusanosCOI.mafft do cenário C_{j8} .

Os resultados gerais das execuções por arquivo–cenário, conforme a [Tabela 29](#), são apresentados de maneira visual em gráficos na [Figura 32](#) que adota como referência a métrica da Mediana para estipular os valores retratados. Devido à amplitude dos tempos de execução a figura foi dividida em dois eixos verticais, com o eixo da direita reservado para arquivos que exigem um tempo maior de processamento, como é o caso de 12-stamatakis-59. Através da representação gráfica fica evidente como o jModelTest se beneficia do aumento na quantidade de CPUs disponíveis, uma vez que este sua execução é totalmente CPU–bound. Também é possível perceber a expressiva diferença nos tempos de execução do arquivo 12-stamatakis-59 em relação ao demais onde sua execução costuma ser responsável por, em média, 63% do tempo total de execução de cada cenário. Outro destaque é o primeiro grupo de arquivos, composto por 01-aP6, 02-rodents, 03-example e 04-18S_insects2, que contabilizam tempos de execução extremamente curtos mesmo em cenários com relativa escassez de recursos como C_{j2} , C_{j4} e C_{j8} , contabilizando, somados, apenas 1% do tempo total de execução de cada cenário.

Através da análise dos tempos de processamento por sistema de substituição molecular testado pelo jModelTest é possível verificar o caráter destas execuções em termos da sua amplitude de duração. Os histogramas apresentados nas [Figuras 33](#) e [34](#) apresentam a frequência com que os sistemas avaliados se enquadram em um determinado intervalo de tempo de execução. É possível determinar pela análise das figuras que a maior parte dos sistemas avaliados se concentra

² Variância representa uma medida de dispersão estatística que indica quanto distantes do valor esperado estão os valores observados sendo calculada através da média do quadrado da distância de cada ponto até a média com a fórmula:

$$\text{VAR} = \text{média} ((X - \text{média})^2)$$

Figura 33 – Histograma da frequência dos tempos de execução para cada um dos sistemas de substituição avaliados durante a execução do arquivo 12-stamatakis-59 no cenário C_{j36} .



Fonte: Elaborado pelo autor.

Figura 34 – Histograma da frequência dos tempos de execução para cada um dos sistemas de substituição avaliados durante a execução do arquivo 07-primate-mtDNA no cenário C_{j2} .



Fonte: Elaborado pelo autor.

na ponta mais baixa das durações, contudo há também uma característica de cauda longa que contempla execuções que levam mais de 4x o tempo de execução da maioria.

Os resultados obtidos na coleta de execuções do jModelTest comprovam a adequação da escolha da aplicação para uso no modelo He-lastic, conforme definido na Seção 4.1 e explorado em detalhes na Seção 4.2. A ampla variação no tempo de execução conforme o arquivo é uma característica relevante para o usuário do jModelTest uma vez que ele não sabe (a não ser por experiência empírica) o tempo de execução e/ou os recursos necessários para que o cálculo de adequação que está prestes a realizar. Este problema torna-se ainda mais relevante em casos onde o usuário pretende executar o jModelTest em arquivos maiores, como 12-stamatakis-59, pois, uma vez que não há indicação de tempo estimado ou complexidade, não é possível saber a quantidade de recursos necessária ou adequada para requisitar em uma determinada execução supondo um ambiente de *cluster* ou *grid* computacional. A execução local de arquivos maiores, embora sem dúvida possível, exige uma significativa dose de paciência do usuário em função dos elevados tempos de execução em computadores pessoais (que atualmente costumam ter entre 4 e 8 cores de processamento) e até mesmo servidores de menor porte.

Ademais, o emprego da estratégia de *Clustering Search*, ativada por padrão no jModelTest, faz com que a execução dos cálculos de adequação de sistemas de substituição molecular deixe de compartilhar as características de computação embaralhosamente paralela para ficar mais parecido com uma computação baseada em paralelismo do tipo BSP, conforme abordado na Subseção 2.2.1. Esta mudança de caráter de paralelismo torna o jModelTest ainda mais sensível

ao maior esforço computacional exigido por sistemas de substituição mais complexos assim como possíveis *outliers* e execuções atrasadas que acabam por determinar o tempo máximo de execução independente da média ou dos demais sistemas avaliados. Embora não seja objetivo do modelo He–lastic atenuar ou tratar esta característica, a adoção de elasticidade pode resultar em um significativo benefício no que diz respeito ao custo financeiro uma vez que recursos ociosos serão descartados enquanto aguardam o término da computação dos sistemas de substituição mais complexos. Além disso a associatividade de custos que emerge da elasticidade permite empregar um número maior de recursos durante momentos de pico enquanto mantém um custo similar ao modelo de alocação fixa por meio do descarte de recursos ociosos.

$$\Delta_{recursos} = \frac{Recursos(C_{x-1})}{Recursos(C_x)} \quad (6.1)$$

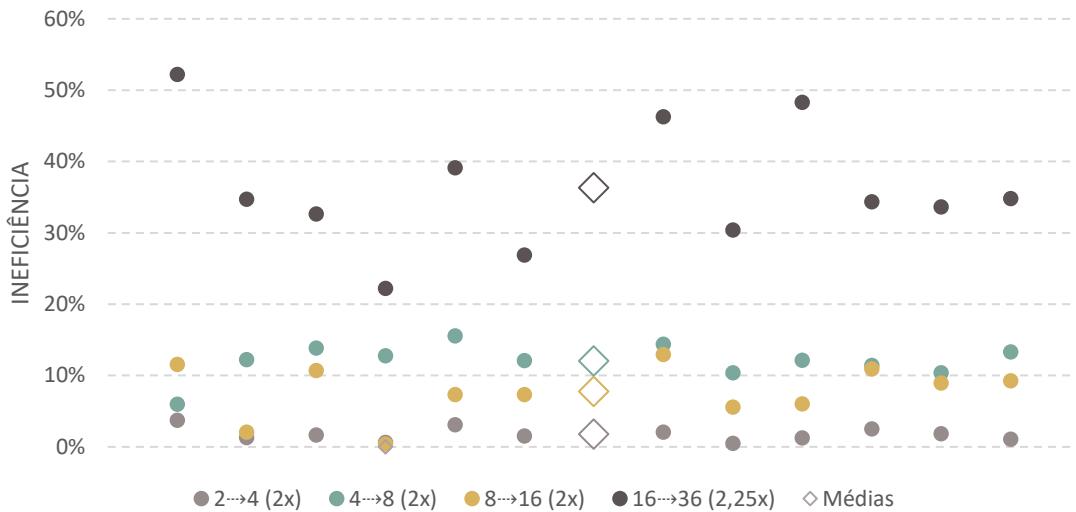
$$Perda = \left(\frac{Tempo(C_x)}{Tempo(C_{x-1}) \times \Delta_{recursos}} \right) - 1 \quad (6.2)$$

O impacto da estratégia de *Clustering Search* também pode ser verificado através da análise de eficiência do jModelTest. Para determinar esta métrica foram tomadas as médias dos tempos de execução por cenário e arquivo, conforme consta na [Tabela 29](#), e realizada a comparação entre cada cenário e o seguinte calculando-se a razão entre a diferença de tempos de execução e de recursos. Desta forma é possível obter um índice normalizado, descrito na [Equação 6.2](#), que expressa a perda, ou ganho, de desempenho conforme varia a quantidade de recursos disponível em cada cenário de execução. Desta forma, uma execução que levava uma hora pra executar com duas unidades de recursos teria 0% de perdas caso levasse a metade do tempo para executar com o dobro de unidades de recursos, no caso meia hora e quatro unidades.

Através da aplicação desta formula torna-se possível visualizar na [Figura 35](#) o comportamento do jModelTest em relação à sua eficiência conforme varia o uso de recursos indicando, de certa forma, seu comportamento de escalabilidade. Na figura mencionada as cores determinam os cenários e a variação na quantidade de recursos e cada ponto representa um arquivo do *dataset* apontando as perdas observadas pelos tempos médios de execução. Como não foram observados ganhos de eficiência a figura apresenta apenas as perdas conforme variam os recursos disponíveis para cada arquivo e, destacados através dos losangos, as médias gerais para cada cenário.

Desta forma é possível visualizar um expressivo aumento na ineficiência na transição entre os cenários C_{j16} para C_{j36} , reportando uma perda de, em média, 36% apesar do aumento de 2,25x na quantidade de recursos disponíveis. Enquanto isso são observadas perdas entre 12% e 8% para as transições entre os cenários C_{j4} para C_{j8} e C_{j8} para C_{j16} respectivamente. Por fim, a transição do cenário C_{j2} para C_{j4} apresenta menor perda média com apenas 2% embora a média geral dentre todos os arquivos do *dataset* e cenários de teste figure em 14% ineficiência. Estes resultados corroboram com observações quanto a utilização de recursos realizadas durante o acompanhamento da execução dos cenários, onde foi possível visualizar longos períodos de recursos ociosos, neste caso *cores* de processamento, conforme a execução se aproxima do final

Figura 35 – Perda de eficiência observada durante as execuções do jModelTest através das médias dos tempos de execução por arquivo (pontos) e transição de cenários (cores) com a média geral em destaque.



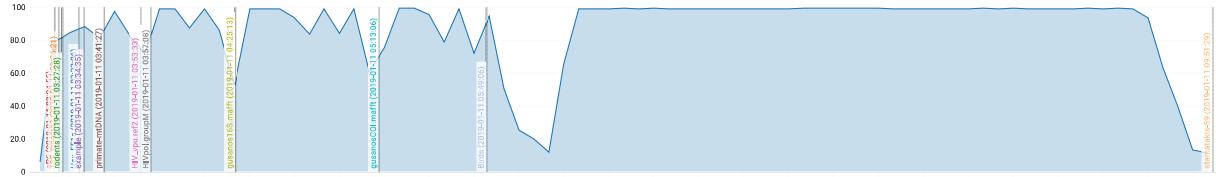
Fonte: Elaborado pelo autor.

de um dos seis estágios da execução por *Clustering Search* (vide Tabela 15).

Embora não seja parte dos cenários propostos na Metodologia de Avaliação, foi realizada uma única execução em uma VM dotada de 72 vCPUs para confirmar a tendência de aumento no desperdício conforme aumentam os recursos disponíveis. Neste teste foi observada uma perda média, ou seja, ao longo de todos os arquivos do *dataset*, de 79% (ou 64% excluindo o arquivo 01-aP6), um aumento expressivo frente à ineficiência de 36% observada na transição dos cenários C_{j16} para C_{j36} . A análise de ambas as pontas do espectro no que diz respeito ao tempo de execução fornecem *insights* valiosos sobre a utilização de recursos onde o arquivo 01-aP6, o menor do *dataset* apresentou uma regressão, contabilizando perdas de 235%, negando qualquer benefício do aumento de recursos uma vez que seu tempo de execução já era bastante reduzido. Na ponta oposta, o arquivo 12-stamatakis-59 apresentou uma ineficiência de 43%, contrabalanceando a maior disponibilidade de recursos com os longos períodos de ociosidade em função do processamento em fases. Contudo há de se advertir que estes dados podem não refletir fielmente a realidade (principalmente nos arquivos menores) visto que o cenário foi executado apenas uma vez e desta forma servem apenas como indicativo de tendência, sem prover a confiança estatística necessária para tirar conclusões.

Para explorar os motivos das perdas de eficiência reportadas é possível recorrer às Figuras 36, 37, 38 e 39 que apresentam a utilização média de CPU em períodos de um minuto conforme medida pelo provedor AWS no serviço CloudWatch enquanto ocorre uma execução completa do cenário de testes, contemplando uma única execução do jModelTest para cada arquivo do *dataset*, onde 100% equivale ao uso completo de todos os *cores* disponíveis. Cada linha vertical corresponde à transição entre a execução referente a um arquivo e o próximo, evidenciando o comportamento do uso de CPU durante o cálculo de adequação de sistemas de substituição

Figura 36 – Utilização média de CPU ao longo de uma execução contemplando todos os arquivos do *dataset* de testes em uma VM com 8 *cores* de processamento equivalente ao cenário C_{j8} (Duração total: 6 horas, 28 minutos).



Fonte: Capturado pelo autor a partir do serviço CloudWatch do provedor AWS.

Figura 37 – Utilização média de CPU ao longo de execução contemplando todos os arquivos do *dataset* de testes em uma VM com 16 *cores* de processamento equivalente ao cenário C_{j16} (Duração total: 3 horas, 44 minutos).



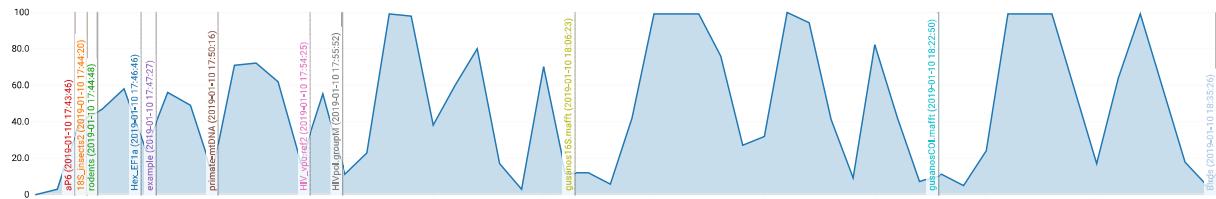
Fonte: Capturado pelo autor a partir do serviço CloudWatch do provedor AWS.

Figura 38 – Utilização média de CPU ao longo de uma execução contemplando todos os arquivos do *dataset* de testes em uma VM com 36 *cores* de processamento equivalente ao cenário C_{j36} (Duração total: 2 horas, 17 minutos).



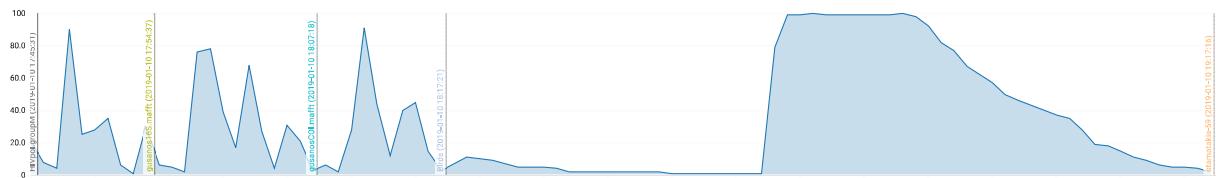
Fonte: Capturado pelo autor a partir do serviço CloudWatch do provedor AWS.

Figura 39 – Detalhe da Figura 38 excluindo a execução do arquivo 12-stamatakis-59.



Fonte: Capturado pelo autor a partir do serviço CloudWatch do provedor AWS.

Figura 40 – Utilização média de CPU ao longo de uma execução contemplando os quatro maiores arquivos do *dataset* de testes em uma VM com 72 *cores* de processamento com o intuito de verificar o comportamento do jModelTest (Duração total: 1 hora, 43 minutos).



Fonte: Capturado pelo autor a partir do serviço CloudWatch do provedor AWS.

molecular. Na [Figura 38](#), que representa uma amostra do cenário C_{j36} , é possível visualizar, no período que contempla os quatro últimos arquivos, por exemplo, o comportamento intermitente de uso de CPU e o desperdício de recursos em função da característica faseada do cálculo realizado através da técnica de *Clustering Search*. A [Figura 40](#) retrata uma execução em VM com 72 *cores* de processamento que, embora não esteja formalmente incluída na metodologia de avaliação, suporta a intuição a respeito dos limites de escalabilidade do jModelTest haja vista sua a incapacidade de aproveitar todos os recursos disponíveis e o adiantamento da rampa descendente no uso de recursos do último arquivo, que resulta em um aumento da ociosidade.

Os impactos da estratégia de *Clustering Search* se destacam conforme aumenta a disponibilidade de recursos e principalmente no caso do último, e maior, arquivo (12-stamatakis-59) que faz uso de apenas duas fases com 8 e 120 sistemas, respectivamente, onde é possível observar um longo período de ociosidade de recursos até que todos os 8 sistemas sejam avaliados. Assim que encerra o cálculo da primeira fase o programa faz uso de todos os *cores* disponíveis até que se inicie, novamente, um longo período de ociosidade enquanto os últimos sistemas são processados. Estes períodos de longa ociosidade são o alvo das ações de elasticidade propostas no modelo He–lastic por meio da camada baseada em Orquestração de Contêineres.

Através da [Figura 39](#), que apresenta os mesmos dados exceto o arquivo 12-stamatakis-59, é possível observar o baixo nível de utilização média de CPU ao longo da execução do cenário apresentado. O comportamento de fases, conforme detalhado na [Tabela 15](#), torna-se ainda mais evidente assim como o desperdício observado durante a execução de arquivos menores, onde não se utiliza completamente os recursos disponíveis, o que indica uma alocação excessiva. Desta forma o modelo He–lastic lança mão da primeira camada de elasticidade que tem como objetivo atenuar a ociosidade e consequente desperdício de recursos que ocorre ao processar arquivos menores, visando alocar apenas a quantidade necessária.

6.3 Avaliação do He–lastic

Os resultados do modelo He–lastic, coletados por meio do protótipo desenvolvido, seguem os mesmos moldes dos cenários de avaliação centrados no jModelTest. Esta estratégia visa estabelecer uma comparação o mais fiel possível enquanto não deixa de endereçar as diferenças existente entre as abordagens avaliadas, principalmente no que diz respeito à camada FaaS do modelo proposto. Desta forma foram avaliados, assim como na seção anterior, os cenários de testes estabelecidos na [Seção 5.1](#) e apresentados na [Tabela 30](#) que apresenta os resultados da coleta de dados baseada na execução do protótipo do modelo He–lastic, totalizando, após filtragem de anomalias (*outliers*), 1063 execuções por arquivo–cenário. Para obter a quantidade de chamadas à ferramenta PhyML basta multiplicar os dados da [Tabela 27](#) pela [Tabela 18](#), mais especificamente a quantidade de sistemas de substituição molecular contemplados por arquivo quando é seguida a estratégia de *Clustering Search*, resultando em aproximadamente 270 mil (268.992) execuções por arquivo–sistema–cenário.

Tabela 30 – Quantidade de amostras coletadas na avaliação do modelo He–lastic para cada arquivo do *dataset* conforme o cenário.

Arquivo	Cenário											Total	
	FaaS				Contêineres				Misto				
	C_{f0}	C_{c8}	C_{c16}	C_{c36}	C_{m1}	C_{m2}	C_{m3}	C_{m4}	C_{m5}	C_{m6}	C_{m7}		
01-aP6	6	5	10	10	11	11	5	5	4	5	5	77	
02-rodents	6	5	10	10	11	9	5	5	5	5	5	76	
03-example	6	5	9	10	11	11	23	5	5	5	5	95	
04-18S_insects2	6	5	5	5	11	11	5	5	5	5	5	68	
05-HIVpol.grou[..]	6	5	6	10	11	33	17	5	5	5	5	108	
06-Hex_EF1a	6	5	5	9	11	32	5	5	5	5	5	93	
07-primate-mtDNA	6	5	5	10	11	31	17	17	5	5	5	117	
08-HIV_vpu.ref2	6	5	5	10	10	22	16	5	5	5	5	94	
09-gusanos16S[..]	5	5	5	10	11	13	11	5	5	11	5	86	
10-Birds	5	5	5	9	10	10	11	5	5	11	5	81	
11-gusanosCOI[..]	5	5	5	10	11	11	11	5	5	11	5	84	
12-stamatakis-59	6	5	5	5	11	16	10	5	5	11	5	84	
Total	69	60	75	108	130	210	136	72	59	84	60	1063	

Fonte: Elaborado pelo autor.

Tabela 31 – Coeficiente de Variação (CV) na avaliação do He–lastic contemplando tempos de execução obtidos por arquivo e cenário de avaliação.

Arquivo	Cenário												
	FaaS				Contêineres				Misto				
	C_{f0}	C_{c8}	C_{c16}	C_{c36}	C_{m1}	C_{m2}	C_{m3}	C_{m4}	C_{m5}	C_{m6}	C_{m7}		
01-aP6	9%	22%	10%	15%	9%	6%	5%	5%	5%	0%	12%		
02-rodents	8%	3%	11%	6%	7%	11%	17%	7%	7%	8%	2%		
03-example	3%	20%	12%	15%	3%	12%	9%	8%	8%	3%	9%		
04-18S_insects2	3%	8%	9%	5%	3%	9%	10%	6%	9%	6%	9%		
05-HIVpol.groupM	4%	3%	11%	7%	5%	6%	3%	5%	6%	4%	11%		
06-Hex_EF1a	5%	7%	10%	18%	7%	5%	6%	5%	4%	5%	4%		
07-primate-mtDNA	4%	5%	8%	8%	5%	5%	5%	40%	7%	9%	5%		
08-HIV_vpu.ref2	18%	18%	9%	8%	18%	16%	20%	11%	2%	16%	2%		
09-gusanos16S.mafft	0%	5%	4%	9%	7%	4%	8%	9%	6%	12%	6%		
10-Birds	0%	4%	8%	7%	18%	3%	16%	7%	6%	19%	1%		
11-gusanosCOI.mafft	0%	4%	10%	11%	1%	6%	2%	5%	7%	7%	2%		
12-stamatakis-59	0%	2%	1%	5%	2%	15%	3%	7%	6%	4%	6%		

Fonte: Elaborado pelo autor.

Apesar do número similar de execuções em relação às avaliações do jModelTest, nos testes do modelo proposto foi necessário reduzir a quantidade de amostras para viabilizar a execução de todos os cenários propostos. Desta forma foi estabelecido que cinco seria o número mínimo desejado de amostras, menor que os onze desejados na avaliação do jModelTest, contudo ainda significativos o suficiente para fornecer uma rede de segurança estatística para que se possam tirar conclusões acerca dos resultados. Os pontos de maior concentração no número de amostras coletadas indicam áreas de interesse devido ao comportamento dinâmico em duas camadas do modelo ou cenários que apresentavam alta variação nos tempos de execução e receberam maior atenção para garantir uma melhor compreensão do comportamento.

No que diz respeito ao Coeficiente de Variação pode-se perceber através dos dados da Tabela 31 que houve um evidente aumento. Esta mudança no comportamento da métrica CV pode ser explicada pela redução na quantidade de execuções, cujos objetivos caíram de 11 para 5 em função do aumento na variedade de cenários avaliados. Além disso é possível perceber que algumas combinações geram resultados atípicos como as tuplas [07-primate-mtDNA, C_{m4}], [01-aP6, C_{c8}], [03-example, C_{c8}], [08-HIV_vpu.ref2, C_{m3}], com CVs de 40%, 22%, 20% e 20%. A respeito da combinação 07-primate-mtDNA– C_{m4} , com 40% de CV a explicação se dá pela agressiva redução no tempo limite de execução (*timeout*) para a camada FaaS, que passou de 60s na maioria dos cenários para apenas 15s. Esta ampla variação pode se originar em função de uma aproximação do tempo limite de execução com o tempo médio de execução de cada teste de adequação de sistemas de substituição molecular gerando *retries* que, devido a existência de uma camada de *caching* de recursos presente no FaaS, permite que as execuções subsequentes sejam mais rápidas. Outra possibilidade é a pura variação nos patamares de performance entregue pela camada FaaS que, embora seja a mais confiável dentre os grandes provedores de computação em nuvem (WANG et al., 2018), ainda demonstra ampla variação nos recursos subjacentes como demonstrou a Tabela 22.

