1. TÍTULO

Gerenciamento de Recursos em Nuvens

2. INTRODUÇÃO

Desde meados dos anos 2000, com o surgimento das primeiras nuvens comerciais, a Computação em Nuvens vem se consolidando como o principal modelo de computação sob o qual empresas de diferentes áreas vêm se apoiando para construir sua infraestrutura computacional. Não raro, empresas que antes possuíam essa infraestrutura em um modelo *On-premises,* no qual servidores ficam localizados fisicamente na empresa, passam a migrar para o ambiente em nuvem. Essa migração se dá devido às facilidades que o ambiente em nuvem proporciona, como a dispensabilidade da necessidade de gerir a manutenção, de *hardware* ou *software,* de seus servidores próprios, atrelada ao custo da mão de obra especializada para tal. O modelo de pagamento da Computação em Nuvem também tem motivado a ascensão deste modelo de computação, já que é semelhante a um serviço de utilidade pública, pagando-se apenas pela capacidade computacional utilizada, como proposto por John McCarthy em 1961 [Garfinkel, 2011]. Ademais, a Computação em Nuvem fornece uma forma eficiente para lidar com a flutuação da demanda, de modo que o serviço é adaptável e se expande de acordo com a necessidade, característica chamada escalabilidade. A escalabilidade faz com que empresas optem pela Computação em Nuvem, pois, por exemplo, caso um site receba uma demanda além do tráfego habitual em um dia de Black Friday, basta que o provedor gerencie isso, o que não ocorreria em um ambiente *On-premises*. Nesse caso, se a empresa quisesse comportar essa demanda, seria necessário aumentar sua infraestrutura para esse tipo de ocasião incomum, o que faria com que, durante a maioria do ano, seu poder computacional ficasse ocioso.

A virtualização é o meio pelo qual provedores de serviços em nuvem são capazes de fornecer computação sob demanda e oferecê-los a seus clientes na Internet [Sharma et. al., 2016]. Esse conceito surgiu na década de 60 com o objetivo de reaproveitar os recursos dos enormes *mainframes* e torná-los ainda úteis frente ao surgimento dos até então chamados "minicomputadores". A virtualização diz respeito à camada de abstração sobre o *hardware* do computador, implementada por *software,* que permite que seus componentes de processamento, memória e armazenamento sejam particionados entre várias máquinas virtuais (VMs, do inglês *virtual machines*). Desse modo, cada máquina virtual executa seu próprio sistema operacional de forma isolada em relação às demais VMs sobre um *host* (a máquina física) comum. Isso favorece a busca por eficiência e contribui para a estabilidade e confiabilidade de sistemas, uma vez que reduz custos ao viabilizar a execução de diversas VMs em um mesmo servidor físico e facilita execuções de aplicações em uma variedade de sistemas operacionais. Entretanto, as VMs apresentam certas desvantagens que devem ser consideradas, como o impacto no tempo de inicialização devido ao

fato de que cada VM possui um sistema operacional completo, o que acabou por motivar a emersão de uma abordagem mais leve de certa forma.

Dentro desse contexto, os *containers* surgem como uma peça crucial no que se refere à implantação e gestão eficiente de aplicações na nuvem. *Containers* são ambientes isolados para o empacotamento de aplicações, com o fim de facilitar a consistência e a portabilidade entre diferentes ambientes; são construídos sobre um tipo de virtualização. Os *containers* podem levar a uma maior eficiência de execução do que as VMs em razão de compartilharem o mesmo *kernel* do sistema operacional, ao contrário das VMs, que precisam de instâncias do *kernel* separadas [Sharma et. al., 2016]. Além disso, o empacotamento da aplicação facilita a migração do ambiente de desenvolvimento para o ambiente de produção ao tornar possível a simulação do ambiente no qual ela executará. Toda essa facilidade fez com que o modelo de arquitetura de *software* de Microsserviços surgisse. Utilizando-se Microsserviços, o *software* se divide em pequenos serviços independentes e de responsabilidade única, que se comunicam utilizando APIs. Esse é um exemplo prático no qual os *containers* em nuvem têm sido implantados, pois comumente cria-se um *container* para cada serviço independente de modo que se torna possível escalar cada um deles de acordo com sua demanda.