Contudo, assim como nas execuções do jModelTest é possível afirmar que as estatísticas mostram uma distribuição concentrada nos pontos centrais como média e mediana. A métrica de Variância, nas execuções do modelo He-lastic também permaneceu zerada para a maior parte das 132 combinações cenário–arquivo, apenas apresentando valores de: dez segundos para a combinação 12-stamatakis-59 cenário C_{m2} , dois segundos para a combinação 01-aP6 cenário C_{c8} e apenas um segundo para outras 6 combinações, totalizando somente 6% de ocorrências maiores que zero. A ampla variância do arquivo 12-stamatakis-59 no cenário C_{m2} pode ser explicada por sua execução acontecer sempre apenas na camada de Orquestração de Contêineres, devido à incapacidade da camada FaaS de absorver as requisições por falta de potência e tempo hábil de processamento. Desta forma existe uma injeção de ruído no tempo de execução oriunda das ações de elasticidade do Orquestrador de Contêineres ao adicionar e remover recursos que é potencializada com a variação adicional decorrente da aplicação da política de *retry* por *backoff* exponencial imposta pelo provedor AWS, e descrita em detalhes na Subseção 5.4.4.

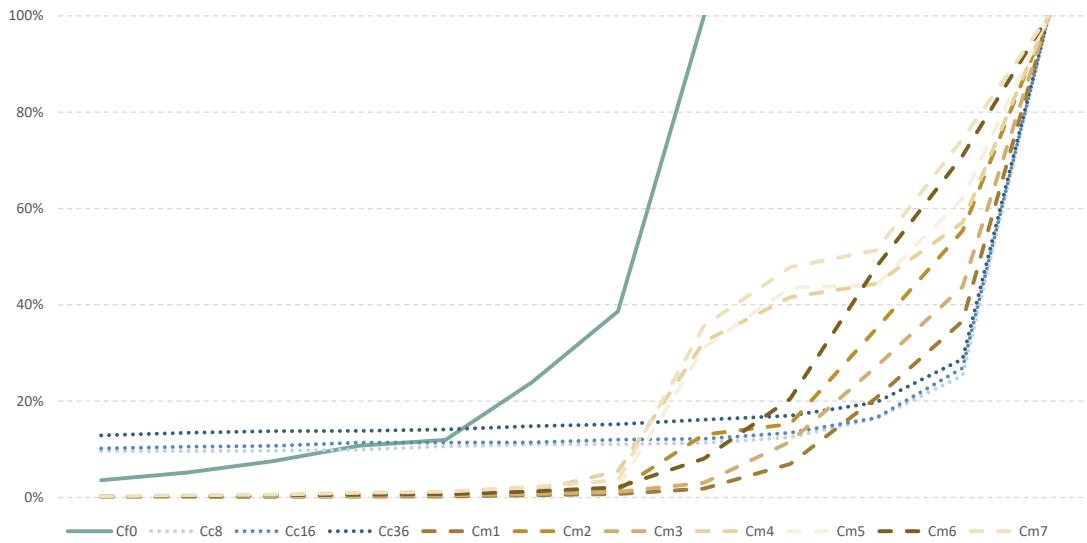
Uma análise da Tabela 32 permite identificar três comportamentos distintos que se manifestam

Tabela 32 – Mediana dos tempos de execução observados no modelo He–lastic por arquivo e cenário de avaliação.

Arquivo	Tempo de Execução (Mediana) / Cenário										
	FaaS	Contêineres				Misto					
		C_{f0}	C_{c8}	C_{c16}	C_{c36}	C_{m1}	C_{m2}	C_{m3}	C_{m4}	C_{m5}	C_{m7}
01-aP6	0:00:09	0:23:14	0:13:46	0:09:48	0:00:09	0:00:11	0:00:12	0:00:09	0:00:09	0:00:11	0:00:12
02-rodents	0:00:13	0:19:49	0:12:03	0:09:07	0:00:12	0:00:14	0:00:13	0:00:12	0:00:12	0:00:14	0:00:22
03-example	0:00:27	0:22:42	0:13:04	0:10:04	0:00:24	0:00:24	0:00:27	0:00:21	0:00:24	0:00:27	0:00:45
04-18S_insects2	0:00:19	0:20:28	0:11:40	0:09:20	0:00:20	0:00:16	0:00:20	0:00:17	0:00:17	0:00:20	0:00:28
05-HIVpol.groupM	0:00:30	0:19:49	0:12:16	0:08:54	0:00:27	0:00:27	0:00:33	0:00:28	0:00:28	0:00:32	0:00:56
06-Hex_EF1a	0:01:00	0:19:59	0:13:07	0:08:32	0:00:58	0:01:00	0:00:57	0:00:56	0:01:01	0:01:02	0:01:41
07-primate-mtDNA	0:01:37	0:22:37	0:13:58	0:10:41	0:01:35	0:01:35	0:01:33	0:04:06	0:01:35	0:01:43	0:02:45
08-HIV_vpu.ref2	0:04:11	0:25:41	0:15:23	0:11:14	0:04:03	0:13:01	0:04:04	0:33:17	0:34:10	0:06:36	0:36:58
09-gusanos16S.mafft	—	0:33:44	0:18:57	0:13:04	0:45:18	0:34:59	0:36:11	0:31:10	0:33:45	0:39:14	0:39:43
10-Birds	—	0:21:52	0:13:04	0:09:08	0:15:09	0:15:13	0:15:24	0:24:16	0:24:06	0:16:46	0:27:33
11-gusanosCOI.mafft	—	0:52:04	0:30:46	0:19:01	1:20:06	0:55:15	0:58:15	0:42:48	0:48:09	0:58:09	0:57:34
12-stamatakis-59	—	3:25:39	1:54:33	1:06:10	3:38:43	1:39:45	2:13:11	1:14:59	1:17:30	1:21:56	1:17:19

Fonte: Elaborado pelo autor.

Figura 41 – Gráfico de evolução do tempo de execução observado a cada arquivo e cenário quando expressados e ordenados em termos do percentual do tempo máximo de execução.



Fonte: Elaborado pelo autor.

através das medianas do tempo total de execução de cada cenário. Ao visualizar na Figura 41 os dados de registros de execução, normalizados através do cálculo do percentual do tempo máximo de execução e ordenados pelo seu valor numérico é possível perceber três grupos de comportamentos bem definidos e distintos entre si. Entretanto, o surgimento de três ‘famílias’ de comportamento não surpreende uma vez que este é o mesmo resultado que emerge da combinação entre as camadas de elasticidade e, por consequência, os cenários de avaliação, comprovando, também, a relevância da estratégia de duas camadas de elasticidade adotada pelo modelo He–lastic, uma vez que as características da interação entre as duas camadas dão origem a um comportamento que é visivelmente diferente daquele demonstrado pelas demais camadas de elasticidade quando funcionando de maneira isolada.

Representado pela linha verde contínua, o cenário apenas FaaS (C_{f0}) se destaca dos demais ao atingir a marca de 100% antes dos restantes, um comportamento que se origina da incapacidade de concluir o processamento de todos os arquivos do *dataset* devido às restrições impostas pelo modelo FaaS no que diz respeito ao uso de recursos e tempo de execução. As linhas azuis pontilhadas representam o grupo baseado na Orquestração de Contêineres, contemplando os cenários C_{c8} , C_{c16} e C_{c36} , onde é possível observar que as primeiras execuções, apesar de compartilhar a progressividade característica dos arquivos contemplados no *dataset*, partem de um patamar mais elevado do que às demais. Embora os patamares iniciais entre os grupos somente FaaS e somente Contêineres não pareçam tão distantes, uma análise dos dados absolutos via Tabela 32 revela que há, de fato, uma larga distância entre eles, sendo o fenômeno observado no gráfico um resultado da relativização em termos do percentual do tempo total de execução. Para compreender melhor as origens dos três comportamentos distintos, uma análise detalhada é

apresentada a seguir.

6.3.1 Cenário Somente FaaS

O paradigma Function as a Service / Serverless permite a simplificação do ambiente das configurações, promete amplo paralelismo e fornece faturamento contabilizado em unidades de 100ms, o que garantiu grande popularidade e adoção no meio comercial. Em contrapartida, nos ambientes acadêmicos ainda são encontradas poucas referências e análises a respeito do assunto, comprovando a necessidade de estudo e melhor caracterização deste modelo computacional. Desta forma o cenário Somente FaaS (C_{f0}) foi incluído no conjunto de cenários da metodologia de avaliação com o intuito de viabilizar um melhor conhecimento a respeito e verificar os níveis de impacto que podem ser causados pela escolha de FaaS.

A execução do cenário C_{f0} teve configurados os seguintes parâmetros: Potência/Memória e Tempo Limite com os valores de 1536 e 60, respectivamente. A escolha dos parâmetros não é por acaso haja vista que é ele que determina o limite superior aos parâmetros das execuções dos cenários Mistos, uma decisão que tem por objetivo permitir uma comparação justa entre os cenários e facilitar a identificação de *overhead* ou interferências do modelo proposto.

Um dos pontos mais convincentes sobre a adoção de FaaS está na sua capacidade de responder rapidamente à mudanças na carga de trabalho através do lançamento de novas unidades de execução. Determinar o número de processadores, processos ou contêineres envolvidos na execução é, sem dúvida, um fator relevante para este e outros trabalhos. Através da extração de *logs* do serviço CloudWatch do provedor de computação em nuvem escolhido é possível capturar o identificador da unidade de execução, geralmente chamado de contêiner ou instância, embora ambos os termos já representem outros conceitos. De posse destes identificadores torna-se possível contabilizar as unidades de execução e verificar questões como reuso e grau de paralelismo na execução dos cenários de avaliação. A fim de reduzir possíveis equívocos foi adotado o nome de Processos, que representam, portanto, a quantidade de unidades distintas de execução alocadas pelo provedor para servir requisições.

A Tabela 33 apresenta, sumarizados através das médias, os resultados encontrados para este cenário assim como uma comparação com o cenário de recursos fixos baseado no jModelTest. A comparação da razão entre os tempos de execução e os tempos médios observados pelos cenários C_{j2} e C_{j36} contempla os dois extremos do espectro que avaliou a execução do jModelTest na Seção 6.2 e permite detectar o impacto de performance ocasionado pelo uso da tecnologia FaaS. São detectados, nesta análise, três casos onde o *overhead* da tecnologia FaaS causou um atraso ou impactou a performance da execução sendo, nomeadamente: a) 01-aP6; b) 03-example; e c) 08-HIV_vpu.ref2.

No que diz respeito ao arquivo 01-aP6, embora o valor percentual de 223% possa assustar, esta variação se deve ao pequeno valor absoluto, menor que dez segundos e, portanto, muito sensível à variações, mesmo que não sejam significativas para o usuário, como é o caso aqui. Já

Tabela 33 – Processos e Tempo de Execução médios do cenário somente FaaS (C_{f0}) do modelo He-lastic.

	Processos	CPU-time	Execução	Percentual Sobre		
				CPU-time	C_{j2}	C_{j36}
01-aP6	63	0:01:31	0:00:09	10%	23%	223%
02-rodents	114	0:04:34	0:00:13	5%	7%	75%
03-example	118	0:06:51	0:00:27	6%	9%	100%
04-18S_insects2	117	0:06:21	0:00:19	5%	6%	83%
05-HIVpol.groupM	120	0:11:56	0:00:30	4%	4%	42%
06-Hex_EF1a	120	0:22:34	0:00:59	4%	5%	57%
07-primate-mtDNA	120	0:34:26	0:01:37	5%	7%	62%
08-HIV_vpu.ref2	122	1:42:02	0:04:41	5%	10%	121%
Média Geral	112	—	—	5%	8%	84%

Fonte: Elaborado pelo autor.

no caso do arquivo 03-example as médias dos tempos de execução via FaaS foram praticamente iguais aos do cenário C_{j36} , com durações de 27 e 26 segundos, respectivamente, um resultado que pode ser considerado ainda satisfatório considerando-se o alto nível de paralelismo obtido através dos quase 120 processos mais de 3 vezes o observado no cenário C_{j36} , que estava limitado a 36 cores. O arquivo 08-HIV_vpu.ref2 apresenta uma situação interessante que se manifesta no impacto de aproximadamente 20% na performance quando comparada ao cenário C_{j36} e que pode ser explicado pela ocorrência de *retries*, detectável através do número maior de processos paralelos (122) do que o paralelismo máximo (120) dos estágios de processamento do jModelTest, como detalhado na Tabela 15.

A existência de *retries* durante o processamento indica a proximidade com os limites estabelecidos para a camada FaaS no cenário C_{f0} , onde os recursos alocados não são mais suficientes para dar conta do processamento solicitado. Contudo, mesmo se aproximando do ponto de saturação e excedendo o tempo limite em alguns processos, é interessante notar que o arquivo ainda é processado integralmente, o que parece contra intuitivo uma vez que não houve alteração nos parâmetros estabelecidos para o cenário. Este fenômeno pode ser explicado por duas características que trabalham juntas e influenciam o tempo de execução de cada processo paralelo, sendo a primeira delas as pequenas variações de performance que ocorrem em função da virtualização e de fatores fora do controle da função FaaS como *scheduler* de sistema operacional e contenção de leitura e escrita. A segunda característica é a existência de um *cache* na camada FaaS que evita a necessidade de obter um novo ambiente de execução e até mesmo os arquivos de entrada que também ficam em *cache* em área temporária de armazenamento, o que acelera de maneira significativa uma reexecução ao remover a necessidade de buscar recursos remotos.

Ainda assim, no resultado geral obtido, é possível dizer que o cenário com apenas FaaS obteve tempos de execução em linha com as execuções do jModelTest com 36 *cores* do cenário C_{j36} , alcançando uma duração de 84%, na média entre os arquivos que ele foi capaz de processar integralmente. Quando comparado com o cenário de apenas dois *cores*, C_{j2} a execução via FaaS levou, em média, somente 8% do tempo total observado, uma significativa melhoria obtida através do alto grau de paralelismo possibilitado pelo uso da camada FaaS. A respeito do paralelismo é possível observar que se manteve muito próximo do limite de 120 estabelecido pela execução em fases do jModelTest, com a média geral em 112 processos paralelos em execução, confirmando a capacidade da camada FaaS de obter rapidamente o alto nível de paralelismo prometido, onde impressiona o resultado do arquivo 02-rodents onde foram observados 114 processos paralelos em apenas 13 segundos de execução total.

No que tange ao tempo de execução em relação ao tempo total de processamento utilizado, chamado na Tabela 33 de CPU-time, é possível perceber que este se mantém estável em torno de 5% no geral, com a notável exceção sendo o arquivo 01-aP6 que reporta um tempo de execução de 10% do tempo total de processamento. A relativa estabilidade deste indicador pode ser explicada devido a natureza do processamento realizado pelo jModelTest e pelo protótipo do modelo He-lastic, onde cada processo paralelo é responsável por executar o cálculo de *best-fit* de sistemas de substituição molecular. Neste caso, os sistemas executados são sempre os mesmos e a variação se dá através do arquivo de múltiplos alinhamentos de sequências, conforme definidos na Seção 5.1, de forma que o sistema com maior tempo de duração do cálculo acaba se tornando balizador do tempo de execução, uma vez que todas as durações são proporcionais ao arquivo fornecido como parâmetro de entrada.

Por fim é importante notar que a grande vantagem da execução via FaaS, e do modelo He-lastic, está na sua capacidade de cobrar apenas pelo tempo efetivo de execução, sendo este o grande fator de diferenciação quando comparado com os cenários de base do jModelTest, que se apoiam na existência de servidores, *grids* ou *clusters* computacionais sempre disponíveis. A comparação dos tempos de execução indica que, embora exista um *overhead* oriundo da adoção do FaaS, uma vez que seu tempo de execução empregando mais de cem processos foi praticamente o mesmo do que o jModelTest com apenas 36 *cores*, este *overhead* não é significativo o suficiente para justificar o abandono do FaaS como alternativa viável. Soma-se a isto o fato observado na Seção 6.2 de que o jModelTest tende a perder eficiência conforme aumentam os recursos disponíveis, causando retornos decrescentes no investimento em hardware. Contudo é evidente, também, que a computação via FaaS tem suas falhas, manifestando-se claramente na incapacidade de processar os arquivos maiores do *dataset* em função dos limites de potência e tempo de execução. Ademais, ainda que estes limites fossem flexibilizados, eventualmente o custo monetário, conforme visto na Seção 6.1, tornaria sua utilização proibitiva, uma vez que o objetivo dos provedores de computação em nuvem com este tipo de serviço é incentivar a rotatividade através de curtas execuções, o que inviabiliza sua adoção como plataforma de computação de propósito geral.

6.3.2 Cenário Somente Orquestrador de Contêineres

Ao contrário do FaaS, a elasticidade por regra–condição–ação, utilizada pelo Orquestrador de Contêineres já encontra-se estabelecida tanto no meio comercial quanto acadêmico, sendo um dos pilares da computação em nuvem. As avaliações realizadas nos cenários compostos somente por Orquestração de Contêineres buscaram verificar o comportamento dinâmico da elasticidade e o seu impacto de performance frente a execuções do jModelTest em um ambiente não distribuído com paralelismo obtido por meio de *threads*. Através desta metodologia é possível observar o impacto na performance e consequente *overhead* computacional pelo emprego da elasticidade frente a um cenário ótimo do jModelTest, uma vez que a execução por *threads* causa o menor impacto quando comparada com a computação distribuída por MPI, por exemplo.

O uso do serviço Batch, do provedor AWS, como gestor de elasticidade embora não permita a explícita configuração dos *thresholds* mínimos e máximos comumente adotados para controle da elasticidade, faz uso destas configurações internamente, juntamente com informações a respeito do nível de ocupação da fila de trabalhos aguardando processamento. Esta estratégia, embora não seja estritamente igual a elasticidade como geralmente definida na literatura, representa uma arquitetura comum à diversos projetos que fazem uso da elasticidade para compor *pipelines* de processamento distribuído e desacoplados, utilizando filas de mensagens em conjunto com ações de elasticidade. Sendo considerado, portanto, adequado para os fins deste trabalho uma vez que facilita a configuração e integração dos diferentes passos de processamento adotados pelo modelo He–lastic.

Ao embutir, em suas informações de contexto, dados a respeito do estado da fila de trabalhos, o serviço Batch permite, assim como o Lambda no FaaS, descartar completamente a necessidade de recursos ociosos enquanto aguarda novas requisições. Esta propriedade é também utilizada pelo modelo He–lastic para prover uma computação ainda mais próxima do modelo pague pelo uso (*pay-per-use*) objetivada pelo conceito de *utility-computing*. Em cenários de elasticidade, mesmo que não haja demanda sendo exercida em um determinado momento, é prática comum manter disponível pelo menos um nível mínimo de recursos computacionais para atender requisições que possam chegar, evitando o risco de rejeitar conexões e aparentar estar indisponível. Embora esta não seja uma preocupação para grandes aplicações de uso contínuo, pode significar maior economia em cenários de baixa demanda ou sazonais, que contemplam longos períodos de ociosidade, por exemplo.

A fim de possibilitar uma comparação justa frente aos cenários de avaliação do jModelTest, os testes da camada de Orquestração de Contêineres do modelo He–lastic contemplaram três cenários variando o parâmetro de número máximo de CPUs utilizadas, uma vez que, conforme anteriormente mencionado, não há a necessidade de configurar *thresholds* ao utilizar o serviço AWS Batch. Desta forma foram estabelecidos os cenários C_{c8} , C_{c16} e C_{c36} com limites máximos de 8, 16 e 36 CPUs, respectivamente. Estes cenários podem ser diretamente comparados com os cenários de avaliação do jModelTest C_{j8} , C_{j16} e C_{j36} para uma análise dos impactos de

Tabela 34 – Tempos de Execução médios dos cenários somente Orquestrador de Contêineres do modelo He-lastic.

Arquivo	Percentual Sobre					
	C_{c8}	C_{c16}	C_{c36}	C_{j8}	C_{j16}	C_{j36}
01-aP6	0:27:00	0:13:37	0:10:03	14613%	13223%	14408%
02-rodents	0:19:44	0:12:06	0:09:06	2083%	2503%	3144%
03-example	0:25:58	0:13:42	0:10:14	1908%	1819%	2305%
04-18S_insects2	0:21:14	0:12:06	0:09:23	1541%	1763%	2519%
05-HIVpol.groupM	0:19:50	0:12:48	0:08:55	556%	669%	754%
06-Hex_EF1a	0:20:54	0:13:35	0:08:28	363%	440%	486%
07-primate-mtDNA	0:23:08	0:14:43	0:10:24	325%	366%	398%
08-HIV_vpu.ref2	0:27:21	0:16:01	0:11:18	216%	240%	292%
09-gusanos16S.mafft	0:34:02	0:19:16	0:13:17	114%	122%	128%
10-Birds	0:22:10	0:13:31	0:09:15	59%	65%	74%
11-gusanosCOI.mafft	0:52:01	0:31:44	0:19:58	104%	116%	123%
12-stamatakis-59	3:25:41	1:54:29	1:05:27	81%	82%	78%
Média Geral				123%	130%	139%

Fonte: Elaborado pelo autor.

performance obtidos através da adoção de elasticidade e do serviço AWS Batch.

Conforme observado previamente na Figura 41, através das linhas pontilhadas de tonalidade azul, que representam os cenários com somente o Orquestrador de Contêineres, é observado um fenômeno interessante no processamento dos arquivos menores. Pela análise da figura é possível visualizar uma reta quase plana que contempla os arquivos de 01-aP6 ao 10-Birds e que se mantém com um tempo de execução entre 10% à 20% do tempo total. Este comportamento também pode ser observado pela análise da Tabela 32, que mostra em valores absolutos as medianas dos tempos de execução observados para os cenários aqui analisados, onde é possível perceber que, na média, os tempos de execução dos arquivos de 01 à 10 do cenário C_{c8} giram em torno de 24 minutos, no cenário C_{c16} em 14 minutos e, finalmente, no cenário C_{c36} giram em torno de 10 minutos.

A existência deste limiar mínimo sobre o tempo de execução de uma parcela significativa dos arquivos do *dataset* indica um alto custo inicial para a adoção da elasticidade via Orquestrador de Contêineres. Este custo inicial também dá indícios de que a utilização desta camada de processamento não é recomendada para os menores arquivos do *dataset*, onde o tempo de execução acaba sendo dominado pelo *overhead* oriundo da utilização da elasticidade. No caso do arquivo 01-aP6, este impacto se traduz em uma execução aproximadamente 140 vezes mais lenta ao longo dos três cenários comparados, um valor definitivamente inaceitável para o usuário

da aplicação que passaria a esperar praticamente meia hora por um resultado que anteriormente obteria em questão de segundos.

Como pode ser visto na [Tabela 34](#), embora o caso do arquivo 01-aP6 seja um extremo, os impactos se estendem de maneira significativa até o arquivo 08-HIV_vpu.ref2 que leva pelo menos o dobro do tempo quando comparado à execução com paralelismo em *threads* do jModelTest. Contudo, os arquivos de 09 em diante apresentam tempos de execução aceitáveis quando comparados a abordagem de base do jModelTest, podendo ser adotados sem grande impacto para o usuário e resultando até mesmo em alguns ganhos de performance na casa de 65% para o arquivo 10-Birds e 80% para o 12-stamatakis-59. De qualquer forma, na média geral, a adoção de uma estratégia baseada apenas na Orquestração de Contêineres para o modelo He–lastic resultaria em uma perda de desempenho que gira em torno de 30% entre todos os três cenários avaliados.