Vale ressaltar que, além de empresas, cientistas também têm se aventurado na Computação em Nuvem para executar tarefas que consomem grande poder computacional sem possuírem grandes computadores, como biólogos ou físicos, com seus alinhamentos genéticos e problemas que envolvem Álgebra Linear, por exemplo. A diversidade de áreas de atuação dos usuários, que nem sempre são da computação, por vezes gera dificuldades, pois na maioria das vezes é de responsabilidade do usuário definir quais recursos computacionais sua tarefa demandará, em quantidade de vCPU, GiB de memória, arquitetura e sistema operacional. Visto que os preços são proporcionais a esses parâmetros, o exagero na alocação de recursos pode gerar custos desnecessários e indesejados pelo usuário, assim como a falta deles pode comprometer a execução da computação. Assim, cabe uma análise a respeito da facilidade de uso das ferramentas de Computação em Nuvem.

Este projeto tem como objetivo principal avaliar a execução de uma aplicação de alto desempenho em ambiente de nuvens, gerenciando o ambiente virtual e uso de recursos disponibilizados. Mais especificamente, este trabalho tem como intuito avaliar a implantação de *containers* para execução da aplicação no serviço de AWS (*Amazon Web Services*) Amazon ECS (*Elastic Container Service*) em conjunto com Amazon EC2 *(Amazon Elastic Compute Cloud*) e AWS Fargate, de forma disjunta. Essa avaliação busca compreender o equilíbrio entre o desempenho em termos de tempo, o custo, e a facilidade de uso dos serviços Amazon EC2 e AWS Fargate quando empregados como infraestrutura para o Amazon ECS. Como caso de estudo, uma aplicação da bioinformática de comparação genética será utilizada, avaliando também a aplicabilidade de tais serviços providos pela nuvem AWS a uma aplicação de importância científica.

3. METODOLOGIA

Dado que a execução dos experimentos perpassa por ferramentas que precisam ser primeiro compreendidas, como Docker e os serviços específicos da AWS nesse escopo, compreendê-los é um ponto fundamental para a metodologia. Como apoio à configuração experimental para avaliar o gerenciamento de recursos em nuvem, faz-se necessária também uma análise sobre a aplicação caso de uso selecionada. Dessa forma, a Seção 3.1 discorre acerca de tecnologias e serviços de interesse ao utilizar a nuvem, a Seção 3.2 sobre a aplicação caso de uso e da metodologia para o entendimento inicial de seu comportamento e a Seção 3.3 descreve as configurações experimentais e estratégias escolhidas para a utilização e avaliação de ferramentas de nuvem em *container*.

3.1. Elementos de Interesse do Ambiente de Nuvens

Esta seção apresenta as principais tecnologias que têm sido amplamente utilizadas no mercado no que diz respeito à execução e gerenciamento de *containers* em nuvem, as quais foram utilizadas no presente trabalho. As ferramentas relacionadas especificamente aos *containers* são as seguintes:

- Docker, para encapsular a aplicação, com boa integração a serviços de orquestração;
- Amazon ECR (*Elastic Container Registry*), para armazenar seguramente as imagens de *containers* geradas;
- Amazon EC2 e AWS Fargate: como ambiente de infraestrutura no qual os *containers* executarão, com características diferentes a serem exploradas;
- Amazon ECS, como serviço de orquestração dos *containers*.

Essas ferramentas foram selecionadas a fim de analisar sua robustez em relação à construção de aplicações escaláveis e confiáveis, no que tange não só ao poder computacional, mas também ao preço e à facilidade de uso que seu uso conjunto pode proporcionar. As seguintes ferramentas propiciaram esta análise, e foram utilizadas do seguinte modo:

- AWS S3 (*Amazon Simple Storage Service*), para armazenar dados de saída e entrada da aplicação;
- AWS Lambda (*Amazon Lambda*), como serviço utilizado para realizar computações em resposta a eventos que ocorrem no *container* que executa a aplicação.

Docker

O Docker é uma *container engine* que simplifica o processo de criação, implantação e execução de aplicativos em *containers*. Dessa forma, é possível provisionar a aplicação contendo apenas o necessário para sua execução, como bibliotecas e *softwares* [Docker Inc., 2024]. Utilizando essa ferramenta, a criação de um *container* se dá por meio da construção de uma imagem de *container*, um arquivo executável que encapsula a aplicação e suas dependências. Esse arquivo se chama Dockerfile e é um *script* de texto que possui a sequência de instruções para

a criação do *container*, promovendo replicabilidade e automatização. Essa imagem pode ser compartilhada via repositórios públicos ou privados, como o Docker Hub ou mesmo o Amazon ECR.

Além da construção de imagens de *containers*, o Docker favorece a integração harmoniosa com ferramentas de orquestração amplamente utilizadas como o Kubernetes, que é "*um sistema de código aberto para implantar, escalonar e gerenciar aplicativos em contêineres em qualquer lugar*" [Google Cloud, 2024]. Isso torna a utilização do Docker interessante, pois facilita o processo de gerenciamento dos diversos *containers* que uma aplicação em Microsserviços pode possuir, por exemplo.

Amazon ECR

O Amazon ECR é um serviço gerenciado pela AWS de registro de imagens de *container*. Ele fornece o recurso de criar repositórios privados com permissões baseadas em regras que envolvem o AWS IAM (*Identity Access Management*), o serviço com o qual é possível gerenciar quais usuários terão acesso aos recursos da AWS, o que provê segurança.

O pagamento é estabelecido com base na quantidade de dados armazenados nos repositórios públicos ou privados do usuário e pelos dados transferidos para a Internet. Além da cobrança pelo armazenamento, há também a cobrança de transferência de dados da conta da AWS que faz o *download* de repositórios públicos [Amazon Web Services Inc, 2024a].

Ambiente virtual Amazon EC2

O Amazon EC2 é um serviço de Computação em Nuvem oferecido pela AWS que permite que os usuários utilizem ambientes de máquinas virtuais (*Virtual Machines ou* VMs) em um modelo de computação em nuvem sob demanda. Por meio desse serviço, o usuário consegue escolher dentre uma ampla variedade de tipos de instâncias virtuais (as VMs), sendo estas otimizadas para casos de uso específicos, incluindo instâncias focadas em computação de alto desempenho. Assim, é possível escolher quantos vCPUs, quantos GiB de memória principal, qual sistema operacional e qual será o armazenamento de sua instância.

Seu modelo de cobrança sob demanda é definido em função do tipo de instância escolhido, cada uma com uma taxa específica de dólares por hora ou segundo, com o pagamento mínimo de 60 segundos [Amazon Web Services Inc, 2024b]. O tempo de utilização é contabilizado desde o momento em que a instância é iniciada até o momento em que é interrompida ou encerrada. Alguns fatores devem ser configurados e refletem no custo e disponibilidade, de acordo com [Amazon Web Services Inc, 2024c]:

- Sistema Operacional: geralmente o mais caro é o sistema *Windows.*
- Região: onde a instância está localizada, como por exemplo no Norte da Virgínia (‘*us-east-1*’) ou Frankfurt (‘*eu-central-1*’).
- Modelo de Mercado ou cobrança: *On-demand* - com disponibilidade para todo o tempo requisitado pelo usuário; *Spot*: com oferta de preço em até 90% inferior ao comparado com a correlata *on-demand*, mas tais instâncias podem ser revogadas pelo provedor quando necessárias. Este trabalho não fará uso de tal mercado.

Ambiente virtual AWS Fargate

O AWS Fargate é um serviço gerenciado pela AWS que simplifica a execução de *containers* sem a necessidade de gerência da infraestrutura subjacente por meio de uma abordagem *serverless*, que tira do usuário a responsabilidade de lidar com os servidores. Assim, a AWS fica responsável por provisionar e dimensionar os recursos demandados. Tanto o AWS ECS quanto o AWS EKS (*Elastic Kubernetes Service*) podem utilizar o AWS Fargate como infraestrutura para lançar seus *containers* e executar suas tarefas.

Seu modelo de pagamento se dá de acordo com os recursos utilizados pelos *containers* executados sobre o ECS e EKS. O preço é, então, dado em função de:

- recursos utilizados: número de vCPUs, arquitetura de CPU, sistema operacional, quantidade de memória e de armazenamento consumidos pela aplicação; e
- região de localização do *container*: preços variam de acordo com a região.