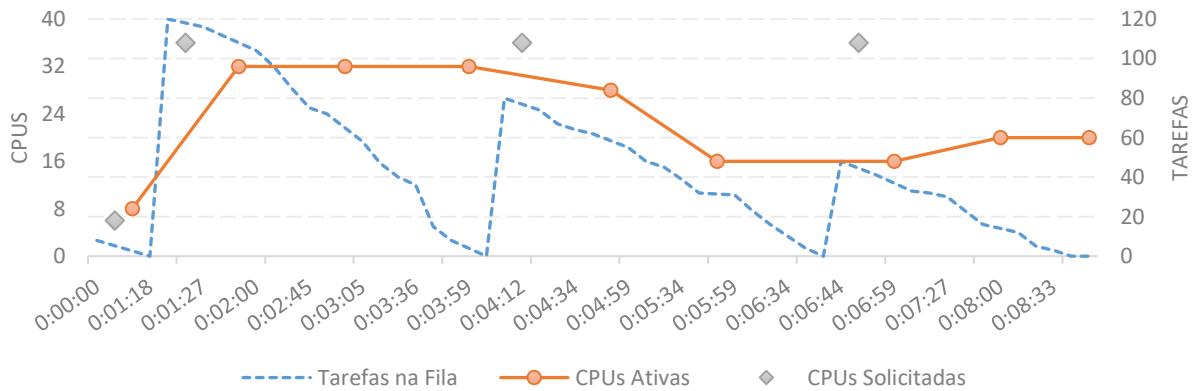
O impacto de apenas 30% no geral, quando existem tantas ocorrências de *overhead* acima de três vezes o tempo de execução, se dá por uma característica da distribuição da computação entre os arquivos do *dataset*, onde o arquivo 12-stamatakis-59 contabiliza, sozinho, aproximadamente 40% de todo o tempo de execução dos cenários, exercendo uma grande influência sobre a média final. Excluindo o arquivo 12-stamatakis-59 do cálculo, os tempos de execução baseados somente na estratégia de Orquestração de Contêineres levariam, em média, o dobro do tempo observado pelos cenários análogos do jModelTest com paralelismo via *threads*.

Os resultados observados, onde o impacto se concentra nos arquivos menores e eventualmente se dissipava conforme crescem os dados, se convertendo até mesmo em modestos ganhos de performance, corroboram para a estratégia de duas camadas proposta pelo modelo He–lastic. Em seu modo completo de funcionamento a camada FaaS fica responsável por absorver os arquivos de menor duração enquanto a camada baseada em Orquestração de Contêineres é ativada somente em casos onde os limites estabelecidos para a camada FaaS não permitem o completo processamento das requisições. O fato de que, no caso do jModelTest e do protótipo implementado, somente uma parcela das avaliações dos sistemas de substituição molecular serão processadas pela camada de Contêineres ajuda a reduzir ainda mais o impacto que ela pode ter na performance quando comparada com o jModelTest sem modificações.

No que tange ao comportamento de elasticidade da camada de Orquestração de Contêineres, é possível inspecioná-lo graficamente através de medições realizadas durante as execuções. Os registros de monitoramento coletados pelo modelo He–lastic, assim como os dados do serviço CloudWatch do provedor de computação em nuvem AWS, permitem observar as ações tomadas pelo motor de elasticidade, ajudando a entender os motivos e as circunstâncias que causaram tais ações.

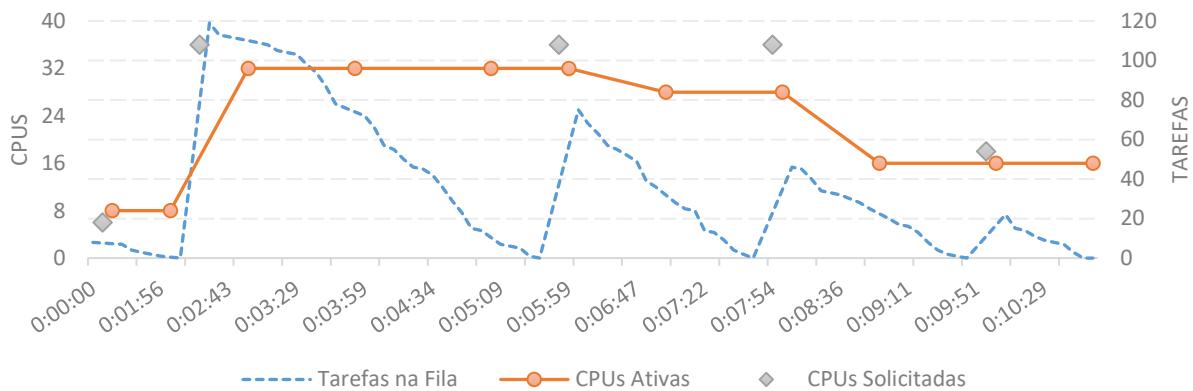
Como pode ser visto no conjunto de Figuras [42](#), [43](#), [44](#), [45](#), [46](#) e [47](#), o funcionamento do motor de elasticidade do serviço AWS Batch costuma seguir de maneira fiel, embora com algum atraso, a curva de demanda, exemplificado, por exemplo, na [Figura 42](#) e [Figura 43](#). Estas duas imagens representam a visão geral do que foi observado pelo autor durante o acompanhamento

Figura 42 – Comportamento de elasticidade para uma execução do arquivo 02-rodents no cenário C_{c36} .



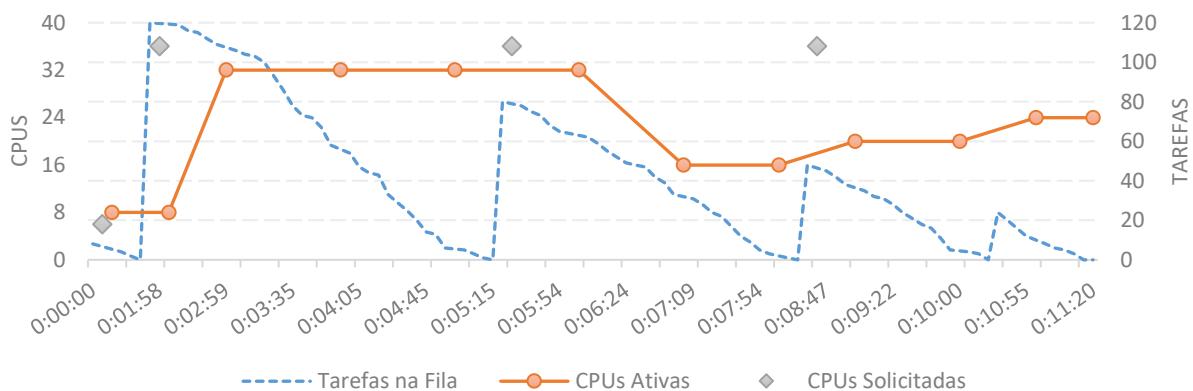
Fonte: Elaborado pelo autor.

Figura 43 – Comportamento de elasticidade para uma execução do arquivo 08-HIV_vpu.ref2 no cenário C_{c36} .



Fonte: Elaborado pelo autor.

Figura 44 – Comportamento de elasticidade contra-intuitivo para uma execução do arquivo 08-HIV_vpu.ref2 no cenário C_{c36} .



Fonte: Elaborado pelo autor.

da execução dos cenários de testes, onde geralmente o motor de elasticidade reage às variações na demanda com um *delay* de até um ou dois minutos. Os pontos demarcados com um losango representam ações preparatórias de elasticidade sugeridas pelo modelo He–lastic, conforme abordado na [Subseção 5.4.4](#) onde, por meio de chamadas de API é informado um número sugerido de CPUs a alocar para o processamento. Contudo, como pode ser observado nas imagens, nem sempre o motor de elasticidade do serviço acata a estas sugestões, o que é considerado correto pelo autor uma vez que esta é apenas uma sugestão oriunda de uma heurística.

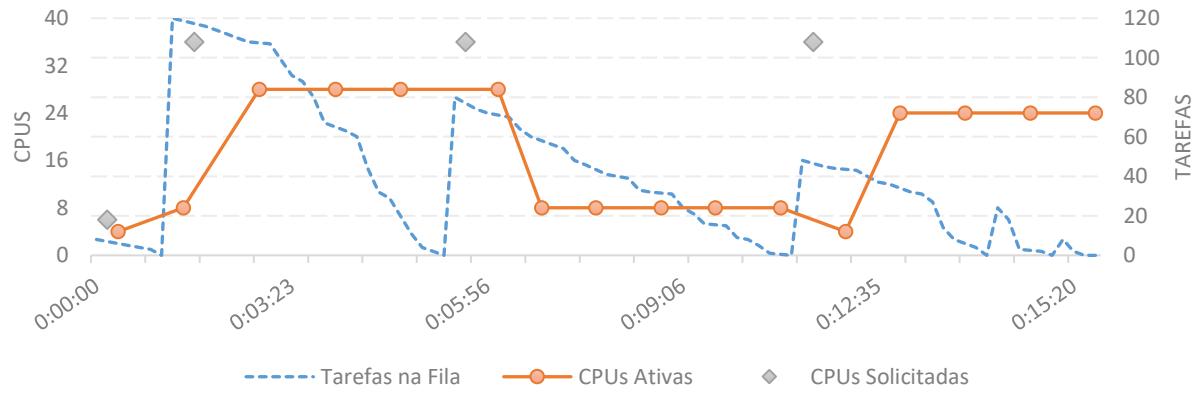
Como resultado da demanda em formato dente de serra, originada pela estratégia de *Clustering Search* adotada pelo jModelTest, podem ocorrer instâncias de comportamento contraintuitivo das ações de elasticidade, como exemplificado pela dupla de [Figura 44](#) e [Figura 45](#). Nestes casos existe uma retomada na quantidade de recursos alocados próximo ao final do processamento, geralmente originada por sugestão da heurística ou, na maioria dos casos, devido ao início de uma nova etapa de processamento que aumenta de maneira abrupta a quantidade de tarefas em espera na fila de processamento. Como resultado deste repentino aumento nos recursos disponíveis, é possível observar um nível elevado de ociosidade nas VMs alocadas para atender estas requisições, um fenômeno que é agravado nos arquivos menores.

Em execuções do arquivo 01-aP6, por exemplo, é comum observar que o serviço Batch, responsável pela camada de Orquestração de Contêineres, tome mais tempo para distribuir as requisições entre as VMs disponíveis do que o tempo necessário para executar um dos cálculos de adequação dos sistemas de substituição molecular. Este comportamento é oriundo do próprio serviço AWS Batch, de forma que não há parâmetro disponível para configurá-lo, e se manifesta através do aumento na ociosidade média dos recursos disponíveis, constituindo um cenário que deve ser evitado por seus usuários. No caso do modelo He–lastic este fenômeno perde importância quando são utilizadas as duas camadas de elasticidade, haja vista que a camada FaaS irá absorver as tarefas mais curtas e evitar que este comportamento se manifeste.

Contudo, para afirmar com segurança que esta é uma característica do serviço utilizado como manipulador da camada de Orquestração de Contêineres ao invés de uma característica do modelo, seriam necessários mais cenários de testes com uma abordagem alternativa para o tratamento desta camada. No entanto a experiência do autor sugere que este comportamento tem origem, de fato, na implementação interna do serviço AWS Batch. Contudo, indiferente da origem deste comportamento, sua ocorrência perde relevância a medida que aumentam os tamanhos dos arquivos e as durações de cada requisição de processamento, reduzindo também seu impacto no desempenho do modelo He–lastic.

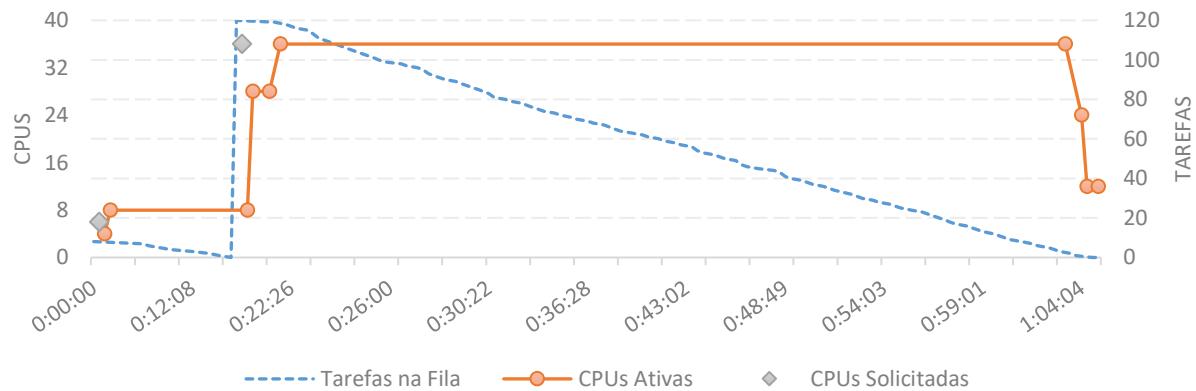
Por fim, as Figuras [46](#) e [47](#) tem por objetivo mostrar a progressividade das ações de elasticidade em um arquivo de maior duração, como o caso do 12-stamatakis-59. Nesta execução, o gerenciador de elasticidade inicialmente lança mão de quatro *cores* de processamento que, quase imediatamente, reconhece como insuficientes, dobrando a alocação para que cada uma das oito tarefas iniciais seja processada por um *core* distinto. Assim que se inicia a segunda etapa do processamento baseado em *Clustering Search* mais uma ação de elasticidade é executada,

Figura 45 – Comportamento de elasticidade contra-intuitivo para uma execução do arquivo 07-primate-mtDNA no cenário C_{c36} .



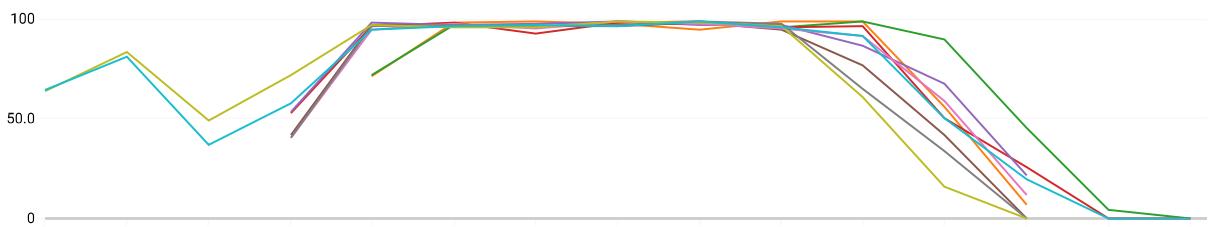
Fonte: Elaborado pelo autor.

Figura 46 – Comportamento de elasticidade para uma execução do arquivo 12-stamatakis-59 no cenário C_{c36} .



Fonte: Elaborado pelo autor.

Figura 47 – Consumo percentual de CPU para cada VM envolvida na execução do arquivo 12-stamatakis-59 no cenário C_{c36} , onde cada linha representa uma VM ativa com quatro cores de processamento.



Fonte: Capturado pelo autor a partir do serviço CloudWatch do provedor AWS.

trazendo o número de *cores* disponível para 28 e, alguns minutos depois, para o teto configurado de 36.

Ao final do processamento, conforme vai aumentando a ociosidade das VMs alocadas o gestor de elasticidade inicia as reduções baixando para 24 *cores* e, logo em seguida, para 12 *cores* distribuídos em 3 VMs. Por meio do monitoramento do arquivo 12-stamatakis-59 é possível perceber a relativa cautela com que o gerenciador de elasticidade executa suas ações, deixando um intervalo de tempo observado na casa dos cinco a dez minutos antes de efetivamente desligar uma VM. Este comportamento pode ser explicado pela existência de alguma tarefa ainda em andamento em um dos *cores* da VM, uma vez que o gerenciador só pode remover o recurso com segurança depois que todas as tarefas em execução na VM estejam concluídas.

Neste caso o gerenciador fica refém de tarefas atrasadas ou que recém foram alocadas, uma situação que se agrava conforme aumentam o número de *cores* disponíveis na VM, o que aumenta também a probabilidade de haver tarefas ainda em andamento, apesar de boa parte da máquina estar ociosa. O conservadorismo na hora de devolver recursos computacionais também pode ser explicado por uma possível política de redução de *thrashing* adotada pelo serviço Batch, o que ajuda a reduzir o impacto de *setup* de novos recursos. Embora não haja menção oficial nas documentações a respeito de uma política de redução de *thrashing*, é seguro assumir que um mecanismo análogo esteja em voga, pois é de interesse do usuário.

6.3.3 Cenário Misto

Já os cenários que contemplam a combinação entre as camadas de FaaS e Orquestração de Contêineres, representando a aplicação completa do modelo He–lastic, embora compartilhem a progressividade oriunda dos arquivos do *dataset*, iniciam de patamar mais baixo que os demais e encerram com uma curva mais suave. Neste caso os dados absolutos endossam os dados relativos, conciliando a [Tabela 32](#) com a [Figura 41](#), uma validação que pode ser realizada através da comparação entre os menores e maiores arquivos do *dataset*. Assim é possível perceber que a estratégia empregada pelo modelo He–lastic é altamente eficaz quando processando em alta granularidade, ou seja, com pequenas tarefas individuais que, no caso desta avaliação, se manifestam por meio dos menores arquivos de alinhamentos de sequências moleculares. No que diz respeito aos arquivos maiores, ou seja, unidades de processamento com menor granularidade, é possível observar uma degradação de performance quando comparados ao cenário que contempla apenas Orquestração de Contêineres, contudo os dados e o gráfico sugerem que este *overhead* diminui conforme crescem os arquivos até eventualmente deixar de ser uma preocupação como pode ser observado no 12-stamatakis-59, o maior arquivo do *dataset*.

Uma observação mais próxima dos cenários mistos do modelo He–lastic (C_{mX}), ainda na [Figura 41](#), mostra uma divergência de comportamento nos arquivos intermediários do *dataset*, que podem ser reunidos em 2 grupos, contendo: a) os cenários que continuam apresentando um comportamento em linha com os cenários baseados em Orquestração de Contêineres (mais

Tabela 35 – Razão dos sistemas de evolução processados pela camada de Orquestração de Contêineres sobre o total por arquivo e cenário observada na avaliação do He–lastic.

	C_{m1}	C_{m2}	C_{m3}	C_{m4}	C_{m5}	C_{m6}	C_{m7}
08-HIV_vpu.ref2	—	1%	—	48%	10%	—	15%
09-gusanos16S.mafft	21%	46%	20%	91%	53%	21%	51%
10-Birds	2%	6%	2%	85%	43%	2%	46%
11-gusanosCOI.mafft	51%	57%	50%	97%	78%	50%	77%
Média Geral	26%	24%	24%	80%	46%	25%	47%

* São apresentados apenas os casos que satisfazem a condição $0\% < Razão < 100\%$.

Fonte: Elaborado pelo autor.

linear e progressivo); e b) os cenários que sofrem um impacto mais agressivo na performance, compostos por (C_{m4} , C_{m5} e C_{m7}). Para compreender este comportamento é necessário recorrer à Tabela 35 que apresenta a razão entre os sistemas de substituição molecular que foram processados pela camada de Orquestração de Contêineres sobre o total.

Assim, uma razão de 40% indica que, do total de sistemas de substituição molecular testados para um determinado arquivo do dataset, 60% dos sistemas foram processados na camada FaaS e os 40% restantes encaminhados para a segunda camada em função do limite no tempo de execução, sendo efetivamente processados pelo Orquestrador de Contêineres. Através da média geral por cenário é possível perceber que os cenários C_{m4} , C_{m5} e C_{m7} , que apresentaram um aumento mais agressivo no tempo de execução, são os que apresentam maior razão de sistemas processados na camada de Orquestração de Contêineres, com valores na faixa de 80% para o cenário C_{m4} e 45% para os cenários C_{m5} e C_{m7} . Os demais cenários se mantém na faixa de 25% dos sistemas processados pelo Orquestrador de Contêineres, resultando em uma curva mais suave conforme observado na Figura 41.

Por meio da análise conjunta sobre os dados da Tabela 32, Figura 41 e Tabela 35 é possível estabelecer que a razão entre os arquivos processados pela camada FaaS e pela camada de Orquestração de Contêineres exerce uma significativa influência no tempo total das execuções quando se utiliza o modelo He–lastic. Também é possível perceber, integrando na análise os dados da Tabela 21, que existe grande sensibilidade ao tempo limite de execução estabelecido para a camada FaaS, onde os arquivos processados pelos cenários C_{m1} , C_{m3} e C_{m6} apresentam os menores níveis de arquivos processados na camada de Orquestração de Contêineres. Contudo, no que diz respeito ao tempo total de execução, a comparação entre os cenários C_{m3} e C_{m4} mostra que através da redução no tempo limite da camada FaaS para 1/4 (de 60s para 15s), enquanto aumenta os recursos do Orquestrador de Contêineres em 2,25x (de 16 para 36), é possível obter melhores tempos de execução, conforme provam os resultados dos arquivos 09-gusanos16S.mafft, 11-gusanosCOI.mafft, e 12-stamatakis-59.

Estes dados sugerem a existência de um faixa do espaço de possíveis configurações do

modelo He–lastic que necessita de uma escolha mais criteriosa para os seus valores, onde uma análise empírica pode ser uma aliada no ajuste fino destas configurações. Embora não seja desejável na formulação de qualquer modelo, a aplicação de estratégias empíricas também é adotada por trabalhos como os de [Shankar et al. \(2018\)](#) e [Jonas et al. \(2017\)](#) devido à dificuldade em formalizar o comportamento dinâmico que se origina da combinação entre os parâmetros de Potência e Tempo Limite na camada FaaS sendo, no caso do modelo He–lastic, agravada pela existência de um terceiro fator, a Quantidade Limite de CPUs na camada de Orquestração de Contêineres. A falta de modelos formais que sejam capazes de representar este comportamento dinâmico evidencia a necessidade de trabalhos futuros visando este objetivo, conclusão também obtida por [Shankar et al. \(2018\)](#).

Como estratégia para avaliar o grau de *overhead* gerado pelo modelo He–lastic sobre a execução do software jModelTest, uma medida de variação no tempo de execução foi adotada, comparando os tempos médios das execuções por arquivo dos cenários C_{mX} versus os cenários C_{jX} com a mesma quantidade de CPUs. Desta forma, o cenário C_{m1} é comparado com C_{j8} , os cenários C_{m2} e C_{m3} são comparados com C_{j16} e os demais cenários são comparados com C_{j36} , como pode ser visto pela análise das Tabelas 20 e 21. Embora a comparação por meio da quantidade de CPUs seja intuitiva, ela se mostra incompleta uma vez que, do lado do jModelTest a quantidade de CPUs mapeia a disponibilidade destes recursos ao longo da execução, enquanto no modelo He–lastic ela mapeia a quantidade máxima do recurso que pode ou não ser alocado conforme o comportamento de elasticidade. Além disso, não é possível estabelecer um paralelo entre as duas abordagens comparadas no que diz respeito à alocação de recursos da camada FaaS, haja vista que não há nenhuma semelhança entre os conceitos de Potência e Tempo Limite de execução, parâmetros fundamentais para o modelo He–lastic e a camada FaaS. Contudo, mesmo que a comparação dos cenários por meio do número de CPUs não seja ideal, ela apresenta–se como a alternativa mais próxima para uma comparação justa entre as abordagens do modelo He–lastic e do jModelTest.

Por meio dos dados apresentados na [Tabela 36](#) é possível perceber que, na maioria dos arquivos do *dataset* de testes, a abordagem do modelo He–lastic se mostra favorável quando avaliada em termos do tempo total de execução. Observando as diferenças nos tempos de execução ao longo de todos os cenários avaliados é possível perceber que os maiores ganhos estão concentrados nos arquivos de tamanho pequeno a intermediários do *dataset*, com destaque para os arquivos 05-HIVpol.groupM e 06-Hex_EF1a, onde foi possível obter tempos de execução que levaram menos de 25% do tempo dos cenários base do jModelTest. O arquivo 01-aP6 pode ser incluído nesta classificação pois, embora sua variação percentual seja significativa, ela não apresenta grande relevância quando avaliada em termos absolutos, devido ao pequeno tamanho do arquivo e consequente curta duração de processamento.