O tempo de utilização desses recursos é computado desde o momento em que se inicia a instalação da imagem do *container* até o momento em que a tarefa do Amazon ECS ou o pod do Amazon EKS termina. Arredonda-se, por fim, esse tempo para o segundo mais próximo. É importante dizer que há uma cobrança mínima de um minuto de execução caso seja necessário executar durante menos tempo [Amazon Web Services Inc, 2024d].

Orquestrador de *containers* Amazon ECS

O Amazon ECS, segundo sua documentação, é um serviço que provê a execução e a orquestração de *containers* na Nuvem de modo que o assinante do serviço não necessite administrar a infraestrutura e fatores como escalabilidade. Definimos esse serviço com o rótulo de orquestrador, pois, enquanto AWS Fargate e Amazon EC2 são ambientes sobre os quais os *containers* podem ser executados, o ECS gerencia a implantação, o escalonamento e a operação desses *containers*, seja no AWS Fargate ou no Amazon EC2. É possível integrar o Amazon ECS a outros serviços da AWS como Amazon CloudWatch, o Amazon S3, entre outros. Há duas opções que podem ser escolhidas como infraestrutura para a execução de tarefas:

- AWS Fargate, uma opção que segue o conceito *serverless,* em que o usuário não precisa se preocupar em configurar instâncias virtuais como as disponibilizadas em Amazon EC2.
- Amazon EC2, que oferece maior possibilidade de personalização ao permitir escolher o tipo de instância utilizada.

Um dos pontos fundamentais da metodologia seguida neste trabalho é avaliar o equilíbrio entre a facilidade de uso e a eficiência de cada serviço, ECS junto ao AWS Fargate e ECS junto ao Amazon EC2.

Para utilizar o AWS ECS, o usuário deve definir um Dockerfile, a imagem do *container*, que contém todas as dependências e o código fonte da aplicação utilizando comandos Docker que criarão o *container*, e enviá-lo a um repositório, como o Docker Hub ou o Amazon ECR. Esse repositório será o local no qual ficará hospedada a imagem. A vantagem de utilizar o AWS ECS em detrimento de utilizar o Docker Swarm, por exemplo, está na maior integração com outros serviços da AWS como monitoramento e o AWS IAM.

Após enviar a imagem ao repositório, o usuário deve definir uma *'definição de tarefa'*, que, de acordo com [Amazon Web Services Inc, 2024e], é "*como um esquema para sua aplicação. É um arquivo de texto em formato JSON que descreve os parâmetros e um ou mais contêineres que formam sua aplicação*". Em uma *'definição de tarefa'*, deve-se definir requisitos de infraestrutura, como o sistema operacional, a arquitetura, o "tamanho da tarefa" (em quantidade de vCPUs e GB de memória) e a visibilidade do(s) *container(s)* a demais serviços da AWS. Além disso, nesse passo são definidos os nomes dos *containers* que serão executados e seus mapeamentos de porta, configurações de variáveis de ambiente e opções de registro e coleta de *logs,* checagens de saúde da aplicação (*healthcheck)* e opções de armazenamento, como a adição de um volume de dados. Por exemplo, pode-se criar uma '*definição de tarefa'* que define um conjunto de recursos *X* a um *container* e um conjunto *Y* a outro *container*, que podem diferir em quantidade de vCPUs ou de memória RAM. Isso é útil, pois há casos em que as aplicações são formadas em pares de "subtarefas". Por exemplo, um *container* pode ser responsável por um serviço *web* que necessita de uma instância que lida com operações em banco de dados. Assim, pode-se criar dois *containers* e especificar portas que são visíveis entre eles para que haja a comunicação, por meio de uma '*definição de tarefa'*.

Neste trabalho, para que não haja confusão entre o conceito de "definição de tarefa", considerado como especificado no parágrafo anterior, e "tarefa", menções à "definição de tarefa" virão formatadas em itálico junto a aspas simples, deste modo: *'definição de tarefa'*. Entende-se, neste trabalho, por "tarefa", a carga de trabalho a ser executada pelo *container*.