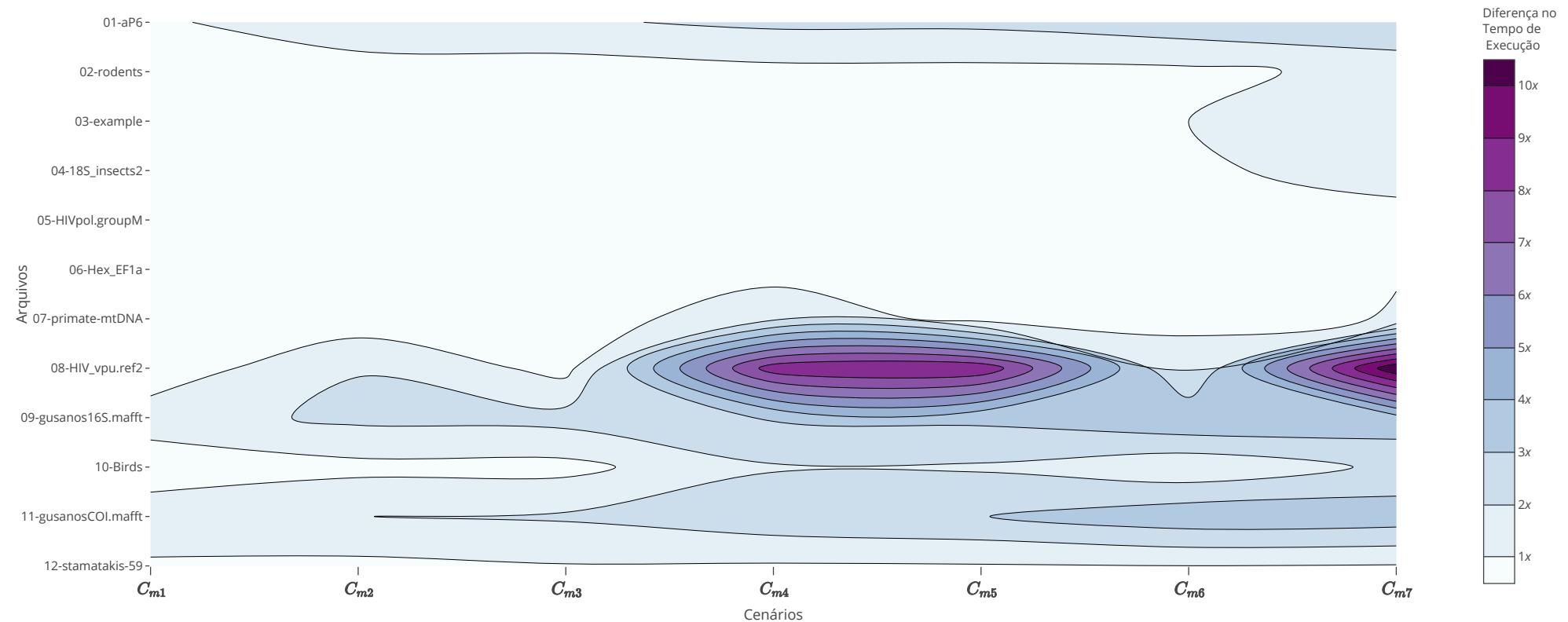
Nos arquivos considerados maiores os resultados são de interpretação mais complexa, uma vez que existe um claro ganho de performance, embora pequeno, no caso do arquivo 12-stamatakis-59, que não é compartilhado com os arquivos. A região que contempla os arquivos

Tabela 36 – Diferenças nos tempos de execução entre os cenários de avaliação do modelo He-lastic com FaaS e Orquestração de Contêineres versus os cenários do jModelTest, com os melhores e piores resultados destacados em verde e vermelho, respectivamente.

Arquivo	C_{m1}		C_{m2}		C_{m3}		C_{m4}		C_{m5}		C_{m6}		C_{m7}	
	Razão	Delta	Razão	Delta	Razão	Delta	Razão	Delta	Razão	Delta	Razão	Delta	Razão	Delta
01-aP6	81%	-0:00:02	175%	0:00:05	188%	0:00:05	220%	0:00:05	221%	0:00:05	263%	0:00:07	297%	0:00:08
02-rodents	22%	-0:00:44	47%	-0:00:15	49%	-0:00:15	73%	-0:00:05	73%	-0:00:05	79%	-0:00:04	126%	0:00:04
03-example	30%	-0:00:57	55%	-0:00:20	58%	-0:00:19	83%	-0:00:04	92%	-0:00:02	100%	-0:00:00	163%	0:00:17
04-18S_insects2	24%	-0:01:03	41%	-0:00:24	47%	-0:00:22	77%	-0:00:05	78%	-0:00:05	89%	-0:00:02	128%	0:00:06
05-HIVpol.groupM	13%	-0:03:06	25%	-0:01:27	28%	-0:01:23	41%	-0:00:42	40%	-0:00:42	45%	-0:00:39	76%	-0:00:17
06-Hex_EF1a	17%	-0:04:46	32%	-0:02:06	30%	-0:02:09	55%	-0:00:47	57%	-0:00:44	58%	-0:00:44	96%	-0:00:04
07-primate-mtDNA	23%	-0:05:31	40%	-0:02:26	39%	-0:02:27	181%	0:02:07	59%	-0:01:05	67%	-0:00:52	105%	0:00:08
08-HIV_vpu.ref2	35%	-0:08:17	195%	0:06:22	67%	-0:02:13	855%	0:29:15	878%	0:30:07	163%	0:02:25	1084%	0:38:06
09-gusanos16S.mafft	151%	0:15:12	223%	0:19:25	237%	0:21:38	308%	0:21:35	322%	0:23:04	395%	0:30:39	362%	0:27:17
10-Birds	38%	-0:23:25	73%	-0:05:39	71%	-0:06:02	192%	0:11:30	189%	0:11:02	124%	0:02:57	220%	0:14:59
11-gusanosCOI.mafft	160%	0:29:58	199%	0:27:11	212%	0:30:36	268%	0:27:21	297%	0:32:05	367%	0:43:24	356%	0:41:37
12-stamatakis-59	87%	-0:33:38	76%	-0:32:49	95%	-0:06:46	90%	-0:08:01	93%	-0:06:06	99%	-0:00:46	94%	-0:04:53
Média Geral	-0:02:50		-0:00:26		0:02:21		0:06:03		0:07:25		0:10:00		0:09:47	
Máxima	0:31:51		0:30:02		0:32:11		0:33:40		0:36:54		0:52:25		0:42:50	

Fonte: Elaborado pelo autor.

Figura 48 – Mapa de calor da diferença nos tempos de execução entre os cenários de avaliação do modelo He-lastic com FaaS e Orquestração de Contêineres versus os cenários do jModelTest, onde cores mais fortes representam maiores diferenças.



Fonte: Elaborado pelo autor.

do intervalo 07 à 11 merece especial atenção devido aos resultados mistos obtidos, conforme o cenário executado. Chamam a atenção nesta análise os resultados obtidos pelo arquivo 08-HIV_vpu.ref2 nos cenários C_{m7} , C_{m5} e C_{m4} , com variações percentuais de 1084%, 878% e 855%, respectivamente.

Para auxiliar na compreensão destes resultados a Figura 48 apresenta um mapa de calor que abrange os dados da Tabela 36, possibilitando a análise visual dos resultados obtidos. Com a ajuda desta figura é possível perceber que os resultados do arquivo 08-HIV_vpu.ref2 são localizados, afetando de maneira mais grave apenas este arquivo nos cenários previamente mencionados. Nos demais arquivos e cenários a variação observada se concentra na faixa entre 100% à 400%, o que, embora não seja encorajador, não causa o mesmo nível de espanto dos valores observados para o arquivo 08.

Uma nova análise da Tabela 35 resulta em uma forte correlação com a Figura 48, fornecendo uma explicação para o formato da região que sofreu maior impacto de performance em função da adoção do modelo He–lastic. Embora nem todos os arquivos e cenários apresentem correlação tão intensa quanto o 10-Birds, por exemplo, a Tabela 35 fornece um importante *insight* sobre o motivo desta degradação de performance, nomeadamente a taxa de requisições que foram atendidas pela segunda camada do modelo. No caso do protótipo que toma por base a implementação do jModelTest, isto se traduz na razão dos sistemas de substituição molecular cujo cálculo não foi possível de realizar na camada FaaS, necessitando a transição para a camada de Orquestração de Contêineres. Uma ressalva importante na análise da Tabela 35 deve ser feita quanto ao arquivo 12-stamatakis-59 que, embora não conste na tabela, tem a totalidade de suas requisições, ou seja, 100%, atendidas pela camada de Orquestração de Contêineres e, por consequência, sofrendo o mesmo tipo de *overhead* que os demais arquivos e cenários.

Como já previamente abordado na Subseção 5.4.4, uma das grandes dificuldades encontradas na avaliação do protótipo se deu em função da incapacidade de influenciar a quantidade de *retries* realizados pela camada FaaS quando uma requisição ultrapassa o tempo limite de execução. Especula-se que os *retries* impostos pelo serviço Lambda do provedor AWS estejam contribuindo de maneira significativa para o impacto de performance observado. A proporcionalidade do tempo de espera entre os *retries*, que segue a política de *backoff* exponencial, também é visível nos dados da Tabela 36, onde o cenário C_{m7} que tem *timeout* configurado de 60 segundos é que apresenta maior impacto de performance, seguido dos cenários C_{m5} e C_{m4} , com *timeouts* de 30 e 15 segundos, respectivamente.

Outro fator que gera impacto é o *overhead* oriundo da camada de Orquestração de Contêineres, que se mostrou, de acordo com resultados prévios, mais conservador nas ações de elasticidade, até mesmo deixando de utilizar na totalidade os recursos disponibilizados. Assim como observado no cenário de avaliação baseado somente em Orquestração de Contêineres, por meio das Figuras 42, 43, 44 e 45, nos cenários de avaliação do modelo He–lastic com suas duas camadas é recorrente o comportamento conservador na alocação de recursos por parte do gerenciador de elasticidade. Nestes casos, mesmo que os parâmetros do modelo permitam utilizar

até 36 *cores* de processamento, como no cenário $C_{m:3}$ por exemplo, o gestor de elasticidade do serviço AWS Batch aloca em torno de 20 para efetuar a execução, o que causa um efeito negativo no tempo de execução, embora positivo do ponto de vista do custo. Mesmo que seja possível justificar este comportamento, que pode ser adotado para mitigar a ocorrência de *thrashing* de recursos, nos cenários de processamento compartilhado entre as duas camadas do modelo He–lastic há um perceptível impacto no desempenho em termos do tempo de processamento, como pode ser observado na Figura 48.

Há, ainda, o agravante da estratégia de *Clustering Search* adotada pelo jModelTest, que faz com que os impactos sejam ainda mais agudos quando o *overhead* da camada de Orquestração de Contêineres inicia próximo do final de uma etapa, o que faz com que todas as demais etapas fiquem represadas até que as últimas requisições sejam atendidas. Ilustra este cenário uma situação onde o último dos sistemas de substituição molecular a ser avaliado em uma etapa do processamento excede os limites da camada FaaS e deve, portanto, ser encaminhado para o Orquestrador de Contêineres, o que faz com que todo o restante do processamento deva necessariamente esperar pela conclusão deste último atrasado, podendo, inclusive, causar um efeito bola de neve caso a situação se repita nas próximas etapas.

Todavia é relevante ponderar também sobre os resultados obtidos pelo arquivo 12-stamatakis-59 que obteve até mesmos ganhos marginais de performance ao longo de todos os cenários de avaliação mesmo enfrentando todos os *delays* e dificuldades mencionadas acima. Neste arquivo, nem o impacto dos *retries* impostos pelo serviço AWS Lambda, nem o impacto da elasticidade do serviço AWS Batch foram capazes de se traduzir em degradação de performance quando avaliado sob o critério do tempo de execução. Assim é possível concluir que tais *overheads* se tornam cada vez menos relevantes conforme cresce o tamanho do arquivo (e o tamanho do grão de elasticidade). Embora o conjunto de arquivos de teste não contemple arquivos maiores do que este, o autor considera seguro assumir que este comportamento representa uma tendência, embora haja carência de evidências.

Também carece investigação os resultados encontrados no que tange ao arquivo 08-HIV_vpu.ref2 em busca de uma conclusão autoritativa acerca do alto impacto de performance encontrado no seu processamento. Como já mencionado anteriormente, parte da resposta pode ser atribuída à necessidade de processar uma parcela das requisições na camada de Orquestração de Contêineres. Contudo este argumento não é capaz de explicar o nível tão elevado de *overhead* quando comparado aos demais arquivos que também tiveram parte das requisições atendidas pela segunda camada do modelo He–lastic. Uma segundo fator pode estar associado à quantidade de etapas de processamento que o arquivo é submetido na estratégia de *Clustering Search* do jModelTest, onde o 08-HIV_vpu.ref2, conforme dados das Tabelas 15 e 18 é submetido à cinco etapas de processamento devido aos seus 280 sistemas de substituição. Dentre os arquivos que passam pela camada de Orquestrador de contêineres, o 08-HIV_vpu.ref2 é o que contempla maior número de etapas, conforme mostra o levantamento a seguir: 08-HIV_vpu.ref2: 5 etapas; 09-gusanos16S.mafft: 4 etapas; 10-Birds: 3 etapas; 11-gusanosCOI.mafft: 4 etapas; e

Tabela 37 – Consolidação dos resultados obtidos nas execuções dos cenários mistos do modelo He–lastic em comparação com os resultados das execuções do jModelTest.

Cenário de Referência		Cenário He–lastic		Diferença	
Código	Duração Média	Código	Duração Média	Média	Percentual
C_{f8}	0:32:01	C_{m1}	0:31:08	-0:00:53	-3%
C_{j16}	0:17:31	C_{m2}	0:15:46	-0:01:45	-10%
		C_{m3}	0:19:36	0:02:05	12%
		C_{m4}	0:15:45	0:05:12	49%
C_{j36}	0:10:33	C_{m5}	0:18:43	0:08:10	77%
		C_{m6}	0:26:41	0:16:08	153%
		C_{m7}	0:20:54	0:10:21	98%

Fonte: Elaborado pelo autor.

12-stamatakis-59: 2 etapas.

Por fim, retornando aos resultados da Tabela 36, foram destacadas duas métricas referentes às diferenças nos tempos de execução, sendo uma delas a média geral, que agrupa todos os dados coletados em termos das suas diferenças nos tempos de execução e fornece um indicador totalizado para o tempo de execução de cada cenário. Assim como a máxima, que demonstra o maior tempo de execução observado dentre todos os dados que compõem os resultados do cenário. Por meio destas duas métricas é possível determinar um dos cenários como o melhor, de acordo com os critérios do usuário ou o ambiente onde será utilizado o modelo He–lastic. Por exemplo, em um ambiente caracterizado por alto volume de requisições paralelas e usuários concorrendo por recursos, pode se mais apropriado adotar uma estratégia que busque minimizar o tempo máximo de espera, reduzindo o efeito conhecido como cauda longa. Para este caso o melhor cenário seria o C_{m2} , embora seguido de perto pelos cenários C_{m1} , C_{m3} e C_{m4} , com a pior escolha, segundo este critério, sendo o cenário C_{m6} .

Contudo, de maneira geral, é possível estabelecer o cenário C_{m1} como o melhor dentre os cenários que fazem uso das duas camadas de elasticidade do modelo He–lastic, onde este apresenta um desempenho levemente superior ao cenário de referência do jModelTest com 8 CPUs disponíveis, C_{f8} . Observando os cenários que disponibilizam 16 CPUs para o jModelTest é possível elencar o C_{m2} como o melhor deles, novamente apresentando resultado em linha com os tempos de execução observados no cenário de referência. Por fim, no comparativo com os cenários que poderiam fazer uso de até 36 CPUs no jModelTest, o cenário C_{m4} é escolhido como o mais apropriado, resultando, infelizmente, em uma ineficiência média frente ao cenário base C_{f36} de seis minutos.

Os resultados gerais podem ser visualizados na Tabela 37 que apresenta, por meio das médias

dos tempos de execução, o panorama comparativo entre as execuções do modelo He–lastic com suas duas camadas de elasticidade versus as execuções do jModelTest. A escolha das médias, embora possa parecer contra–intuitiva, foi necessária em função da diferença na quantidade de execuções por arquivo e cenário, sendo utilizada neste contexto como meio para obter uma comparação justa entre as duas abordagens. Desta forma, em termos do tempo médio de execução por arquivo, é possível estabelecer que a abordagem do modelo proposto é vantajosa quando comparada a execuções do jModelTest com recursos fixos de 8 e 16 CPUs, conforme dados dos cenários C_{j8} e C_{j16} . Contudo, este resultado não se sustenta quando os recursos disponíveis para o jModelTest são aumentados para 36 CPUs no cenário C_{j36} , onde há uma perda de eficiência da abordagem com elasticidade, representada pelo modelo proposto, frente a abordagem de recursos fixos do jModelTest.

6.4 Avaliação de Custos

Uma vez compreendido o comportamento do modelo He–lastic no que diz respeito a performance medida pelo tempo de execução, o presente trabalho prossegue para a análise de custo. Todos os dados apresentados aqui tem como base as métricas de avaliação estabelecidas na Seção 4.5, contudo, para fins de apresentação, somente serão demonstrados os valores de custo Financeiro, uma vez que esta é a única métrica capaz de harmonizar os custos do modelo de alocação de recursos fixos, do jModelTest puro, da camada FaaS e da camada de Orquestração de Contêineres utilizadas pelo modelo proposto.

Adicionalmente, os cenários utilizados para comparação de custos foram reduzidos para somente os que contemplam a execução do modelo He–lastic na sua abordagem mista, ou seja, com ambas as camadas de elasticidade habilitadas. Os identificadores de tais cenários seguem o formato C_{mX} e foram escolhidos para esta análise aqueles que apresentaram melhor desempenho em termos dos tempos de execução conforme a quantidade de recursos disponível para alocação. Como já discutido, embora a quantidade de recursos não seja um indicativo confiável de semelhança entre os cenários, ele se mostrou o único viável na comparação entre jModelTest e o modelo proposto.

O critério de seleção dos cenários que participam da análise comparativa de custos se baseou nos dados das Tabelas 36 e 37, escolhendo o que obteve melhor desempenho quando agrupados em termos da quantidade de CPUs disponíveis para alocação, conforme determinado na definição dos cenários de avaliação para o modelo He–lastic na Tabela 21. Como resultado, obtêm-se os cenários C_{m1} , C_{m2} e C_{m4} como participantes da análise de custos. Não foram selecionados para esta etapa de análise os cenários dedicados a uma única das camadas de elasticidade pois a intenção é avaliar de maneira consolidada as características de desempenho, seja em termos do tempo de execução ou de custo financeiro, do modelo proposto. Além disso, no caso do cenário somente FaaS (C_{f0}), não seria possível obter um resultado conclusivo, haja vista que a camada FaaS é incapaz de processar todos os arquivos contidos no *dataset* em função dos limites

estabelecidos pelo próprio modelo computacional.

São apresentados na [Tabela 38](#) os principais indicadores que afetam na determinação do custo financeiro para os cenários do modelo **H_e-lastic**, onde, além dos cenários selecionados para análise, foram incluídos os cenários exclusivos de avaliação de uma das camadas de elasticidade para referência. Todos os valores do corpo da tabela são summarizados através das suas médias, com a exceção dos dados totalizados que realizam o somatório dos dados de duração e custo financeiro. O indicador de CPUs, presente na tabela, representa, em contraste com o parâmetro de configuração do modelo, a quantidade efetiva de unidades de processamento utilizadas durante a avaliação da dupla arquivo–cenário.

O termo CPU, neste caso, adquire um sentido mais amplo, contemplando tanto o conceito clássico de *cores* um microprocessador, quanto os contêineres ou unidades de processamento da camada FaaS, o que explica a grande quantidade reportada na tabela. Desta forma, as 127 CPUs utilizadas em média no cenário C_{m4} correspondem a um valor $CPU = CPU_{faas} + CPU_{contêineres}$, onde a quantidade de CPUs utilizadas na camada de Orquestração de Contêineres é um valor inteiro compreendido no intervalo $0 \leq x \leq CPU_{max}$, sendo o limite superior o valor do parâmetro do modelo.

Ainda no que diz respeito a CPUs utilizadas é possível confirmar o fenômeno mencionado na [Subseção 6.3.1](#), onde o número de Processos / CPUs tende a se aproximar do nível máximo de paralelismo estabelecido pelo mecanismo de *Clustering Search* adotado pelo jModelTest. Contudo, ainda mais interessante é a análise do cenário C_{c36} , correspondente ao modelo **H_e-lastic** utilizando apenas a camada de Orquestração de Contêineres, que apresenta mais uma evidência a respeito do comportamento conservador do gerenciador de elasticidade do serviço AWS Batch, utilizado pela camada. Mesmo configurado para utilizar até 36 *cores* de processamento, o gerenciador de elasticidade adota, neste cenário, um comportamento bastante conservador ao alocar menos de cinco CPUs para os arquivos 04-18S_insects2 e 12-stamatakis-59, o que representa menos de 15% do valor permitido. Embora este comportamento não seja nocivo para o custo financeiro, uma vez que o custo acompanha a quantidade de recursos utilizados ao longo do tempo, a baixa alocação efetiva traz impactos perceptíveis na duração do processamento, e provavelmente está ocorrendo também nos demais cenários, embora mascarado pela grande quantidade de CPUs da camada FaaS.

Por fim, ainda no contexto da [Tabela 38](#), as colunas denominadas Razão apresentam os mesmos dados contidos na [Tabela 35](#), ilustrando a divisão de tarefas entre as camadas do modelo **H_e-lastic**. Assim como nos casos anteriores, esta razão representa a quantidade de sistemas de substituição molecular, ou as tarefas individuais, que foram processadas pela camada de Orquestração de Contêineres sobre o total das tarefas executadas no processamento de cada arquivo do *dataset*. No contexto da [Tabela 38](#) sua presença é relevante para observar o impacto, tanto a nível de custo quanto a nível de tempo de execução, causado pela transição entre camadas. Os arquivos 08-HIV_vpu.ref2 e 09-gusanos16S.mafft no cenário C_{m1} ilustram bem esta situação, onde um aumento na razão para aproximadamente 20%, faz com que o custo financeiro passe de

Tabela 38 – Resultados de performance e custo financeiro das avaliações dos cenários do modelo He–lastic para cada arquivo do *dataset*, contemplando apenas FaaS (C_{f0}), apenas Orquestrador de Contêineres (C_{c36}) e os cenários Mistos com melhor desempenho (C_{m1} , C_{m2} e C_{m4}).

Arquivo	C_{f0}			C_{c36}			C_{m1}			C_{m2}			C_{m4}					
	Duração	CPUs	Financeiro	Duração	CPUs	Financeiro	Duração	CPUs	Razão	Financeiro	Duração	CPUs	Razão	Financeiro	Duração	CPUs	Razão	Financeiro
01-aP6	0:00:09	63	0,000	0:10:02	21	0,240	0:00:09	70	0%	0,000	0:00:11	29	0%	0,000	0:00:09	78	0%	0,000
02-rodents	0:00:13	114	0,010	0:09:06	20	0,190	0:00:13	113	0%	0,010	0:00:14	88	0%	0,007	0:00:13	121	0%	0,008
03-example	0:00:27	118	0,010	0:10:14	21	0,236	0:00:24	115	0%	0,010	0:00:25	114	0%	0,011	0:00:22	114	0%	0,010
04-18S_insects2	0:00:19	117	0,010	0:09:23	20	0,159	0:00:20	102	0%	0,010	0:00:17	118	0%	0,010	0:00:17	116	0%	0,010
05-HIVpol.grou[...]	0:00:30	120	0,020	0:08:55	22	0,208	0:00:28	120	0%	0,020	0:00:28	120	0%	0,020	0:00:29	120	0%	0,020
06-Hex_EF1a	0:00:59	120	0,030	0:08:28	23	0,172	0:00:59	119	0%	0,032	0:00:59	120	0%	0,036	0:00:57	120	0%	0,030
07-primate-mtDNA	0:01:37	120	0,050	0:10:51	21	0,255	0:01:36	120	0%	0,050	0:01:35	121	0%	0,050	0:04:44	121	0%	0,052
08-HIV_vpu.ref2	0:04:41	122	0,150	0:11:18	23	0,324	0:04:31	120	0%	0,153	0:13:01	144	1%	0,185	0:32:45	156	48%	0,454
09-gusanos16S[...]				0:13:17	28	0,388	0:44:12	166	21%	0,546	0:35:21	190	46%	0,540	0:32:28	174	91%	0,581
10-Birds				0:09:15	23	0,242	0:14:31	144	2%	0,218	0:15:28	145	6%	0,254	0:23:19	147	85%	0,578
11-gusanosCOI[...]				0:19:58	25	0,460	1:18:26	200	51%	0,840	0:54:39	174	57%	0,796	0:43:13	166	97%	0,785
12-stamatakis-59				1:05:33	26	1,346	3:41:16	135	100%	1,385	1:47:52	142	100%	1,411	1:15:33	153	100%	1,456
Total [†]	0:08:55	112	0,28	3:06:21	23	4,22	6:07:05	127	14%	3,27	3:50:29	125	18%	3,32	3:34:29	132	35%	3,99

[†] Na linha de totais as informações são apresentadas da seguinte forma: Duração e Financeiro = Soma, CPUs e Razão = Média.

Fonte: Elaborado pelo autor.

Tabela 39 – Resultados de performance e custo financeiro das avaliações dos cenários do jModelTest por arquivo do *dataset*.

Arquivo	C_{j2}		C_{j4}		C_{j8}		C_{j16}		C_{j36}	
	Duração	Financeiro	Duração	Financeiro	Duração	Financeiro	Duração	Financeiro	Duração	Financeiro
01-aP6	0:00:40	0,0010	0:00:21	0,0010	0:00:11	0,0010	0:00:06	0,0012	0:00:04	0,0018
02-rodents	0:03:20	0,0047	0:01:41	0,0048	0:00:57	0,0054	0:00:29	0,0055	0:00:17	0,0074
03-example	0:04:42	0,0067	0:02:23	0,0068	0:01:22	0,0077	0:00:45	0,0085	0:00:27	0,0113
04-18S_insects2	0:04:52	0,0069	0:02:27	0,0069	0:01:23	0,0078	0:00:41	0,0078	0:00:22	0,0095
05-HIVpol.groupM	0:11:58	0,0170	0:06:10	0,0175	0:03:34	0,0202	0:01:55	0,0217	0:01:11	0,0302
06-Hex_EF1a	0:20:12	0,0286	0:10:16	0,0291	0:05:45	0,0326	0:03:05	0,0350	0:01:44	0,0444
07-primate-mtDNA	0:24:23	0,0345	0:12:26	0,0352	0:07:07	0,0403	0:04:01	0,0455	0:02:37	0,0666
08-HIV_vpu.ref2	0:45:38	0,0647	0:22:56	0,0650	0:12:39	0,0717	0:06:41	0,0757	0:03:52	0,0987
09-gusanos16S.mafft	1:44:52	0,1486	0:53:06	0,1504	0:29:46	0,1687	0:15:47	0,1788	0:10:24	0,2651
10-Birds	2:11:50	0,1868	1:07:34	0,1914	0:37:38	0,2133	0:20:53	0,2366	0:12:28	0,3179
11-gusanosCOI.mafft	2:58:49	0,2533	1:31:03	0,2580	0:50:14	0,2847	0:27:22	0,3101	0:16:15	0,4145
12-stamatakis-59	14:51:54	1,2635	7:30:39	1,2768	4:15:20	1,4469	2:19:30	1,5811	1:23:34	2,1310
Total	23:43:11	2,02	12:01:01	2,04	6:45:56	2,30	3:41:15	2,51	2:13:16	3,40

Fonte: Elaborado pelo autor.

\$ 0,153 para \$ 0,546. Como comparação, esta mesma transição avaliada no cenário composto apenas pelo Orquestrador de Contêineres (C_{c36}), passa de \$ 0,324 para \$ 0,388, resultando em um aumento no custo de apenas 16%, em linha com o respectivo aumento no tempo de execução, enquanto na ocorrência de transição entre camadas de elasticidade, o aumento observado no custo é maior que 70% e acompanhado de um aumento de 90% no tempo de execução.

Estes resultados mostram que a ineficiência observada durante a transição entre camadas, em função de fatores previamente mencionados como o obrigatoriedade de *retries* imposto pelo serviço AWS Lambda, assim como possíveis ineficiências ou políticas subjacentes opacas na gestão de elasticidade adotada pelo serviço AWS Batch, é um grave ponto de atenção para o modelo He–lastic. O mapa de calor dos tempos de execução, apresentado na Figura 48, volta a ser relevante aqui para caracterizar este fenômeno como centrado nos pontos de transição entre camadas. No caso das execuções que apresentam uma baixa razão de transição entre camadas, assim como aqueles que nem precisariam passar pela primeira camada, como o caso dos arquivos maiores como o 12-stamatakis-59, o nível de impacto causado pela adoção do modelo proposto é negligenciável. Assim, o resultado final desta análise, evidencia a necessidade de trabalhos futuros para investigar os motivos deste *overhead*, assim como o investimento na concepção de técnicas e métodos capazes de mitigar este impacto.

Os resultados da análise dos custos associados a execução do programa jModelTest, nos cenários de avaliação estabelecidos na Seção 5.1, pode ser visto na Tabela 39, onde constam os resultados médios de tempo de execução e custo financeiro para as duplas arquivo–cenário, assim como os totalizadores por cenário. Um ponto de fundamental atenção é que os resultados ali apresentados contemplam apenas os custos incorridos durante o tempo de execução dos cenários, sendo definitivamente otimistas uma vez que o projeto jModelTest, devido à sua ausência de diretrivas de elasticidade, necessita de um servidor ou *cluster* computacional disponível para atender as solicitações de processamento. Contudo, sua análise permanece relevante ao estabelecer uma referência de eficiência máxima para os resultados do modelo He–lastic, onde é adotada a suposição de que os cenários do jModelTest sejam capazes de funcionar com recursos intermitentes, como é o caso do modelo proposto.

Assim como nas análises do tempo de execução, fica visível nos resultados da Tabela 39 a perda de eficiência do jModelTest conforme aumentam os recursos. Esta perda é atribuída ao método de *Clustering Search* adotado pela aplicação, que divide em etapas o processamento dos sistemas de substituição molecular, fazendo com que haja ociosidade significativa no final de cada etapa conforme aumentam os recursos disponíveis para execução dos cálculos, que são, por natureza, embarrasosamente paralelos. Tal perda também se manifesta na forma de custo financeiro uma vez que, no formato tradicional de alocação de recursos em nuvem, a cobrança se dá pelo tempo que uma VM foi utilizada, desconsiderando o nível de utilização da mesma.

Segundo esta lógica, se o comportamento de escalabilidade do programa jModelTest fosse próximo de linear, o custo observado deveria se manter praticamente constante, uma vez que o aumento nos recursos é compensado, em mesma proporção, pela redução no tempo de execução.

Tabela 40 – Consolidação dos resultados obtidos dentre os cenários selecionados do modelo He–lastic e seus cenários análogos no jModelTest.

CPUs Alocadas	Cenário	Totais		Percentual		Ineficiência He–lastic
		Duração	Custo	Duração	Custo	
8	C_{m1}	6:07:31	3,27	48%	59%	6%
	C_{j8}	6:45:56	2,30	52%	41%	
16	C_{m2}	3:50:29	3,32	51%	57%	8%
	C_{j16}	3:41:15	2,51	49%	43%	
36	C_{m4}	3:34:29	3,99	62%	54%	16%
	C_{j36}	2:13:16	3,40	38%	46%	

Fonte: Elaborado pelo autor.

É possível verificar indícios deste comportamento ótimo de escalabilidade na evolução dos primeiros cenários de teste, onde C_{j2} , C_{j4} e C_{j8} apresentam leves aumentos, que são considerados normais para um programa com boas características de elasticidade. No entanto, a medida que segue aumentando a quantidade de recursos disponíveis, neste caso a CPU por conta da natureza *CPU-bound* do cálculo, tanto o tempo de execução quanto o custo se descolam de uma linha reta e passam a assumir um comportamento que lembra mais uma função polinomial. Assim, o resultado de custo \$ 3,40 para o cenário C_{j36} , representa uma ineficiência de quase 70% sobre o custo esperado de \$ 2,02 caso o programa apresentasse comportamento linear de escalabilidade, reconhecidamente uma tarefa difícil mas possível devido a natureza embaraçosamente paralela do processamento realizado.

Consolidando os resultados obtidos para os cenários de avaliação do jModelTest e do modelo He–lastic, a Tabela 40 apresenta o comparativo dos desempenhos em termos de tempo de execução e custos dos dados coletados. Estão representados, na tabela, os cenários de teste do jModelTest com seus custos mínimos, calculados estritamente em função do tempo de execução, assim como os dados dos cenários com melhor desempenho em termos dos tempos de execução obtidos para o modelo proposto. A comparação entre eles foi realizada através do cálculo do percentual que cada uma das métricas representa sobre a soma dos resultados obtidos pelo cenário do jModelTest e o do He–lastic, onde um resultado de 50% representa o equilíbrio entre as abordagens, consequência de tempos de execução e/ou custos iguais. Adicionalmente, os resultados da tabela são agrupados por número de CPUs alocadas em cada cenário, sendo esta a característica que chega mais próxima de estabelecer um nível de comparação, conforme mencionado anteriormente.

Por fim, na última coluna da Tabela 40, é apresentada uma métrica que representa a soma das variações sobre o valor ideal de 50% para cada uma das características avaliadas, representando o resultado geral do comparativo entre os cenários contrapostos. Sua análise fornece um critério

final de escolha, capaz de determinar a melhor relação custo–benefício em termos da redução de custo financeiro e tempo total de execução. Chamada de Ineficiência He–lastic, a métrica é calculada através da fórmula $VarPctg(\text{Duração}) + VarPctg(\text{Custo}) - 1$, representando, portanto, uma visão agregada das características de desempenho avaliadas onde ambas compartilham o mesmo nível de importância para o valor final.

Desta maneira é possível perceber que o modelo proposto apresenta, de maneira geral um impacto de performance, ou seja, uma perda de eficiência sobre a abordagem base do jModelTest, no intervalo de aproximadamente 5% à 15% entre todos os cenários contemplados na avaliação. Também é possível identificar que as perdas acompanham o crescimento da quantidade de recursos disponível em termos do número de CPUs alocados para uso, lembrando que devido às duas camadas de elasticidade empregadas no modelo o verdadeiro valor acaba variando conforme as características do processamento e as configurações adotadas.

Os resultados demonstram que, na comparação entre os cenários que contemplam o uso de até oito CPUs, há um impacto da ordem de 6% na adoção do modelo He–lastic, traduzindo em um número as conclusões obtidas no que diz respeito à duração e custo, onde foi possível atingir uma redução da duração de 2% sobre a abordagem do jModelTest enquanto utilizaram-se quase 10% mais recursos na forma de custo financeiro. Embora a redução na duração pareça pequena em termos percentuais, ela se traduz numa economia de quase 40 minutos (0:38:25) sobre as quase sete horas de execução contabilizadas na abordagem do jModelTest. Contudo, a contrapartida de custos para este ganho de 2% no tempo de execução, se manifesta na forma de um custo 9% maior, saindo de \$ 2,30 na abordagem do jModelTest, para \$ 3,27 quando há adoção do modelo He–lastic parametrizado conforme o cenário de testes C_{m1} .

No comparativo entre os cenários que alocam até 16 CPUs o quadro se agrava no que diz respeito à perda de eficiência, com o índice subindo para 8%. Esta perda é resultado da variação marginal no tempo de execução total do cenário, que variou somente 1% saindo de aproximadamente três horas e quarenta minutos (3:41:15), no caso do jModelTest, para três horas e cinquenta minutos (3:50:29), como resultado da adoção do modelo proposto. Sendo assim, o restante do impacto origina-se da variação do custo financeiro entre os cenários comparados, onde, apesar de próximos aos custos detectados na comparação com 8 CPUs, o modelo proposto incorre numa penalidade da ordem de 7%, saindo de \$ 2,51 para \$ 3,32.

Apesar de não negligenciáveis, os impactos obtidos na comparação dos cenários de 8 e 16 CPUs podem ser considerados aceitáveis uma vez que se mantém abaixo de 10%. Este cenário, contudo, se modifica na análise dos cenários que contemplam o uso de até 36 CPUs, onde a adoção do modelo He–lastic resulta em um impacto da ordem de 16%, dobrando as perdas identificadas no cenário anterior. A parcela mais significativa deste resultado é oriunda do tempo total de duração da execução do cenário, que levou aproximadamente três horas e meia (3:34:29), enquanto o jModelTest, mesmo com sua perda de eficiência previamente mencionada, atingiu um resultado na casa de duas horas e quinze minutos (2:13:16). O tempo total de execução no caso do modelo He–lastic surpreende na comparação com o cenário de 16 CPUs onde, apesar da

quantidade de recursos disponíveis mais que dobrar, o tempo de execução é reduzido em apenas 16 minutos. No que diz respeito ao custo a situação é mais favorável, com um impacto da ordem de 4% sobre o custo de referência do jModelTest, onde o valor passou de \$ 3,40 para \$ 3,99.

Ao revisitar os dados da [Tabela 38](#) e comparar os dados obtidos pelo cenário C_{m4} com o cenário que utiliza apenas a camada de Orquestração de Contêineres (C_{c36}), é possível determinar a origem desta perda de eficiência. Apesar do alto custo inicial para os arquivos de menor duração do *dataset*, o cenário baseado apenas na camada de Orquestração de Contêineres mantém durações médias próximas de 10 minutos para os arquivos 07, 08, 09 e 10, com o arquivo 11 resultando em aproximadamente 20 minutos de duração média. No caso do cenário misto, C_{m4} , os tempos de execução significativamente mais baixos para os arquivos iniciais, e de tamanho menor, acabam sendo ofuscados pela duração da casa de meia hora para os arquivos 08, 09 e 10, com o arquivo 11 tomando em média 43 minutos para o cálculo. No caso do arquivo 12-stamatakis-59 o impacto decorrente do uso das duas camadas é de apenas 15%, resultando em 10 minutos à mais de duração média frente ao tempo de execução base de uma hora, aproximadamente. Assim é possível, mais uma vez, determinar que existe necessidade de trabalhos futuros para otimizar a transição entre as camadas, uma iniciativa que parte da determinação do principal fator que causa impacto, com os proeminentes candidatos sendo a política de *retries* do serviço AWS Lambda, responsável pela camada FaaS, e o comportamento conservador do gerenciador de elasticidade do serviço AWS Batch, responsável pela camada de Orquestração de Contêineres.

Por fim, cabe salientar que, ainda na [Tabela 40](#), o custo financeiro total de execução do cenário do modelo He–lastic frente ao cenário do jModelTest variou apenas \$ 0,60, aproximadamente. Esta variação representa o menor nível de diferença dentre todas as análises comparativas, com a distância entre os custos diminuindo de 42%, para 32%, para finalmente 17% nos cenários de 8, 16 e 36 CPUs, respectivamente. Esta redução na distância entre os custos financeiros sugere que eventualmente o modelo He–lastic alcançaria, em termos dos custos financeiros, o custo oriundo do jModelTest. Contudo, este indício sozinho não garante melhorias na eficiência sem avaliar também a variação no tempo de execução dos respectivos cenários.

Outra característica que chama a atenção, ainda a respeito do custo, é pequena variação em termos absolutos, com valores que variam de \$ 2,30 à \$ 3,99 para a execução de toda a suite de cálculos de *best-fit* de sistemas de substituição filogenéticos contemplando todos os doze arquivos do *dataset*. Apesar do presente trabalho não exercer qualquer influência sobre a definição de custos por parte dos provedores de computação em nuvem, não deixa de ser relevante perceber a relativa facilidade com que se obtém, hoje em dia, grande quantidade de recursos computacionais com um custo relativamente baixo em valores absolutos. Estes custos reiteram, também, a necessidade de abordagens adaptadas para uso em ambientes de computação em nuvem, como é o caso do modelo He–lastic, aqui proposto, que tem como premissa a total inexistência de recursos alocados durante períodos de ociosidade.

No que diz respeito ao custo da ociosidade, uma evidente desvantagem do jModelTest

Tabela 41 – Evolução dos custos financeiros para manter disponíveis servidores capazes de processar os cenários avaliados para o jModelTest.

	C_{j2}	C_{j4}	C_{j8}	C_{j16}	C_{j36}
Resultado	(23:43:11) 0,08	(12:01:01) 0,09	(6:45:56) 0,10	(3:41:15) 0,10	(2:13:16) 0,14
Hora Cheia [†]	(24h) 0,09	(13h) 0,09	(7h) 0,10	(4h) 0,11	(3h) 0,19
+ 4h	0,10	0,12	0,16	0,23	0,45
+ 8h	0,11	0,15	0,21	0,34	0,70
+ 12h	0,13	0,18	0,27	0,45	0,96
+ dia	0,17	0,26	0,44	0,79	1,72
+ semana	0,68	1,28	2,48	4,87	10,90
+ mês	2,64	5,19	10,30	20,51	46,09

[†] Dentre os principais provedores de computação em nuvem é usual que a cobrança ocorra por hora, onde o uso por 5 minutos ou 55 minutos, por exemplo, é arredondado para cima, sendo cobrado como uma hora cheia.

Fonte: Elaborado pelo autor.

em ambientes de computação em nuvem, a Tabela 41 ilustra como os poucos centavos por execução do cenário de teste podem se transformar em dezenas de dólares em custo no fim do mês, indiferentemente do nível de utilização dos recursos. Impressiona, por exemplo, o caso do cenário C_{j36} , que é capaz de processar todo o *dataset* de testes num tempo médio de duas horas a um custo de apenas \$ 0,14, onde o custo para caso este mesmo nível de recursos seja mantido disponível durante um mês inteiro se aproxima dos 50 dólares. Mesmo que o nível de recursos fosse disponibilizado apenas durante uma semana, o custo financeiro já ultrapassa os 10 dólares, um valor que seria cobrado independentemente do nível de utilização dos recursos disponibilizados, se mantendo o mesmo seja para uma única execução do cenário de avaliação quanto para 75 delas (que é a quantidade máxima de execuções do cenário que cabem no período de uma semana).

Esta dinâmica de custo pela disponibilidade, ao invés do uso efetivo, também é observada em casos onde são utilizados recursos locais como na aquisição de *clusters* ou *grids* computacionais, fazendo com que o custo financeiro da execução dos cenários não seja produto da sua duração associada a quantidade de recursos utilizados. Nestes casos, seja no uso do jModelTest em ambiente de computação em nuvem, seja com recursos físicos, o custo financeiro da execução de qualquer um dos cenários aqui avaliados é auferido de maneira inversamente proporcional ao nível de utilização dos recursos. Este fenômeno faz com que exista uma significativa pressão para justificar a alocação ou compra de recursos computacionais, uma vez que seu uso deve estar embasado em uma real e pungente necessidade que comprove o uso contínuo destes recursos.

Por conta de burocracias e processos institucionais, são criadas barreiras de entrada para pesquisadores e/ou profissionais que não são especialistas em suas áreas e, portanto, carecem das habilidades necessárias para justificar um investimento com aportes de capital. Esta dinâmica

institucional acaba suprimindo a capacidade de experimentação e exploração sem compromisso, uma notável fonte de inovação que pode alimentada através de uma nova e/ou inusitada abordagem, mas que, por consequência, dificilmente será aceita como justificativa para um investimento em recursos computacionais dado que não há qualquer garantia prévia sobre a relevância dos resultados obtidos.

Embora o modelo proposto não tenha relação com o processo de compra e alocação de recursos computacionais, nem tampouco com as abordagens de pesquisa (seja por especialistas ou por novos entrantes), o fato de abolir custos ociosos através de um engenhoso emprego das suas duas camadas de elasticidade, uma abordagem totalmente *cloud-native*, favorece o acesso à grandes volumes de recursos computacionais em um curto espaço de tempo. Esta capacidade é de valorizada não somente para pesquisadores e profissionais individuais, como também para instituições de menor porte, onde a demanda por recursos computacionais não justifique a aquisição de servidores ou *clusters*.

A associatividade de custos oriunda do uso elástico de recursos da computação em nuvem permite o acesso sob demanda a grandes quantidades de recursos computacionais sem a necessidade de justificar seu uso contínuo. Neste sentido, o modelo He-lastic se apresenta como uma abordagem capaz de habilitar a elasticidade no projeto jModelTest, assim como outras aplicações mediante adaptações. Além disso, seu uso das camadas FaaS e de Orquestração de Contêineres permite o total descarte de recursos em momentos de ociosidade, contrastando com a clássica abordagem de elasticidade intermediada por平衡adores de carga que exige uma quantidade mínima de recursos sempre disponíveis. Desta forma o modelo proposto se mostra apropriado para cenários em que existe intermitência entre períodos de ociosidade e de necessidade por grande quantidade de recursos computacionais, podendo lidar tanto com execuções de curta duração, através da camada FaaS, quanto longas, utilizando o Orquestrador de Contêineres.

6.5 Considerações Parciais

Neste capítulo foram apresentados os resultados da avaliação do modelo proposto, seja através de dados sintéticos em um estudo de custo–benefício, seja através da execução dos cenários definidos na metodologia de avaliação. A análise destes resultados revela que, embora o modelo proposto esteja associado a uma perda de eficiência no processamento frente ao cenário ótimo de execução do jModelTest, sua adoção ainda é viável devido ao tratamento de recursos ociosos por meio das ações de elasticidade combinadas entre as duas camadas. Diferentemente do cenário base do jModelTest, o modelo proposto não necessita de recursos disponíveis enquanto espera por requisições, evitando cobranças em momentos de ociosidade.

Durante a análise de custos por meio de dados simulados foi possível compreender o comportamento que emerge da combinação entre as duas camadas de elasticidade do modelo proposto. Através das tabelas e gráficos apresentados fica evidente como se comporta a distribuição de custos entre as camadas conforme são ajustados seus parâmetros de funcionamento e cenários

em execução. Esta análise conjunta foi acompanhada de análises individualizadas das camadas de elasticidade, demonstrando as características de cada uma e representando a manifestação de uma abordagem prática sobre os conceitos previamente estudados a respeito das camadas de *Function as a Service* (FaaS) e de Orquestração de Contêineres.

A avaliação dos cenários base do jModelTest permitiu verificar seu comportamento a medida que é alterada a quantidade de recursos disponíveis. Estes dados são particularmente relevantes para estabelecer uma base sólida de resultados, abordando não apenas o tempo de execução como custo financeiro e métricas de variabilidade, também. Além disso, os resultados indicam uma tendência de desperdício de recursos e ociosidade que se manifestam de maneira proporcional ao aumento de recursos computacionais disponíveis. Através de análise do código-fonte da aplicação em conjunto com o comportamento ineficiente observado é possível indicar que a maior contribuição vem da estratégia de *Clustering Search*, adotada como heurística para evitar a busca exaustiva e que funciona através de fases de execução que contemplam de 8 a 120 tarefas paralelas em até seis etapas.