Assim, configurada a '*definição de tarefa'* do AWS ECS, cria-se um *cluster*, que será formado pelas instâncias escolhidas pelo usuário, caso ele esteja usando o Amazon EC2, ou por uma VM desconhecida ao cliente, caso ele

esteja usando o AWS Fargate. Nessa etapa, a configuração comum à combinação do ECS, ou com EC2 ou com o Fargate, é a escolha da ativação do *CloudWatch Container Insights*, um serviço para coletar, agregar e resumir métricas e *logs* das aplicações e microsserviços conteinerizados [Amazon Web Services Inc, 2024f].

Aspectos específicos de configuração entre cada serviço são os seguintes:

- **ECS com instâncias EC2:**

  Ao criar um *cluster* com instâncias EC2, o usuário deve escolher:

  - o sistema operacional, a arquitetura e o tipo de suas instâncias, o grupo de *Auto Scaling* e o tipo de mercado (Sob-Demanda ou Spot);
  - a quantidade de instâncias idênticas que serão executadas *no cluster*;
  - o par de chaves SSH e o tamanho do volume raiz do Amazon EBS (*Elastic Block Store*);
  - a configuração da rede para as instâncias, escolhendo a *VPC* (*Amazon Virtual Private Cloud*), as sub-redes, o grupo de segurança e habilitar ou desabilitar a atribuição automática de IP público.

  Uma observação interessante é que no caso de uso de *cluster* de instâncias EC2, existirão agentes ECS nessas instâncias, os quais se comunicarão com o gerenciador do *cluster* ECS, o qual receberá requisições de execução de tarefas e as distribuirá entre os recursos disponíveis. Neste caso, a instalação desses agentes não é responsabilidade do usuário e o serviço ECS cuida desse gerenciamento.

  Por outro lado, para criar um *cluster* usando AWS Fargate, é necessário apenas selecionar a opção "AWS Fargate" na aba "Infraestrutura" da interface.

- **ECS com Fargate:**

  Apesar de ser mais simples a criação de um *cluster* com AWS Fargate, executar uma tarefa nessa configuração demanda:

  - configurar aspectos associados à rede, tais como: VPC, sub-redes, grupo de segurança e atribuição automática de IP público.

  Essa é uma configuração necessária na criação de um *cluster* Amazon EC2, mas no AWS Fargate ela só é necessária na etapa de execução de tarefas.

Finalmente, feitas as configurações, entra-se na etapa de **execução das tarefas:**

Seja utilizando o **ECS no AWS Fargate** ou o **ECS no AWS EC2**, para executar as tarefas sobre o *cluster* de *containers* criado, deve-se escolher uma configuração de distribuição dessas tarefas entre *containers*. Define-se também a configuração de implantação, que neste trabalho foi escolhida como tipo

> ‘Tarefa’, em que as tarefas são independentes e são executadas e encerradas, como trabalhos em lote. Define-se o número de tarefas a serem iniciadas. Além disso, caso a definição da tarefa contenha um volume especificado também é possível configurá-lo

Por fim, em relação à cobrança, dois modelos distintos podem ser encontrados para o Amazon ECR: ao utilizar o *AWS Fargate*, o modelo de pagamento deste serviço se aplica; ao utilizar o EC2, paga-se, em função do tempo, pelos recursos da AWS, como as instâncias e os volumes do Amazon EBS, em que não há taxas mínimas [Amazon Web Services Inc, 2024g].

AWS S3

O AWS S3 é um serviço de armazenamento de objetos de diferentes tipos, dinamicamente escalável, com dados replicados entre servidores, garantindo durabilidade e disponibilidade. Esse serviço promove a possibilidade de controlar o acesso a seus recursos por meio do IAM, versionar seus objetos e integrá-lo ao CloudWatch, CloudTrail, entre outros.

Há classes diferentes de armazenamento de acordo com a frequência de acesso aos dados, e a cobrança do serviço é em conformidade com qual classe está sendo utilizada. Além disso, o preço também depende dos tamanhos dos objetos e do tempo de armazenamento. Para cada classe, há um preço por GB por mês. Há também taxas de consumo por solicitação para o uso de PUT, COPY e regras de ciclo de vida (com as quais o usuário pode programar a migração de uma classe a outra). Essas taxas variam o valor por região e são cobradas em número de solicitações de PUT, COPY ou transição de ciclo de vida [Amazon Web Services Inc, 2024h].