A seguir foram apresentados os resultados referentes a utilização do modelo He-lastic medidos através do protótipo implementado. Conforme definido na Metodologia de Avaliação ([Capítulo 5](#)), os cenários foram divididos em três grupos, conforme a utilização das camadas de elasticidade, com um cenário utilizando apenas FaaS (C_{f0}), três cenários utilizando somente Orquestração de Contêineres e variações na quantidade de recursos disponíveis (C_{C8} , C_{C16} e C_{C32}) e, por fim, mais sete cenários explorando variações nos parâmetros do modelo em uso Misto (C_{m1} , C_{m2} , C_{m3} , C_{m4} , C_{m5} , C_{m6} e C_{m7}). Uma diferença sobre os resultados do jModelTest foi que, apesar a Variância se manter baixa ao longo de todos os resultados, foi observado um aumento nos Coeficientes de Variação dos resultados, especificamente nos arquivos que fazem a transição entre camadas, contudo a suspeita é que este efeito foi ocasionado pela redução no número de amostras, consequência do aumento na quantidade de cenários avaliados.

A avaliação dos resultados que utilizam apenas a camada FaaS resultou na execução incompleta dos cenários em função dos arquivos maiores e que ultrapassavam os limites de recursos e tempo de execução estabelecidos pela tecnologia. Contudo, ainda assim foi possível obter valiosas informações a respeito da quantidade de Processos paralelos em execução, que se mostrou geralmente muito próxima ao nível máximo de paralelismo da execução por etapas. Também foi possível verificar indícios do comportamento de *retries* imposto pelo AWS Lambda, serviço utilizado para executar a camada FaaS, e como eles afetam a performance sob o ponto de vista do tempo de execução e a capacidade de processar integralmente os arquivos do *dataset*. Cabe mencionar também que os resultados de tempo de execução, quando comparados aos tempos nos cenários de avaliação do jModelTest, indicam que, para os arquivos que foram completamente processados, a camada FaaS leva de 8% a 84% do tempo, uma melhoria que se mostra ineficiente pois requer uma quantidade maior de recursos.

Ao analisar os resultados a execução baseada somente na Orquestração de Contêineres um dos objetivos foi determinar o comportamento de elasticidade adotado pelo serviço AWS

Batch, responsável por esta camada. A respeito dos tempos de execução foi possível observar um alto custo inicial, que se manifesta no tempo de execução dos arquivos menores, onde o *overhead* inicial acaba dominando o tempo total de execução. Quando comparados aos cenários do jModelTest fica evidente que este tipo de atraso inviabiliza o uso da camada de Orquestração de Contêineres para pequenos arquivos. Contudo, no caso dos maiores arquivos do *dataset*, esta demora na execução é progressivamente mitigada até se reverter em tempos de execução mais vantajosos no caso dos arquivos 10-Birds e 12-stamatakis-59.

O comportamento de elasticidade foi demonstrado graficamente juntamente com a quantidade de tarefas aguardando execução, o que permitiu verificar como o gerenciador de elasticidade se comporta em diferentes situações de carga. No geral é possível dizer que houve uso conservador de *thresholds* de elasticidade, o que pode ser visto pelo tempo que o gerenciador de elasticidade leva para descartar recursos ociosos. Também foi possível observar que a estratégia de *Clustering Search* adotada pela aplicação jModelTest pode confundir o gerenciador de elasticidade, causando atrasos até que ações corretivas sejam tomadas e reequilibrem a relação de carga e recursos. De maneira geral foi possível determinar que a camada de Orquestração de Contêineres é melhor adaptada para o tratamento de processos de maior duração e com menor amplitude nas variações de carga.

Os resultados dos cenários de uso Misto, com ambas as camadas de elasticidade funcionando em conjunto, mostrou que o modelo He–lastic é capaz de tirar proveito das melhores características de cada camada. Nos extremos do *dataset*, ou seja, nos menores e maiores arquivos, a performance do modelo se equipara à do jModelTest, contudo a interação entre as duas camadas dá origem a um significativo *overhead* que ocorre na transição entre as camadas. Neste caso merece destaque o caso do arquivo 08-HIV_vpu.ref2, cujo tempo de execução em alguns cenários chegou a levar 10 vezes aquele observado nos cenários do jModelTest.

Apesar do elevado nível de *overhead* associado a transição entre camadas, é possível como possíveis pontos de melhoria o controle sobre a quantidade de *retries* executados na camada FaaS, assim como a política de *backoff* exponencial, que é apropriada para uma ampla variedade de falhas, mas que, no caso do protótipo, não faz diferença. Contudo, os dados obtidos nos cenários de avaliação definidos neste trabalho não permitem estabelecer com segurança os motivos que causam esta degradação de performance, indicando a necessidade de investigações futuras sobre a transição entre as camadas de elasticidade.

Por fim, o capítulo apresentou uma análise do ponto de vista do custo financeiro, contemplando os cenários com uso Misto, ou seja, ambas as camadas de elasticidade ativas e comparando-os com os cenários do jModelTest. Dentre os sete cenários do modelo He–lastic foram escolhidos três para comparação direta com três outros cenários de avaliação do jModelTest, onde o casamento entre eles foi realizado em função da quantidade de CPUs utilizada. Nesta comparação foi possível determinar que o *overhead* de custo associado a utilização do modelo proposto está contido na faixa entre 4% e 9% acima do custo do jModelTest em cenário análogo. Quando associados ao tempo de execução, estes impactos podem ser avaliados através de um

indicador de ineficiência do He–lastic, que apresenta um comportamento ascendente, e portanto acompanha a quantidade de recursos disponíveis, estando contido na faixa entre 6% e 16%.

Contudo, é necessário ponderar que, apesar do impacto de performance e custo do modelo proposto, sua arquitetura permite o descarte total de recursos em momentos de ociosidade. Desta forma, não há a necessidade de nenhum servidor ou VM aguardando requisições de clientes, diferentemente do modelo de execução do jModelTest, e até mesmo de outras abordagens de elasticidade. Ao comparar os tempos de execução e custos incorridos exclusivamente durante o cálculo do jModelTest, este trabalho assume métricas ótimas de execução, desconsiderando custo associado ao tempo de ociosidade dos recursos utilizados na execução. Desta forma, a análise de viabilidade de uso do modelo He–lastic deve considerar não somente as métricas de tempo de execução e custo, como também um fator de ociosidade, que determinará o ponto de equilíbrio entre a abordagem proposta e o modelo de execução do jModelTest. Embora o fator de ociosidade seja uma característica específica para cada cenário de utilização, implicando conhecimentos específicos não contemplados neste trabalho, é possível adotar uma regra de referência, com base no indicador de ineficiência do modelo, em que a utilização do He–lastic se justifica em casos onde o nível de ociosidade ultrapassa 5% a 15%, conforme a quantidade de recursos utilizados.

7 CONCLUSÃO

*“The difference between winning and losing
is most often not quitting.”*

Walt Disney

A biologia, assim como diversas outras áreas do conhecimento, se beneficia de recursos tecnológicos para aumentar sua produtividade (DENNING; ROSENBLOOM, 2009). Tão importante são estas ferramentas que sua utilização deu origem a uma sub-área específica, a bioinformática que unificou pesquisadores da ciência da computação, biologia e até mesmo medicina para enfrentar problemas como o mapeamento do genoma humano (VENTER et al., 2001).

Apesar dos frutos desta união, paradigmas mais recentes como a computação em nuvem ainda são ausentes em se tratando de aplicações para filogenética. Visando mapear o estado da arte no que diz respeito aos projetos de software no âmbito da filogenética foi elaborada uma taxonomia, disponível na Figura 8, que os classifica em termos das suas categorias, métodos e finalidades. Embora esta taxonomia tenha cumprido seu papel no contexto deste trabalho, ainda é possível aprimorá-la buscando, por exemplo, eliminar a categoria ‘outras finalidades’, que ainda detém um número não trivial de trabalhos conforme mostra a Tabela 9.

Através de um levantamento bibliográfico, apresentado na Tabela 11 e summarizado do ponto de vista da computação pela Tabela 10 foi possível confirmar a hipótese da baixa penetração de novas tecnologias. Como resultado do levantamento foram identificadas três lacunas de pesquisa, sendo elas: 1) aceleração por GPU; 2) estratégias de balanceamento de carga; e 3) capacidade de tirar proveito da elasticidade de recursos. Assim é possível concluir que o estado atual, no que diz respeito ao uso de elasticidade no contexto de softwares para filogenética, sugere que seja possível obter significativos ganhos de desempenho e economia ao aplicar um modelo de computação apropriado para este cenário.

Para comprovar este ponto, o projeto jModelTest, um dos mais populares softwares no âmbito da filogenética (Seção 4.1), foi selecionado para servir como base para receber melhorias no seu projeto de maneira a contemplar o tratamento elástico de recursos em um ambiente de computação em nuvem. Uma análise aprofundada do software mostrou que existe um problema no que diz respeito à distribuição de complexidade computacional em suas tarefas paralelas, ilustrada pela Figura 14, onde determinadas tarefas são rapidamente concluídas e outras levam tempo considerável. Este cenário se mostra especialmente nocivo em um contexto de recursos fixos, como um *cluster* ou *grid* computacional, uma vez que recursos alocados para a execução ficam ocupados apenas brevemente, esperando ociosos a conclusão dos processos mais longos. A técnica de *Clustering Search* adotada pelo jModelTest agrava este cenário ao transformar uma computação embarcassamente paralela em fases, lembrando uma arquitetura BSP, cujas etapas

apresentam grande variabilidade na quantidade de tarefas, com valores variando entre 8 e 120 tarefas paralelas por fase.

Visando especificamente combater este cenário, a proposta do modelo He–lastic lança mão de uma estratégia de elasticidade em duas camadas, detalhada nas Figuras 16, 18 e 20, onde a primeira camada, baseada em FaaS, fica responsável por absorver tarefas de curta duração e a segunda camada, baseada em Orquestração de Contêineres, absorve o restante das tarefas de médio e longo prazo. Tal estratégia se justifica uma vez compreendidos os pontos fortes de cada uma das abordagens, descritas em detalhes na Tabela 14. Enquanto a camada FaaS tem por característica fundamental o modelo de cobrança mais granular e a ampla elasticidade embutida, ela impõe limites agressivos no tempo de execução e recursos disponíveis, fazendo com que não seja apropriada para qualquer tipo de processamento, vide Subseção 2.2.5. Em contrapartida, a camada de Orquestração de Contêineres alivia esta limitação em troca de maior fardo operacional, obrigando o usuário a gerenciar a elasticidade em termos dos mecanismos de Regra–Condição–Ação e se assegurar do efetivo aproveitamento dos recursos alocados.

Para avaliar a proposta foi adotada uma metodologia baseada na comparação entre cenários de comportamento do jModelTest e do modelo He–lastic. Para viabilizar esta análise comparativa, um conjunto de arquivos de tamanho e complexidade de cálculos progressivos foram selecionados como *dataset* de testes, tendo por objetivo detectar as características de comportamento das aplicações avaliadas conforme se altera a carga de trabalho. No caso da avaliação do jModelTest foram adotados cinco cenários de teste com variação na quantidade de CPUs disponível, uma vez que este é o recurso mais demandado pela aplicação. Os cenários do modelo He–lastic foram divididos em três grupos, contemplando a avaliação de cada camada, individualmente, assim como cenários de uso combinado, resultando em 11 variações conforme os parâmetros do modelo para quantidade de CPUs, Potência e Tempo Limite.

Ainda como parte da metodologia de avaliação foram detalhadas as características da aplicação jModelTest e que puderam ser exploradas para viabilizar uma avaliação fiel entre o programa e o protótipo do modelo proposto. A respeito do protótipo, foram detalhadas as etapas de implementação e suas características, assim como as ferramentas e arquitetura empregadas durante o desenvolvimento, o que, por fim, permitiu relatar pontos de dificuldade e desafios em relação aos serviços de computação em nuvem utilizados, adiantando possíveis anomalias que poderiam se manifestar nos resultados. Também foram descritos os parâmetros de operação do protótipo e sua relação com os parâmetros do modelo, além das características do ambiente de computação em nuvem adotado para desenvolvimento execução.

Por fim, os resultados obtidos através da aplicação da metodologia de avaliação sobre o protótipo e o programa jModeltest são detalhados no Capítulo 6, precedidos de um estudo de custo–benefício da utilização de duas camadas de elasticidade. Embora tenha sido realizado com dados hipotéticos, o estudo de custo–benefício permite uma melhor compreensão das dinâmicas de uso e cobrança referentes a elasticidade baseada em FaaS, assim como sua relação com a elasticidade baseada em Orquestração de Contêineres. Questões como a associatividade

de custos, definição de parâmetros de potência e tempo limite de execução, assim como a proporcionalidade da divisão de tarefas são exploradas em detalhes neste estudo, permitindo ao leitor uma melhor compreensão dos resultados que seguem.

A seguir são apresentados os dados referentes aos resultados obtidos pelos cenários de avaliação do *software* jModelTest, sendo demonstrados através de tabelas e visualizações gráficas. Os dados coletados descrevem o comportamento da aplicação conforme varia a quantidade de recursos disponível, mostrando claramente as características da aplicação, sendo de especial interesse a degradação de performance observada em função da estratégia de *Clustering Search*. Esta conclusão é suportada por uma profunda análise das características de execução e aproveitamento de recursos do jModelTest, acompanhada de gráficos que descrevem este comportamento.

Prosseguindo, uma análise dos resultados obtidos nos cenários de avaliação do protótipo do modelo He–lastic são apresentados em três partes, referentes aos resultados com processamento somente na camada FaaS, somente na camada de Orquestração de Contêineres, e os cenários do modelo completo, utilizando ambas as camadas. Nesta seção é possível observar que cada uma das camadas de elasticidade apresenta características distintas, com a camada FaaS resultando em baixa latência e amplo paralelismo, contudo incapaz de processar arquivos maiores do *dataset*, enquanto a camada de Orquestração de Contêineres demonstra um grande *overhead* associado ao início da sua operação, com o tempo de execução sendo dominado por este *overhead* para a maior parcela dos arquivos do *dataset*.

Os resultados da abordagem de camadas combinadas, ao mesmo tempo que herda a agilidade característica do FaaS nos arquivos menores, mantém um tempo de execução similar a abordagem de Orquestração de Contêineres para os maiores arquivos, contudo foi observada uma degradação significativa de performance na zona de migração de uma camada para a outra. Este *overhead* tem relação direta com a taxa de tarefas que transicionou de uma camada para a outra, sendo especialmente afetada pela política de *retries* imposta pela implementação da camada FaaS, assim como a política conservadora do gerenciador de elasticidade e pelo custo de inicialização da camada de Orquestração de Contêineres. Apesar da detalhada discussão acerca dos motivos que podem levar a este comportamento, os dados obtidos não fornecem insumos para uma conclusão segura acerca da causa raiz, sugerindo a necessidade de cenários de testes e avaliações adicionais.

Uma análise sobre o custo financeiro decorrente da execução dos cenários de teste encerra o capítulo de resultados, comparando os melhores cenários do modelo He–lastic com os cenários mais próximos da avaliação do jModelTest. Através desta análise foi possível quantificar o grau de impacto decorrente da adoção do modelo proposto, resultando num *overhead* que varia de 6% a 16% conforme a quantidade de recursos disponibilizados. Contudo, ao comparar a duração e o custo financeiro incorrido estritamente durante o tempo de execução dos cenários, a análise coloca o projeto jModelTest em pé de igualdade com o modelo proposto no que diz respeito a sua capacidade de escalar o uso de recursos, o que não é realista.

Desta forma, uma nova caracterização acerca dos custos financeiros associados a rodar o

jModelTest em um ambiente de nuvem é apresentada, contemplando não somente o tempo total de execução dos cenários, e seus respectivos custos, como também o tempo de disponibilidade dos recursos computacionais. Esta análise comprova que os custos extras associados a adoção do modelo He–lastic são justificáveis, uma vez que, no caso do jModelTest, o usuário acaba sendo cobrado pelo tempo em que o recurso computacional está disponível, independentemente do seu nível de utilização.

O conjunto destas características torna o modelo proposto especialmente apropriado para cenários de baixa utilização ou em que existe grande variabilidade na quantidade de recursos solicitados, o que na abordagem do jModelTest exigiria a alocação excessiva para atender às requisições de maneira satisfatória. A capacidade do modelo proposto de permanecer ocioso com custo zero representa um diferencial em situações onde há baixa frequência de requisições e que demandam grandes quantidades de recursos, resultando em uma cobrança efetivamente *pay-per-use* que se apoia na associatividade de custos oriunda da elasticidade. Por meio dos resultados consolidados é possível estabelecer que, conforme a quantidade de recursos utilizados, o modelo He–lastic passa a ser vantajoso para níveis de ociosidade maiores que 5% a 15%.

7.1 Contribuições

Traçando um paralelo aos objetivos que nos propomos a alcançar com este trabalho e que foram delineados na [Seção 1.3](#), podemos destacar cinco deles:

- 1) Taxonomia: com base no levantamento bibliográfico realizado no contexto deste trabalho e objetivando um entendimento compartilhado no âmbito da filogenética e suas ferramentas elaboramos a taxonomia apresentada na [Figura 8](#);
- 2) Mapeamento do Estado da Arte: onde foi identificado que uma parcela significativa dos projetos de *software* utilizados no âmbito da filogenética, apesar de já bem estabelecidos e populares, ainda não dá suporte à técnicas de computação de alto desempenho como a elasticidade;
- 3) Modelo He–lastic: partindo de um dos mais populares projetos de *software* filogenético, foi proposto um modelo, descrito na [Figura 20](#), que permite um comportamento verdadeiramente elástico no aproveitamento de recursos computacionais e capaz de lidar com tarefas com ampla variação de complexidade para que seu uso seja mais eficiente do ponto de vista computacional e financeiro;
- 4) Elasticidade em Duas Camadas: como decorrência do modelo adotado para a solução no contexto da filogenética chegamos a uma estratégia de elasticidade combinada em dois níveis ([Figura 16](#)) que pode ser generalizada e aplicada à uma série de problemas computacionais cuja característica marcante é a heterogeneidade na complexidade e esforço computacional de suas etapas paralelas, sendo dividido, a grosso modo, em uma camada

de elasticidade agressiva mas limitada em termos do tempo de execução, e uma camada de elasticidade mais estável e configurável, indicada para processamentos de média e longa duração;

- 5) Estudo de Custo–Benefício: que demonstrou, por meio de dados hipotéticos, as características da interação entre as duas camadas de elasticidade utilizadas pelo modelo proposto no que diz respeito ao custo financeiro apresentando, também, as características de custo das camadas de elasticidade de maneira independente, o que permite compreender melhor cada camada assim como sua estrutura de custos e cenários ideais de utilização.

Com base nestas contribuições esperamos, ao longo dos próximos meses, elaborar e submeter múltiplos artigos detalhando nossa abordagem e os resultados obtidos.

7.2 Trabalhos Futuros

A medida que foram analisados os resultados dos cenários de avaliação, ficam claras determinadas características e comportamentos do modelo proposto. É natural que dentre as descobertas reveladas pela análise dos resultados existam pontos de melhoria e aprimoramento, sendo os principais elencados abaixo:

- 1) Taxonomia: apesar de cumprir os objetivos propostos neste trabalho, a taxonomia elaborada teve de recorrer a categorias como ‘Outras Finalidades’ para contemplar todos os resultados do levantamento bibliográfico, constituindo um evidente ponto de aprimoramento devido à grande quantidade de elementos nesta categoria;
- 2) Conjuntos de Dados: a criação de *datasets* apropriados para representar de maneira fiel as necessidades de pesquisadores da bioinformática permanece uma questão em aberto, principalmente no que diz respeito a avaliação de *softwares* que realizam testes de adequação de sistemas de substituição molecular, como é o caso do jModelTest;
- 3) Métrica de Custo: as grandes diferenças oriundas da computação baseada em FaaS impediram o uso de uma métrica de custo capaz de unificar medidas oriundas da computação tradicional, elástica e por meio de FaaS. Desta forma, uma métrica compartilhada e normalizada dentre os três cenários seria ideal para remover a dependência no fator financeiro, que é estabelecido por provedores de computação em nuvem e inclui fatores econômicos que não são necessariamente relevantes para uma análise acadêmica de viabilidade;
- 4) Reaproveitamento de Processos na camada FaaS: nos arquivos muito pequenos, como o 01-aP6, foi detectado um impacto de performance significativo em termos percentuais devido ao *overhead* incorrido pela inicialização dos processos da camada FaaS, que acaba se tornando significativo frente ao curto tempo de execução do cálculo. Este comportamento se origina da decisão de executar cada teste de adequação dos sistemas de substituição

molecular em uma tarefa do FaaS, sendo que, para as tarefas de curtíssima duração, seria mais eficiente manter um número menor de unidades de execução ocupadas com múltiplas tarefas até que expirem seus *timeouts*.

- 5) Heurística: assim como abordado na Subseção 5.4.4, é possível obter ganhos de performance e redução de custo financeiro através da aplicação de uma heurística capaz determinar quando é vantajoso que uma solicitação seja processada diretamente pela camada de Orquestração de Contêineres, pulando completamente a camada FaaS, seja porque é possível prever que tal requisição não poderá ser atendida com os recursos e limites estabelecidos, ou porque a camada de Orquestração de Contêineres já está ativa e tem slots de processamento livres;
- 6) Comportamento de Grandes Arquivos: embora o *dataset* de avaliação conte com o arquivo 12-stamatakis-59, que pode ser considerado grande frente aos demais, foi possível perceber através das análises gráficas e tabelas que este arquivo apresenta um comportamento distinto dos anteriores. Contudo, a ausência de outros arquivos tão grandes quanto, ou maiores do que ele, dificulta extrapolar o comportamento e afirmar se as características observadas em seu processamento representam uma tendência e são preservadas conforme crescem os arquivos utilizados como entrada no cálculo;
- 7) Otimização da Transição entre Camadas: conforme indicado pelos resultados deste trabalho, há a necessidade de dedicar atenção especial para os custos associados à transição de camadas, como é o caso do arquivo 08-HIV_vpu.ref2, que chega a apresentar uma degradação de 1084% em casos extremos. Embora dois fenômenos se apresentem como prováveis causas, os resultados aqui obtidos não permitem atribuição quanto a causa raiz deste comportamento, carecendo de investigação aprofundada.

REFERÊNCIAS

- ABASCAL, F.; ZARDOYA, R.; POSADA, D. ProtTest: selection of best-fit models of protein evolution. **Bioinformatics**, [S.l.], v. 21, n. 9, p. 2104–2105, may 2005.
- ADACHI, J.; HASEGAWA, M. **MOLPHY version 2.3: programs for molecular phylogenetics based on maximum likelihood**. 1996. 1–150 p. Tese (Doutorado em Ciência da Computação) — University of Tokyo Japan, 1996.
- AFGAN, E. et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. **Nucleic acids research**, [S.l.], v. 44, n. W1, p. W3–W10, 2016.
- AJI, A. M. et al. On the efficacy of GPU-integrated MPI for scientific applications. **Proceedings of the 22nd international symposium on High-performance parallel and distributed computing - HPDC '13**, [S.l.], p. 191, 2013.
- AKAIKE, H. A New Look at Statistical Model Identification. **IEEE Transactions on Automatic Control**, [S.l.], v. 19, n. 6, p. 716–723, 1974.
- ANISIMOVA, M.; GASCUEL, O. Approximate likelihood-ratio test for branches: a fast, accurate, and powerful alternative. **Systematic Biology**, [S.l.], v. 55, n. 4, p. 539–552, 2006.
- ARCHER, J. et al. The evolutionary analysis of emerging low frequency HIV-1 CXCR4 using variants through time—an ultra-deep approach. **PLoS Computational Biology**, [S.l.], v. 6, n. 12, 2010.
- ARMBRUST, M. et al. A view of cloud computing. **Communications of the ACM**, Berkeley, v. 53, n. 4, p. 50, apr 2010.
- AUBIN, M.; RIGHI, R. D. R. **ELASTIPIPE: promovendo a elasticidade de aplicações organizadas em pipelines**. 2015. 30 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) — Universidade do Vale do Rio dos Sinos, 2015.
- AWADA, U.; BARKER, A. Resource efficiency in container-instance clusters. In: **SECOND INTERNATIONAL CONFERENCE ON INTERNET OF THINGS, DATA AND CLOUD COMPUTING - ICC '17**, 2017, New York, New York, USA. **Proceedings...** ACM Press, 2017. p. 1–5.
- BAKER, M.; BUYYA, R. Cluster computing: the commodity supercomputer. **Software: Practice and Experience**, Hoboken, v. 29, n. 6, p. 551–576, may 1999.
- BAKER, M.; CARPENTER, B.; SHAFI, A. MPJ express: towards thread safe java hpc. In: **IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, ICCC, 2006. Proceedings...** [S.l.: s.n.], 2006.
- BERNSTEIN, D. Containers and Cloud: from lxc to docker to kubernetes. **IEEE Cloud Computing**, [S.l.], v. 1, n. 3, p. 81–84, sep 2014.
- BERSANI, M. M. et al. Towards the formalization of properties of cloud-based elastic systems. In: **INTERNATIONAL WORKSHOP ON PRINCIPLES OF ENGINEERING SERVICE-ORIENTED AND CLOUD SYSTEMS - PESOS 2014**, 6., 2014, New York, New York, USA. **Proceedings...** ACM Press, 2014. n. August, p. 38–47.