AWS Lambda

O AWS Lambda é um serviço de computação *serverless* que permite a execução de uma função centrado em resposta a eventos, como mudanças em um *bucket* S3 ou mesmo a captura de uma determinada expressão em um *stream* de *logs* do *Amazon CloudWatch*. Ela permite que o usuário foque apenas no código, que deve ser implementado em funções que podem ser escritas em diversas linguagens como Javascript, Python e Java. Cada função Lambda é configurada com permissões específicas e pode ser integrada facilmente com outros serviços da AWS, como o RDS (*Relational Database Service*), por exemplo. Contudo, cada função lambda deve terminar de executar dentro do tempo máximo de 15 minutos.

O modelo de precificação da AWS Lambda é dado com base no número de solicitações para as funções do usuário e a duração necessária para a execução do código, medida em milissegundos. O preço depende da quantidade de

memória que o usuário aloca para a função. Essas funções podem ser executadas em processadores baseados em arquiteturas x86_64 ou ARM64, o que deve ser escolhido pelo usuário e também influencia no valor pago. A duração cobrada é calculada a partir do momento em que o código começa a ser executado até ele encerrar, arredondando para o milissegundo mais próximo [Amazon Web Services Inc, 2024i].

3.2. Aplicação Estudo de Caso *Multi-Platform Architecture for Sequence Aligners* (MASA)

A aplicação escolhida como estudo de caso para a análise do gerenciamento de recursos da nuvem foi a ferramenta de alinhamentos de sequências genéticas ***Multi-Platform Architecture for Sequence Aligners*** (MASA) [De O. Sandes et al., 2016]. Isso porque os alinhamentos genéticos assumem papel importante na ciência, pois é por meio deles que se torna possível compreender a evolução das espécies ao longo do tempo. Consequentemente, no âmbito médico, o alinhamento de sequências genéticas proporciona uma base sólida para a identificação de mutações associadas a doenças hereditárias. Ademais, uma serventia desse recurso que recentemente se mostrou importante devido à pandemia do vírus SARS-CoV-2 se refere a sua utilização para estudar novas variantes de vírus. Por meio de alinhamentos, torna-se viável entender o grau de infecção que um vírus pode causar, a facilidade com que se espalha, a gravidade dos sintomas e a eficácia para as vacinas. Assim, a aplicação é de interesse de profissionais da Bioinformática e impacta positivamente a sociedade.

Além da relevância do problema que aborda, o MASA é considerado uma aplicação de Computação de Alto Desempenho (HPC, do inglês *High Performance Computing*), sendo capaz de realizar alinhamentos de sequências com mais de 200 milhões de nucleotídeos em variadas arquiteturas de *hardware*. Selecionou-se a versão MASA-OpenMP [De O. Sandes et al., 2016], projetada para aproveitar os vários núcleos em um servidor. MASA-OpenMP alinha sequências de DNA ou RNA utilizando uma variação do algoritmo clássico de Smith-Waterman [Myers and Miller, 1988] para encontrar o alinhamento de solução ótima, empregando um método de *pruning* com o objetivo de reduzir os custos computacionais dos alinhamentos. A aplicação possui complexidade quadrática de tempo e linear de espaço de memória [De O. Sandes et al., 2016].

Inicialmente, em colaboração com o aluno de iniciação científica Gabriel Carneiro Mills, foram feitos estudos com o intuito de entender o comportamento da execução do MASA-OpenMP, principalmente no que diz respeito à previsibilidade do tempo de execução de alinhamentos. Esse estudo foi realizado por meio do ambiente de nuvem Amazon EC2, com a instância otimizada para computação c7g.xlarge, cujas características principais são: 4vCPUs, 8 GiB de memória RAM e um volume SSD do tipo GP3, Ubuntu Jammy 22.04, situada na região *us-east-1*, Norte da Virgínia. As sequências alinhadas foram retiradas do banco público de dados biológicos *GenBank* do *National Center for Biotechnology Information* (NCBI). Selecionou-se 18 sequências iniciais, formou-se todos os pares possíveis entre essas sequências (incluindo o alinhamento de uma sequência com ela

mesma) e cada um desses pares foi alinhado de forma sequencial três vezes. Os pares foram formados de modo que a ordem das sequências de entrada não importasse, isto é, a entrada (‘seq1.fasta’, ‘seq2.fasta’) foi gerada, mas a entrada (‘seq2.fasta’, ‘seq1.fasta’) não foi. À medida que esses alinhamentos foram executados, construiu-se um arquivo ‘.csv’ contendo o tempo de cada alinhamento, a data, a hora, a porcentagem de *pruning* e demais dados como métricas de otimalidade da solução (*matches, mismatches, entre outras*).