- BLOMBERG, S. P.; GARLAND, T.; IVES, A. R. TESTING FOR PHYLOGENETIC SIGNAL IN COMPARATIVE DATA: behavioral traits are more labile. **Evolution**, [S.l.], v. 57, n. 4, p. 717–745, apr 2003.
- BOLLOBACK, J. P. SIMMAP: stochastic character mapping of discrete traits on phylogenies. **BMC Bioinformatics**, [S.l.], v. 7, 2006.
- BOUSSAU, B.; GOUY, M. Efficient likelihood computations with nonreversible models of evolution. **Systematic Biology**, [S.l.], v. 55, n. 5, p. 756–768, 2006.
- BOUSSAU, B.; GUÉGUEN, L.; GOUY, M. A mixture model and a Hidden Markov Model to simultaneously detect recombination breakpoints and reconstruct phylogenies. **Evolutionary Bioinformatics**, [S.l.], v. 2009, n. 5, p. 67–79, 2009.
- BURNS, B. et al. Borg, Omega, and Kubernetes - Lessons Learned from Three Container-Management Systems over a Decade. **ACM Queue**, [S.l.], v. 14, n. February, p. 70–93, 2016.
- CARMEL, L. et al. **EREM: parameter estimation and ancestral reconstruction by expectation-maximization algorithm for a probabilistic model of genomic binary characters evolution**. 2010. v. 2010.
- CATANZARO, D.; PESENTI, R.; MILINKOVITCH, M. C. Estimating phylogenies under maximum likelihood: a very large-scale neighborhood approach. In: **BMC BIOINFORMATICS**, 2008. **Anais...** [S.l.: s.n.], 2008.
- CHEN, S. C.; ROSENBERG, M. S.; LINDSAY, B. G. MixtureTree: a program for constructing phylogeny. **BMC Bioinformatics**, [S.l.], v. 12, 2011.
- CHOR, B.; TULLER, T. Maximum Likelihood of Evolutionary Trees Is Hard. In: **RESEARCH IN COMPUTATIONAL MOLECULAR BIOLOGY**, 2005, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 2005. p. 296–310.
- CLAVERIE, J.-M.; NOTREDAME, C. **Bioinformatics for Dummies**. 2nd. ed. [S.l.]: Wiley Publishing, 2007.
- COUTINHO, E. F. et al. Elasticity in cloud computing: a survey. **annals of telecommunications - annales des télécommunications**, [S.l.], v. 70, n. 7-8, p. 289–309, aug 2015.
- DARRIBA, D. **Selection of Models of Genomic Evolution in HPC Environments**. 2015. 145 p. Tese (Doutorado em Ciência da Computação) — Universidade da Coruña, 2015.
- DARRIBA, D. et al. ProtTest-HPC: fast selection of best-fit models of protein evolution. In: **EUROPEAN CONFERENCE ON PARALLEL PROCESSING**, 2010, Berlin. **Anais...** Springer, 2010. p. 177–184. (Lecture Notes in Computer Science).
- DARRIBA, D. et al. ProtTest 3: fast selection of best-fit models of protein evolution. **Bioinformatics**, [S.l.], v. 27, n. 8, p. 1164–1165, apr 2011.
- DARRIBA, D. et al. jModelTest2: more models, new heuristics and parallel computing. **Nature Methods**, [S.l.], v. 9, n. 8, p. 772, 2012.

- DARRIBA, D. et al. High-performance computing selection of models of DNA substitution for multicore clusters. **The International Journal of High Performance Computing Applications**, [S.I.], v. 28, n. 1, p. 112–125, feb 2014.
- DARWIN, C. **On the Origin of Species by Means of Natural Selection**. [S.l.: s.n.], 1859.
- DENNING, P. J.; ROSENBLOOM, P. S. The profession of ITComputing. **Communications of the ACM**, [S.I.], v. 52, n. 9, p. 27, sep 2009.
- DEREPPER, A. et al. Phylogeny.fr: robust phylogenetic analysis for the non-specialist. **Nucleic acids research**, [S.I.], v. 36, n. Web Server issue, 2008.
- DONGARRA, J. J.; LUSCZEK, P.; PETITET, A. The LINPACK Benchmark: past, present and future. **Concurrency and Computation: Practice and Experience**, [S.I.], v. 15, n. 9, p. 803–820, aug 2003.
- DRUMMOND, A. J. et al. Estimating mutation parameters, population history and genealogy simultaneously from temporally spaced sequence data. **Genetics**, [S.I.], v. 161, n. 3, p. 1307–1320, 2002.
- DRUMMOND, A.; STRIMMER, K. PAL: an object-oriented programming library for molecular evolution and phylogenetics. **Bioinformatics**, [S.I.], v. 17, n. 7, p. 662–663, 2001.
- DUTHEIL, J. et al. Bio++: a set of c++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics. **BMC Bioinformatics**, [S.I.], v. 7, 2006.
- EDGAR, R. C. MUSCLE: multiple sequence alignment with high accuracy and high throughput. **Nucleic Acids Research**, [S.I.], v. 32, n. 5, p. 1792–1797, 2004.
- EDWARDS, A. Estimation of the Branch Points of a Branching Diffusion Process. **Journal Of The Royal Statistical Society Series B-Methodological**, [S.I.], v. 32, n. 2, p. 155–174, 1970.
- EDWARDS, A. W. F.; CAVALLI-SFORZA, L. L. The reconstruction of evolution. **Human Genetics**, [S.I.], n. 27, 1963.
- EIVY, A. Be Wary of the Economics of "Serverless"Cloud Computing. **IEEE Cloud Computing**, [S.I.], v. 4, n. 2, p. 6–12, mar 2017.
- FELSENSTEIN, J. The Number of Evolutionary Trees. **Systematic Zoology**, [S.I.], v. 27, n. 1, p. 27, 1978.
- FELSENSTEIN, J. Evolutionary trees from DNA sequences: a maximum likelihood approach. **Journal of Molecular Evolution**, [S.I.], v. 17, n. 6, p. 368–376, nov 1981.
- FELSENSTEIN, J. Phylogenies From Molecular Sequences: inference and reliability. **Annual Review of Genetics**, [S.I.], v. 22, n. 1, p. 521–565, 1988.
- FELSENSTEIN, J. PHYLIP—Phylogeny Inference Package (Version 3.2). **Cladistics**, [S.I.], v. 5, n. 2, p. 163–166, 1989.
- FELSENSTEIN, J.; CHURCHILL, G. A. A Hidden Markov Model approach to variation among sites in rate of evolution. **Molecular Biology and Evolution**, [S.I.], v. 13, n. 1, p. 93–104, 1996.

- FELTER, W. et al. An updated performance comparison of virtual machines and Linux containers. In: IEEE INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE (ISPASS), 2015., 2015. **Anais...** IEEE, 2015. v. 25297, p. 171–172.
- FLEISSNER, R.; METZLER, D.; Von Haeseler, A. Simultaneous statistical multiple alignment and phylogeny reconstruction. **Systematic Biology**, [S.I.], v. 54, n. 4, p. 548–561, 2005.
- FRIEDMAN, N. et al. A structural EM algorithm for phylogenetic inference. **Journal of computational biology**, [S.I.], v. 9, n. 2, p. 331–53, 2002.
- GALANTE, G.; BONA, L. C. E. de. A Survey on Cloud Computing Elasticity. In: IEEE FIFTH INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 2012., 2012, Honolulu. **Anais...** IEEE, 2012. p. 263–270.
- GALTIER, N.; GOUY, M.; GAUTIER, C. SEAVIEW and PHYLO_WIN: two graphic tools for sequence alignment and molecular phylogeny. **Computer applications in the biosciences : CABIOS**, [S.I.], v. 12, n. 6, p. 543–548, 1996.
- GASCUEL, O. BIONJ: an improved version of the nj algorithm based on a simple model of sequence data. **Molecular Biology and Evolution**, [S.I.], v. 14, n. 7, p. 685–695, 1997.
- GOLOBOFF, P. A.; CATALANO, S. A. TNT version 1.5, including a full implementation of phylogenetic morphometrics. **Cladistics**, [S.I.], v. 32, n. 3, p. 221–238, 2016.
- GOOGLE, C. P. **Google Cloud Functions documentation | Cloud Functions | Google Cloud Platform**. Disponível em: <<https://cloud.google.com/functions/docs/>>. Acessado em junho 2018.
- GOUY, M.; GUINDON, S.; GASCUEL, O. SeaView Version 4: a multiplatform graphical user interface for sequence alignment and phylogenetic tree building. **Molecular Biology and Evolution**, [S.I.], v. 27, n. 2, p. 221–224, 2010.
- GOYAL, D. **Lambda vs EC2 cost comparasion - The ABC of Cloud**. Disponível em: <<http://www.theabcofcloud.com/lambda-vs-ec2-cost/>>. Acessado em dezembro 2018.
- GU, X.; ZHANG, J. A simple method for estimating the parameter of substitution rate variation among sites. **Molecular Biology and Evolution**, [S.I.], v. 14, n. 11, p. 1106–1113, 1997.
- GUINDON, S. et al. New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: assessing the performance of phylml 3.0. **Systematic Biology**, [S.I.], v. 59, n. 3, p. 307–321, 2010.
- GUINDON, S.; GASCUEL, O. A Simple, Fast, and Accurate Method to Estimate Large Phylogenies by Maximum Likelihood. **Systematic Biology**, [S.I.], v. 52, n. 5, p. 696–704, 2003.
- HAN, B. **An Introduction to Serverless and FaaS (Functions as a Service)**. 2017.
- HASEGAWA, M.; YANO, T.-a.; KISHINO, H. A New Molecular Clock of Mitochondrial DNA and the Evolution of Hominoids. **Proceedings of the Japan Academy, Series B**, [S.I.], v. 60, n. 4, p. 95–98, 1984.

- HELAERS, R.; MILINKOVITCH, M. C. MetaPIGA v2.0: maximum likelihood large phylogeny estimation using the metapopulation genetic algorithm and other stochastic heuristics. **BMC Bioinformatics**, [S.l.], v. 11, 2010.
- HENNESSY, J. L.; PATTERSON, D. A. **Arquitetura de Computadores: uma abordagem quantitativa**. 5. ed. Rio de Janeiro: Campus Elsevier, 2013. 744 p.
- HIGGINS, J.; HOLMES, V.; VENTERS, C. Orchestrating Docker containers in the HPC environment. In: LECTURE NOTES IN COMPUTER SCIENCE (INCLUDING SUBSERIES LECTURE NOTES IN ARTIFICIAL INTELLIGENCE AND LECTURE NOTES IN BIOINFORMATICS), 2015. **Anais...** Springer International Publishing, 2015. v. 9137 LNCS, p. 506–513.
- HINCHLIFF, C. E. et al. Synthesis of phylogeny and taxonomy into a comprehensive tree of life. **Proceedings of the National Academy of Sciences**, [S.l.], v. 112, n. 41, p. 12764–12769, 2015.
- HINDMAN, B. et al. Mesos: a platform for fine-grained resource sharing in the data center. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI '11), 8., 2011. **Proceedings...** [S.l.: s.n.], 2011.
- HORDIJK, W.; GASCUEL, O. Improving the efficiency of SPR moves in phylogenetic tree search methods based on maximum likelihood. **Bioinformatics**, [S.l.], v. 21, n. 24, p. 4338–4347, 2005.
- HOWE, K.; BATEMAN, A.; DURBIN, R. QuickTree: building huge neighbour-joining trees of protein sequences. **Bioinformatics**, [S.l.], v. 18, n. 11, p. 1546–1547, 2002.
- HUELSENBECK, J. P.; CRANDALL, K. A. PHYLOGENY ESTIMATION AND HYPOTHESIS TESTING USING MAXIMUM LIKELIHOOD. **Annual Review of Ecology and Systematics**, [S.l.], v. 28, n. 1, p. 437–466, nov 1997.
- HUELSENBECK, J. P.; RONQUIST, F. MRBAYES: bayesian inference of phylogenetic trees. **Bioinformatics**, [S.l.], v. 17, n. 8, p. 754–755, 2001.
- HUSON, D. H.; BRYANT, D. Application of phylogenetic networks in evolutionary studies. **Molecular Biology And Evolution**, [S.l.], v. 23, n. 2, p. 254–267, 2006.
- HUXLEY, J. **Evolution: the modern synthesis**. 3rd ed.. ed. [S.l.]: Allen and Unwin, 1974.
- IMAI, S.; CHESTNA, T.; VARELA, C. A. Elastic Scalable Cloud Computing Using Application-Level Migration. In: IEEE FIFTH INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING, 2012., 2012, Honolulu. **Anais...** IEEE, 2012. p. 91–98.
- JOBB, G.; Von Haeseler, A.; STRIMMER, K. TREEFINDER: a powerful graphical analysis environment for molecular phylogenetics. **BMC Evolutionary Biology**, [S.l.], v. 4, 2004.
- JONAS, E. et al. Occupy the cloud. In: SYMPOSIUM ON CLOUD COMPUTING - SOCC '17, 2017., 2017, New York, New York, USA. **Proceedings...** ACM Press, 2017. p. 445–451.
- JUKES, T.; CANTOR, C. Evolution of protein molecules. In: **In mammalian protein metabolism, vol. 3 (1969), pp. 21-132.** [S.l.: s.n.], 1969. v. III, p. 21–132.

- KALINOWSKI, S. T. How well do evolutionary trees describe genetic relationships among populations? **Heredity**, [S.l.], v. 102, n. 5, p. 506–513, 2009.
- KEANE, T. M. **Computational methods for statistical phylogenetic inference**. 2006. 155 p. PhD Thesis — National University of Ireland, 2006.
- KEANE, T. M. et al. DPRml: distributed phylogeny reconstruction by maximum likelihood. **Bioinformatics**, [S.l.], v. 21, n. 7, p. 969–974, apr 2005.
- KEANE, T. M. et al. Assessment of methods for amino acid matrix selection and their use on empirical data shows that ad hoc assumptions for choice of matrix are not justified. **BMC Evolutionary Biology**, [S.l.], v. 6, 2006.
- KEANE, T. M.; NAUGHTON, T. J.; MCINERNEY, J. O. MultiPhyl: a high-throughput phylogenomics webserver using distributed computing. **Nucleic Acids Research**, [S.l.], v. 35, n. Web Server, p. W33–W37, may 2007.
- KIMURA, M. A simple method for estimating evolutionary rate of base substitutions through comparative studies of nucleotide sequences. **Journal of Molecular Evolution**, [S.l.], v. 16, n. 2, p. 111–120, 1980.
- KORPELA, E. et al. SETI@HOME - Massively distributed computing for SETI. **Computing in Science and Engineering**, [S.l.], v. 3, n. 1, p. 78–83, 2001.
- Kosakovsky Pond, S. L.; FROST, S. D.; MUSE, S. V. HyPhy: hypothesis testing using phylogenies. **Bioinformatics**, [S.l.], v. 21, n. 5, p. 676–679, 2005.
- KUMAR, S.; STECHER, G.; TAMURA, K. MEGA7: molecular evolutionary genetics analysis version 7.0 for bigger datasets. **Molecular biology and evolution**, [S.l.], v. 33, n. 7, p. 1870–1874, jul 2016.
- KWIATKOWSKI, J. Evaluation of Parallel Programs by Measurement of Its Granularity. In: **Parallel processing and applied mathematics**. [S.l.: s.n.], 2002. p. 145–153.
- LARSON, S. M. et al. Folding@ Home and Genome@ Home: using distributed computing to tackle previously intractable problems in computational biology. In: **Modern methods in computational biology**. [S.l.: s.n.], 2002.
- LARTILLOT, N.; PHILIPPE, H. A Bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. **Molecular Biology and Evolution**, [S.l.], v. 21, n. 6, p. 1095–1109, 2004.
- Le Vinh, S.; SCHMIDT, H. A.; HAESELER, A. von. PhyNav: a novel approach to reconstruct large phylogenies. In: **CLASSIFICATION — THE UBIQUITOUS CHALLENGE**, 2005, Berlin, Heidelberg. **Anais...** Springer Berlin Heidelberg, 2005. p. 386–393.
- LEE, C. et al. CoMET: a mesquite package for comparing models of continuous character evolution on phylogenies. **Evolutionary bioinformatics online**, [S.l.], v. 2, p. 183–6, 2006.
- LEIGH, J. W. et al. Testing congruence in phylogenomic analysis. **Systematic Biology**, [S.l.], v. 57, n. 1, p. 104–115, 2008.
- LIÒ, P. et al. PASSML: combining evolutionary inference and protein secondary structure prediction. **Bioinformatics**, [S.l.], v. 14 8, p. 726–733, 1998.

- LOFF, J.; GARCIA, J. Vadara: predictive elasticity for cloud applications. In: IEEE 6TH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE, 2014., 2014, Singapore. **Anais...** IEEE, 2014. p. 541–546.
- LOLE, K. S. et al. Full-Length Human Immunodeficiency Virus Type 1 Genomes from Subtype C-Infected Seroconverters in India, with Evidence of Intersubtype Recombination. **Journal of Virology**, [S.l.], v. 73, n. 1, p. 152–160, 1999.
- LOUREIRO, M. et al. Grid selection of models of nucleotide substitution. **Studies in Health Technology and Informatics**, [S.l.], v. 159, p. 244–248, 2010.
- LUDWIG, W. et al. ARB: a software environment for sequence data. **Nucleic Acids Research**, [S.l.], v. 32, n. 4, p. 1363–1371, 2004.
- MADDISON, W. P.; MADDISON, D. R. **Mesquite: a modular system for evolutionary analysis**. 2018.
- MAIDAK, B. L. et al. The ribosomal database project. **Nucleic Acids Research**, [S.l.], v. 22, n. 17, p. 3485–3487, 1994.
- MAO, M.; HUMPHREY, M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: INTERNATIONAL CONFERENCE FOR HIGH PERFORMANCE COMPUTING, NETWORKING, STORAGE AND ANALYSIS ON - SC '11, 2011., 2011. **Proceedings...** [S.l.: s.n.], 2011.
- MARQUEZ, E. **Save yourself a lot of pain (and money) by choosing your AWS Region wisely**. Disponível em: <<https://www.concurrencylabs.com/blog/choose-your-aws-region-wisely/>>. Acessado em dezembro 2018.
- MARSHALL, P.; KEAHEY, K.; FREEMAN, T. Elastic Site: using clouds to elastically extend site resources. In: IEEE/ACM INTERNATIONAL CONFERENCE ON CLUSTER, CLOUD AND GRID COMPUTING, 2010., 2010, Melbourne. **Anais...** IEEE, 2010. p. 43–52.
- MARTIN, P. et al. Autonomic management of elastic services in the cloud. In: IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC), 2011., 2011, Kerkyra. **Anais...** IEEE, 2011. p. 135–140.
- MASSINGHAM, T.; GOLDMAN, N. EDIBLE: experimental design and information calculations in phylogenetics. **Bioinformatics**, [S.l.], v. 16, n. 3, p. 294–295, 2000.
- MASSINGHAM, T.; GOLDMAN, N. Detecting amino acid sites under positive selection and purifying selection. **Genetics**, [S.l.], v. 169, n. 3, p. 1753–1762, 2005.
- MAYROSE, I. et al. Comparison of site-specific rate-inference methods for protein sequences: empirical bayesian methods are superior. **Molecular Biology and Evolution**, [S.l.], v. 21, n. 9, p. 1781–1791, 2004.
- MAYROSE, I.; MITCHELL, A.; PUPKO, T. Site-specific evolutionary rate inference: taking phylogenetic uncertainty into account. **Journal of Molecular Evolution**, [S.l.], v. 60, n. 3, p. 345–353, 2005.

- MCGRATH, G.; BRENNER, P. R. Serverless Computing: design, implementation, and performance. In: IEEE 37TH INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS, ICDCSW 2017, 2017. **Proceedings...** [S.l.: s.n.], 2017.
- MELL, P.; GRANCE, T. The NIST definition of cloud computing. **NIST Special Publication**, [S.I.], v. 145, p. 7, 2011.
- MENDEL, G. Experiments in Plant Hybridisation. **Proceedings of the Natural History Society**, [S.I.], 1866.
- MEYER, S.; Von Haeseler, A. Identifying site-specific substitution rates. **Molecular Biology and Evolution**, [S.I.], v. 20, n. 2, p. 182–189, 2003.
- MILLER, M. A.; PFEIFFER, W.; SCHWARTZ, T. Creating the CIPRES Science Gateway for inference of large phylogenetic trees. In: GATEWAY COMPUTING ENVIRONMENTS WORKSHOP, GCE 2010, 2010., 2010. **Anais...** [S.l.: s.n.], 2010.
- MILOJIĆIĆ, D.; LLORENTE, I. M.; MONTERO, R. S. OpenNebula: a cloud management tool. **IEEE Internet Computing**, [S.I.], v. 15, n. 2, p. 11–14, 2011.
- MININ, V. et al. Performance-Based Selection of Likelihood Models for Phylogeny Estimation. **Systematic Biology**, [S.I.], v. 52, n. 5, p. 674–683, 2003.
- MISHRA, S. K.; SAHOO, B.; PARIDA, P. P. Load Balancing in Cloud Computing: a big picture. **Journal of King Saud University - Computer and Information Sciences**, [S.I.], feb 2018.
- MOORE, J. H. Bioinformatics. **Journal of Cellular Physiology**, [S.I.], v. 213, n. 2, p. 365–369, nov 2007.
- MOUNT, D. W. **Bioinformatics: sequence and genome analysis**. 2nd. ed. [S.I.]: Cold Spring Harbor Laboratory Press, 2004.
- MÜLLER, K. PRAP - Computation of Bremer support for large data sets. **Molecular Phylogenetics and Evolution**, [S.I.], v. 31, n. 2, p. 780–782, 2004.
- MÜLLER, K. SeqState: primer design and sequence statistics for phylogenetic dna datasets. **Applied Bioinformatics**, [S.I.], v. 4, n. 1, p. 65–69, 2005.
- NASCIMENTO, F. F.; REIS, M. D.; YANG, Z. **A biologist's guide to Bayesian phylogenetic analysis**. 2017. 1446–1454 p. v. 1, n. 10.
- NGUYEN, L. T. et al. IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. **Molecular Biology and Evolution**, [S.I.], v. 32, n. 1, p. 268–274, 2015.
- NOTREDAME, C. et al. T-coffee: a novel method for fast and accurate multiple sequence alignment. **Journal of Molecular Biology**, [S.I.], v. 302, n. 1, p. 205–217, 2000.
- NVIDIA. **CUDA C Programming Guide, version 9.2**. [S.l.: s.n.], 2018.
- NYLANDER, J. A. A. **MrModeltest 2.3**. 2004.

- OLSEN, G. J. et al. fastDNAmL: a tool for construction of phylogenetic trees of dna sequences using maximum likelihood. **Computer applications in the biosciences CABIOS**, [S.I.], v. 10, n. 1, p. 41–48, 1994.
- OSTELL, J.; MCENTYRE, J. The NCBI Handbook. **NCBI Bookshelf**, [S.I.], p. 1–8, 2007.
- PAHL, C. Containerization and the PaaS Cloud. **IEEE Cloud Computing**, [S.I.], v. 2, n. 3, p. 24–31, may 2015.
- PARETO, V. **Manual of Political Economy**. [S.I.]: Augustus M. Kelley, 1971. 504 p.
- PERERA, C. et al. Context Aware Computing for The Internet of Things: a survey. **IEEE Communications Surveys & Tutorials**, [S.I.], v. PP, n. 99, p. 1–41, 2013.
- POSADA, D. Using MODELTEST and PAUP* to Select a Model of Nucleotide Substitution. **Current Protocols in Bioinformatics**, [S.I.], v. 00, n. 1, p. 6.5.1–6.5.14, jan 2003.
- POSADA, D. Selecting models of evolution. In: **The phylogenetic handbook: a practical approach to dna and protein phylogeny**. 1. ed. Cambridge: Cambridge University Press, 2003. p. 430.
- POSADA, D. jModelTest: phylogenetic model averaging. **Molecular Biology and Evolution**, [S.I.], v. 25, n. 7, p. 1253–1256, apr 2008.
- POSADA, D.; CRANDALL, K. A. MODELTEST: testing the model of dna substitution. **Bioinformatics**, [S.I.], v. 14, n. 9, p. 817–818, oct 1998.
- PRICE, M. N.; DEHAL, P. S.; ARKIN, A. P. Fasttree: computing large minimum evolution trees with profiles instead of a distance matrix. **Molecular Biology and Evolution**, [S.I.], v. 26, n. 7, p. 1641–1650, 2009.
- RAJAN, D. et al. Converting a High Performance Application to an Elastic Cloud Application. In: **IEEE THIRD INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE**, 2011., 2011, Athens. **Anais...** IEEE, 2011. p. 383–390.
- RAMBAUT, A. Estimating the rate of molecular evolution: incorporating non-contemporaneous sequences into maximum likelihood phylogenies. **Bioinformatics**, [S.I.], v. 16, n. 4, p. 395–399, 2000.
- RAMÍREZ-FLANDES, S.; ULLOA, O. Bosque: integrated phylogenetic analysis software. **Bioinformatics**, [S.I.], v. 24, n. 21, p. 2539–2541, 2008.
- RANNALA, B.; YANG, Z. Probability distribution of molecular evolutionary trees: a new method of phylogenetic inference. **Journal of Molecular Evolution**, [S.I.], v. 43, n. 3, p. 304–311, 1996.
- RAVEENDRAN, A.; BICER, T.; AGRAWAL, G. A Framework for Elastic Execution of Existing MPI Programs. In: **IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING WORKSHOPS AND PHD FORUM**, 2011., 2011, Shanghai. **Anais...** IEEE, 2011. p. 940–947.
- RICE, P.; LONGDEN, L.; BLEASBY, A. EMBOSS: the european molecular biology open software suite. **Trends in Genetics**, [S.I.], v. 16, n. 6, p. 276–277, 2000.

- RIGHI, R. D. R. Elasticidade em cloud computing: conceito, estado da arte e novos desafios. **Revista Brasileira de Computação Aplicada**, Passo Fundo, v. 5, n. 2, p. 2–17, nov 2013.
- RIGHI, R. d. R. et al. Elastipipe: on providing cloud elasticity for pipeline-structured applications. In: **Lecture notes on data engineering and communications technologies**. [S.l.: s.n.], 2017. v. 1, p. 293–304.
- RIGHI, R. et al. Usando a Elasticidade de Recursos em Nuvem para Aumentar o Desempenho de Aplicações Pipeline. In: **Anais do xxxvi congresso da sociedade brasileira de computação (csbc 2016)**. Porto Alegre: [s.n.], 2016.
- ROBERTS, M.; FOWLER, M. **Serverless Architectures**. 2016.
- RODRÍGUEZ, A. A. et al. **Economics of ‘Serverless’**. Disponível em: <<https://www.bbva.com/en/economics-of-serverless/>>. Acessado em junho 2018.
- RODRIGUES, V. F. **AUTOELASTIC: explorando a elasticidade de recursos de computação em nuvem para a execução de aplicações de alto desempenho iterativas**. 2016. 2016 p. Mestrado — Universidade do Vale do Rio dos Sinos, 2016.
- SALTER, L. A.; PEARL, D. K. Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. **Systematic Biology**, [S.l.], v. 50, n. 1, p. 7–17, 2001.
- SÁNCHEZ, R. et al. Phylemon 2.0: a suite of web-tools for molecular evolution, phylogenetics, phylogenomics and hypotheses testing. **Nucleic Acids Research**, [S.l.], v. 39, n. SUPPL. 2, 2011.
- SANTORUM, J. M. et al. Jmodeltest.org: selection of nucleotide substitution models on the cloud. **Bioinformatics**, [S.l.], v. 30, n. 9, p. 1310–1311, 2014.
- SAUNDERS, C. T.; GREEN, P. Insights from modeling protein evolution with context-dependent mutation and asymmetric amino acid selection. **Molecular Biology and Evolution**, [S.l.], v. 24, n. 12, p. 2632–2647, 2007.
- SCHMIDT, H. A. et al. TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. **Bioinformatics**, [S.l.], v. 18, n. 3, p. 502–504, 2002.
- SCHWARZ, G. Estimating the Dimension of a Model. **The Annals of Statistics**, [S.l.], v. 6, n. 2, p. 461–464, 1978.
- SCHWARZKOPF, M. et al. Omega: flexible , scalable schedulers for large compute clusters. In: SOSP, 2013. **Anais...** [S.l.: s.n.], 2013.
- SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. OpenStack: toward an open-source solution for cloud computing. **International Journal of Computer Applications**, [S.l.], v. 55, n. 03, p. 38–42, 2012.
- SHANKAR, V. et al. **Numpywren: serverless linear algebra**. 2018.
- SHIMODAIRA, H.; HASEGAWA, M. CONSEL: for assessing the confidence of phylogenetic tree selection. **Bioinformatics**, [S.l.], v. 17, n. 12, p. 1246–1247, 2001.

- SHIRTS, M.; PANDE, V. S. **Screen savers of the world, unite!** 2000. 1903–1904 p. v. 290, n. 5498.
- SIEVERS, F. et al. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. **Molecular Systems Biology**, [S.l.], v. 7, n. 1, p. 539–539, 2014.
- SPILLNER, J.; MATEOS, C.; MONGE, D. A. FaaSter , Better , Cheaper : the prospect of serverless scientific computing and hpc. **Communications in Computer Science**, [S.l.], 2017.
- SPOIALA, C. **Pros and Cons of Serverless Computing. FaaS comparison: aws lambda vs azure functions vs google functions.** 2017.
- STAMATAKIS, A. An Efficient Program for Phylogenetic Inference Using Simulated Annealing. In: IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 19., 2005. **Anais...** IEEE, 2005. p. 198b–198b.
- STAMATAKIS, A. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. **Bioinformatics**, [S.l.], v. 30, n. 9, p. 1312–1313, 2014.
- STEWART, C. A. et al. Parallel Implementation and Performance of FastDNAml - A Program for Maximum Likelihood Phylogenetic Inference. **Supercomputing, ACM/IEEE 2001 Conference**, [S.l.], n. May 2014, p. 32, 2001.
- SULEIMAN, B. Elasticity Economics of Cloud-Based Applications. In: IEEE NINTH INTERNATIONAL CONFERENCE ON SERVICES COMPUTING, 2012., 2012, Honolulu. **Anais...** IEEE, 2012. p. 694–695.
- SWOFFORD, D. L. PAUP*. Phylogenetic analysis using parsimony (*and other methods). Version 4. **Sinauer Associates, Sunderland, Massachusetts**, [S.l.], 2002.
- TAMURA, K.; NEI, M. Estimation of the Number of Nucleotide Substitutions in the Control Region of Mitochondrial-DNA in Humans and Chimpanzees. **Molecular Biology and Evolution**, [S.l.], v. 10, n. 3, p. 512–526, 1993.
- TANABE, A. S. Kakusan4 and Aminosan: two programs for comparing nonpartitioned, proportional and separate models for combined molecular phylogenetic analyses of multilocus sequence data. **Molecular Ecology Resources**, [S.l.], v. 11, n. 5, p. 914–921, 2011.
- TÁRRAGA, J. et al. Phylemon: a suite of web tools for molecular evolution, phylogenetics and phylogenomics. **Nucleic Acids Research**, [S.l.], v. 35, n. SUPPL.2, 2007.
- TAVARÉ, S. Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences. **Lectures on Mathematics in the Life Sciences**, [S.l.], v. 17, p. 57–86, 1986.
- THAN, C.; RUTHS, D.; NAKHLEH, L. PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. **BMC Bioinformatics**, [S.l.], v. 9, 2008.
- TOSATTO, A.; RUIU, P.; ATTANASIO, A. Container-Based Orchestration in Cloud: state of the art and challenges. In: NINTH INTERNATIONAL CONFERENCE ON COMPLEX, INTELLIGENT, AND SOFTWARE INTENSIVE SYSTEMS, 2015., 2015. **Anais...** IEEE, 2015. p. 70–75.

TRAN, N.-L.; SKHIRI, S.; ZIMÁNYI, E. EQS: an elastic and scalable message queue for the cloud. In: IEEE THIRD INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE, 2011., 2011, Athens. **Anais...** IEEE, 2011. p. 391–398.

Van Noorden, R.; MAHER, B.; NUZZO, R. **The top 100 papers.** 2014. 550–553 p. v. 514, n. 7524.

VARIA, J.; MATHEW, S. **Overview of Amazon Web Services.** [S.l.: s.n.], 2017. (January).

VENTER, J. C. et al. The Sequence of the Human Genome. **Science**, [S.l.], v. 291, n. 5507, p. 1304–1351, feb 2001.

VERMA, A. et al. Large-scale cluster management at Google with Borg. In: TENTH EUROPEAN CONFERENCE ON COMPUTER SYSTEMS - EUROSYS '15, 2015. **Proceedings...** [S.l.: s.n.], 2015.

VINH, L. S.; Von Haeseler, A. IQPNNI: moving fast through tree space and stopping in time. **Molecular Biology and Evolution**, [S.l.], v. 21, n. 8, p. 1565–1571, 2004.

WANG, H. C. et al. Testing for covarion-like evolution in protein sequences. **Molecular Biology and Evolution**, [S.l.], v. 24, n. 1, p. 294–305, 2007.

WANG, L. et al. Peeking Behind the Curtains of Serverless Platforms. In: {USENIX} ANNUAL TECHNICAL CONFERENCE ({USENIX} {ATC} 18), 2018., 2018, Boston, MA. **Anais...** {USENIX} Association, 2018. p. 133—146.

WILKINSON, B.; ALLEN, C. M. **Parallel Programming: techniques and applications using networked workstations and parallel computers.** New Jersey: Prentice Hall, 1999. (An Alan R. Apt book).

XIA, X. **The DAMBE Manual.** [S.l.: s.n.], 2014.

XIA, X. DAMBE6: new tools for microbial genomics, phylogenetics, and molecular evolution. **Journal of Heredity**, [S.l.], v. 108, n. 4, p. 431–437, 2017.

YANG, Z. Statistical Properties of the Maximum Likelihood Method of Phylogenetic Estimation and Comparison With Distance Matrix Methods. **Systematic Biology**, [S.l.], v. 43, n. 3, p. 329–342, 1994.

YANG, Z. Estimating the pattern of nucleotide substitution. **Journal of Molecular Evolution**, [S.l.], v. 39, n. 1, p. 105–111, 1994.

YANG, Z. PAML 4: phylogenetic analysis by maximum likelihood. **Molecular Biology and Evolution**, [S.l.], v. 24, n. 8, p. 1586–1591, 2007.

YANG, Z. **Molecular Evolution: a statistical approach.** First Edit. ed. Oxford: Oxford University Press, 2014. 512 p.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of Internet Services and Applications**, Heidelberg, v. 1, n. 1, p. 7–18, may 2010.

ZHANG, S. et al. Asynchronous stochastic gradient descent for DNN training. In: ICASSP, IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING - PROCEEDINGS, 2013. *Anais...* [S.l.: s.n.], 2013. p. 6660–6663.

ZWICKL, D. J. **Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion.** 2006. 115 p. Dissertation — University of Texas at Austin, 2006.

APÊNDICE A – INFORMAÇÕES COMPLEMENTARES

Tabela 42 – Quantidade de citações para cada trabalho encontrado no levantamento bibliográfico
(continua)

	Projeto	Citações
PHYLIP	(FELSENSTEIN, 1989)	26148
MEGA	(KUMAR; STECHER; TAMURA, 2016)	22463
MUSCLE	(EDGAR, 2004)	21288
modeltest	(POSADA; CRANDALL, 1998)	20139
mrBayes	(HUELSENBECK; RONQUIST, 2001)	18909
paup*	(SWOFFORD, 2002)	17779
jmodeltest	(POSADA, 2008)	7725
PhyML	(GUINDON et al., 2010)	6776
jmodeltest2	(DARRIBA et al., 2012)	6053
EMBOSS	(RICE; LONGDEN; BLEASBY, 2000)	6020
T-Coffee	(NOTREDAME et al., 2000)	5878
PAML	(YANG, 2007)	5824
RAxML	(STAMATAKIS, 2014)	5726
ARB	(LUDWIG et al., 2004)	5416
splitstree	(HUSON; BRYANT, 2006)	4943
Clustal Omega	(SIEVERS et al., 2014)	4907
CIPRES	(MILLER; PFEIFFER; SCHWARTZ, 2010)	3627
GARLI	(ZWICKL, 2006)	3316
SeaView	(GOUY; GUINDON; GASCUEL, 2010)	2990
Phylogeny.fr	(DEREOPER et al., 2008)	2912
ProtTest	(ABASCAL; ZARDOYA; POSADA, 2005)	2685
mesquite	(MADDISON; MADDISON, 2018)	2650
tree-puzzle	(SCHMIDT et al., 2002)	2471
PHYSIG	(BLOMBERG; GARLAND; IVES, 2003)	2300
phylo_win	(GALTIER; GOUY; GAUTIER, 1996)	2254
HyPhy	(Kosakovsky Pond; FROST; MUSE, 2005)	1966
simplot	(LOLE et al., 1999)	1896

Tabela 42 – Quantidade de citações para cada trabalho encontrado no levantamento bibliográfico
(continuação)

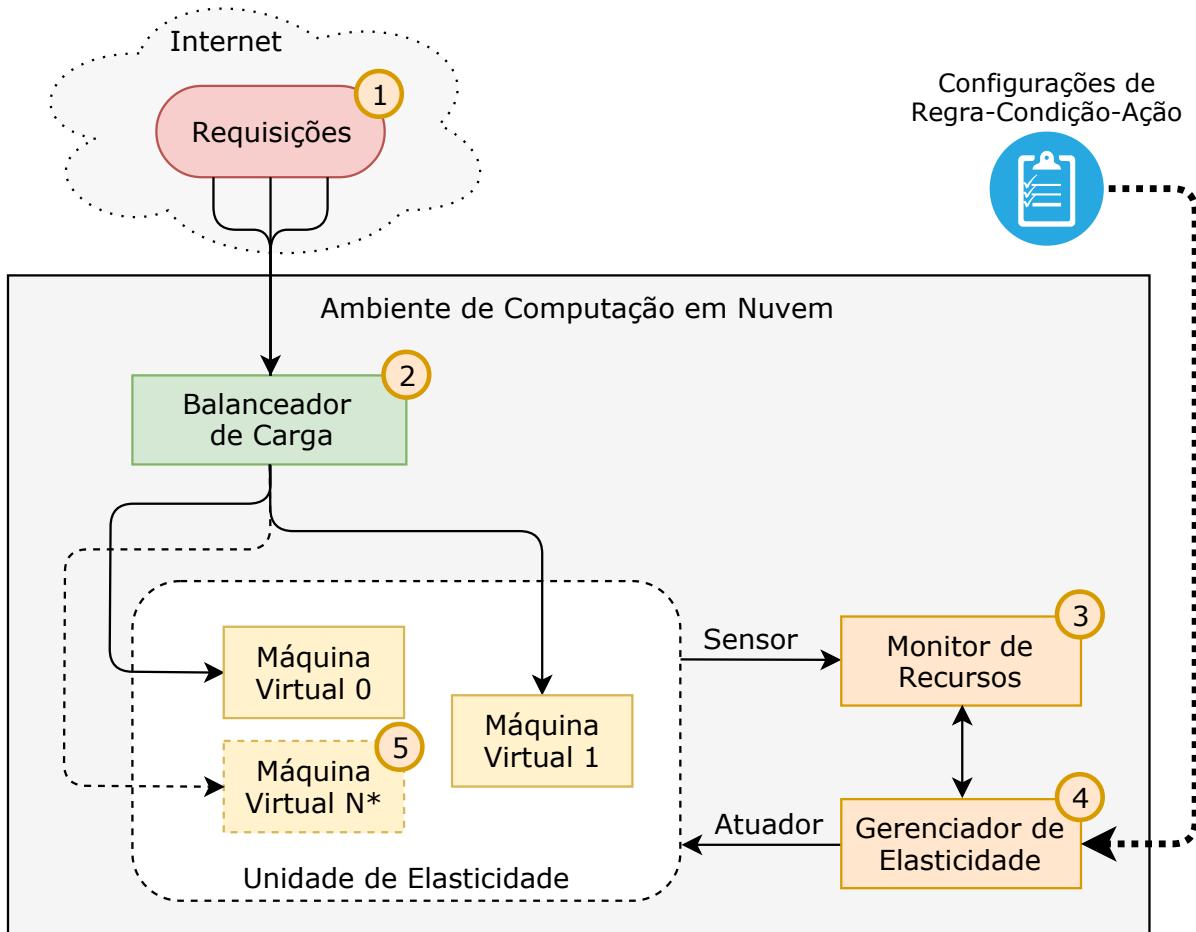
	Projeto	Citações
CONSEL	(SHIMODAIRA; HASEGAWA, 2001)	1764
PHYML-aLRT	(ANISIMOVA; GASCUEL, 2006)	1680
BioNJ	(GASCUEL, 1997)	1486
fasttree	(PRICE; DEHAL; ARKIN, 2009)	1357
fastDNAml	(OLSEN et al., 1994)	1212
treefinder	(JOBB; Von Haeseler; STRIMMER, 2004)	1118
MOLPHY	(ADACHI; HASEGAWA, 1996)	936
PhyloBayes	(LARTILLOT; PHILIPPE, 2004)	897
modelgenerator	(KEANE et al., 2006)	848
BEAST	(DRUMMOND et al., 2002)	830
DNAML	(FELSENSTEIN; CHURCHILL, 1996)	830
IQ-TREE	(NGUYEN et al., 2015)	708
DNArates	(MAIDAK et al., 1994)	697
GALAXY	(AFGAN et al., 2016)	462
SIMMAP	(BOLBACK, 2006)	425
DTModSel	(MININ et al., 2003)	377
rate4site	(MAYROSE et al., 2004)	323
TipDate	(RAMBAUT, 2000)	317
kakusan	(TANABE, 2011)	315
QuickTree	(HOWE; BATEMAN; DURBIN, 2002)	245
PRAP	(MÜLLER, 2004)	225
SLR	(MASSINGHAM; GOLDMAN, 2005)	200
PhyloNET	(THAN; RUTHS; NAKHLEH, 2008)	192
PAL	(DRUMMOND; STRIMMER, 2001)	179
Concaterpillar	(LEIGH et al., 2008)	162
TNT	(GOLOBOFF; CATALANO, 2016)	161
IQPNNI	(VINH; Von Haeseler, 2004)	146
SEMPHY	(FRIEDMAN et al., 2002)	145
GZ-gamma	(GU; ZHANG, 1997)	145

Tabela 42 – Quantidade de citações para cada trabalho encontrado no levantamento bibliográfico
(continuação)

	Projeto	Citações
TreeFit	(KALINOWSKI, 2009)	116
Bio++	(DUTHEIL et al., 2006)	114
Phylemon 2.0	(SÁNCHEZ et al., 2011)	110
nhPhyML	(BOUSSAU; GOUY, 2006)	110
Phylemon	(TÁRRAGA et al., 2007)	107
ALIFRITZ	(FLEISSNER; METZLER; Von Haeseler, 2005)	100
SSA	(SALTER; PEARL, 2001)	99
SeqState	(MÜLLER, 2005)	98
MultiPhyl	(KEANE; NAUGHTON; MCINERNEY, 2007)	75
METAPIGA	(HELAERS; MILINKOVITCH, 2010)	73
segminator	(ARCHER et al., 2010)	71
PROCOV	(WANG et al., 2007)	66
DPRML	(KEANE et al., 2005)	59
BOSQUE	(RAMÍREZ-FLANDES; ULLOA, 2008)	58
passml	(LIÒ et al., 1998)	56
PARAT	(MEYER; Von Haeseler, 2003)	46
McRate	(MAYROSE; MITCHELL; PUPKO, 2005)	28
DAMBE	(XIA, 2017)	27
CoMET	(LEE et al., 2006)	26
PhyML-Multi	(BOUSSAU; GUÉGUEN; GOUY, 2009)	15
EDIBLE	(MASSINGHAM; GOLDMAN, 2000)	15
PhyNav	(Le Vinh; SCHMIDT; HAESELER, 2005)	14
EREM	(CARMEL et al., 2010)	8
mixturetree	(CHEN; ROSENBERG; LINDSAY, 2011)	3
CodeAxe	(SAUNDERS; GREEN, 2007)	2
MrModeltest	(NYLANDER, 2004)	0
PhyloCoco	(CATANZARO; PESENTI; MILINKOVITCH, 2008)	0

Fonte: Elaborado pelo autor.

Figura 49 – Diagrama conceitual da abordagem tradicional para obtenção de elasticidade em um contexto de computação em nuvem: um balanceador de carga distribui as requisições para diversas réplicas de uma máquina virtual enquanto um monitor de recursos avaliam o estado das réplicas e sugerem ações ao gerenciador de elasticidade com o objetivo de adequar os recursos disponíveis à carga computacional.



Fonte: Elaborado pelo autor.

Fluxo de Execução:

1. Um número arbitrário de requisições são geradas na Internet;
2. Dentro de um ambiente de computação em nuvem, um balanceador de carga absorve estas requisições e as distribui para um grupo de máquinas virtuais configuradas em uma unidade de elasticidade, tipicamente de maneira *round-robin*;
3. Um serviço de monitoramento de recursos oferecido pelo provedor de computação em nuvem emite métricas de uso a respeito do consumo de recursos das máquinas virtuais;
4. Através do gerenciador de elasticidade são executadas as tuplas do tipo ‘Regra – Condição – Ação’, previamente configuradas pelo usuário e que determinam como a unidade de elasticidade deve reagir, seja adicionando ou removendo réplicas;
5. No caso de excesso de demanda um conjunto de novas máquinas virtuais podem ser criadas, distribuindo a carga entre todas as disponíveis.