O conjunto de sequências possuía alto grau de disparidade entre seus elementos, contendo vírus, bactérias, algas, leguminosas, entre outras classes de sequências. Ele também continha variações artificiais de SARS-CoV-2, geradas pela manipulação de sequências reais para atingir tamanhos determinados. Essa foi a estratégia abordada visando à compreensão da robustez da solução da proposta por [Sodré et al., 2022] para o escalonamento de diversos alinhamentos sobre uma única instância. O escalonamento proposto é feito sob a premissa de que esses diversos alinhamentos possuem tempos de execução semelhantes. Buscou-se verificar, então, em que medida o tamanho das sequências e a similaridade biológica entre elas impactam no tempo de execução e se há outros fatores que influenciam nesse sentido, e também se esse tempo de execução é passível de ser previsto.

Foi realizada a média do tempo das três execuções de cada par de sequências alinhadas, a fim de diminuir o erro associado às variações do ambiente de execução. Dessa forma, por meio da linguagem R e dos dados registrados no arquivo ‘.csv’, foram criados gráficos que correlacionam o produto do tamanho das sequências ao tempo médio de alinhamento que tal par demandou. A escolha dessa correlação se deu devido à complexidade de pior caso do algoritmo *Smith-Waterman*, O(m . n), em que *m* e *n* representam os tamanhos de cada uma das sequências. Esses gráficos foram gerados com o intuito de realizar uma previsão de tempo de execução com base nas sequências de entrada e verificar de quais fatores esse tempo de execução seria dependente. Para analisar o fator de similaridade entre as sequências, foram criadas duas classes de alinhamentos par-a-par: *alinhamentos de similaridade total* (isto é, alinhamentos de sequências iguais) e *de similaridade não total* (isto é, alinhamentos de sequências diferentes). Dada essa separação, foram criadas duas retas preditoras para cada classe de alinhamento, representadas por cores diferentes nos gráficos e construídas por meio de *Regressão Linear*, apresentadas na Figura 3 da Seção 4.

Além do tamanho e das classes observadas de alinhamentos, a inclinação das retas preditoras em relação à taxa de *pruning* também foi estudada, devido ao *speed-up* que o *pruning* promove. Para tal, um gráfico que relaciona o coeficiente angular da reta preditora à porcentagem de *pruning* foi gerado, possuindo uma reta que aproxima essa relação por Regressão Linear. Entretanto, durante esse processo, foi evidenciada a falta de sequências que produzissem *pruning* dentro do intervalo de [5, 40] %, para estabelecer uma relação válida entre a porcentagem de *pruning* e a inclinação da reta do tempo de execução. Uma vez que alinhamentos entre sequências similares possuíam menor tempo de execução e maior porcentagem de *pruning*, decidiu-se explorar a similaridade entre as

sequências a fim de povoar o gráfico que relaciona o coeficiente angular da reta preditora à porcentagem de *pruning*. Dessa forma, foram geradas sequências sintéticas com o formato [X_S, Y_S, Z_S] que seriam alinhadas a uma sequência de referência retirada do conjunto inicial de sequências, nomeada aqui como 'sequência base *Y'*. Isso para que a porcentagem de similaridade entre a sequência base *Y* e a sequência sintética fosse controlada. O formato seguido, [X_S, Y_S, Z_S], é definido da seguinte forma:

- X_S é uma subsequência**, começando do início,** de uma sequência qualquer *X* **diferente da sequência base *Y***
- Y_S é uma subsequência**, começando do início,** da sequência base *Y*
- Z_S é uma subsequência**, começando do início,** de uma sequência qualquer *Z* **diferente da sequência base *Y***
- O tamanho desta sequência sintética é similar ao tamanho da sequência base *Y*

É importante dizer que as subsequências X_S e Z_S poderiam ser vazias. Foram geradas sequências sintéticas variando tanto o tamanho da subsequência Y_S quanto o seu ponto de início na sequência sintética, com o passo de 10% em relação ao tamanho da sequência *Y.*

Estipulando-se que blocos de nucleotídeos equivalem a 10% do tamanho da sequência *Y*, ao pensar em um uma tupla do tipo (*bi* , *tam*(Y_S))', onde *bi* é o índice do bloco inicial da subsequência Y_S e *tam* (Y_S) é o número de blocos de 10% de Y_S, foram formadas sequências com estas configurações:

[(0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 10),
(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9),
(2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8),
(3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7),
(4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6),
(5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5),
(6, 0), (6, 1), (6, 2), (6, 3), (6, 4),
(7, 0), (7, 1), (7, 2), (7, 3),
(8, 0), (8, 1), (8, 2),
(9, 0), (9, 1)]

Assim, para a configuração de (*bi* , *tam*(Y_S)) = (3, 2), há a disposição de sequências representada na Figura 1, onde cada bloco de cada sequência equivale à 10% do tamanho da sequência de base. Neste exemplo, as três sequências de entrada (*X, Y* e *Z*) têm o mesmo tamanho. Já que (*bi* , *tam*(Y_S)) = (3, 2), a posição dos 20% iniciais de Y_S na sequência sintética é 30%, os três primeiros blocos dessa sequência sintética serão preenchidas com os

três primeiros blocos de *X* (X_S) e os últimos blocos restantes serão preenchidos com os respectivos primeiros blocos de *Z*.

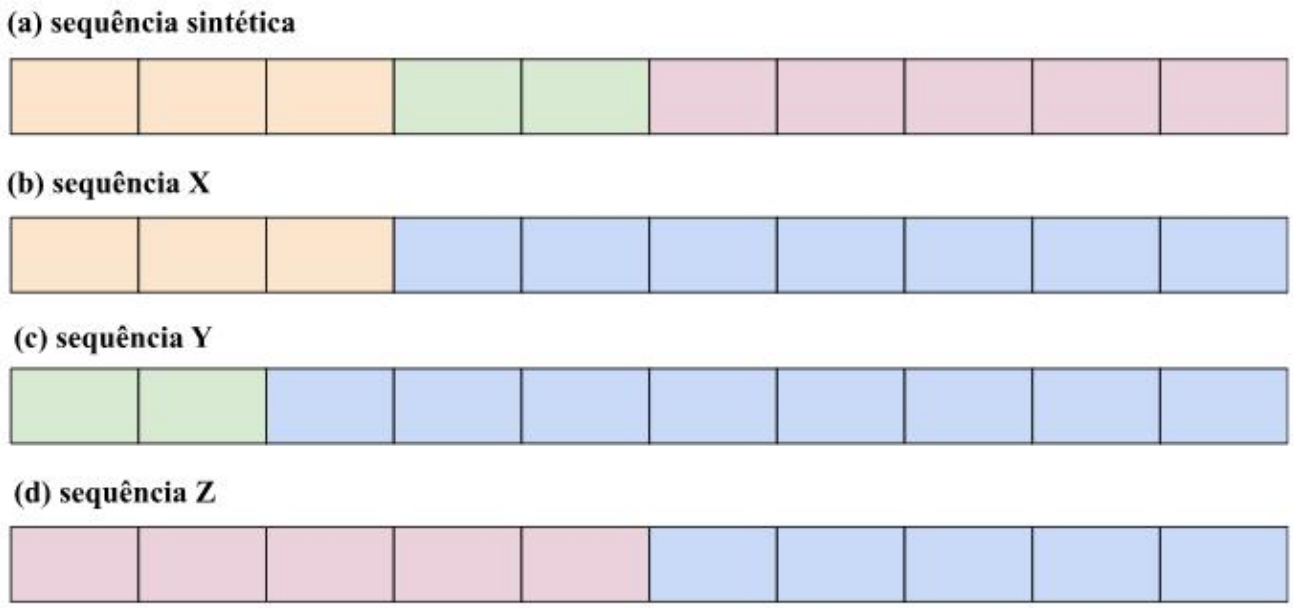

Figura 1: Regra de formação da sequência sintética (a) por meio das sequências (b), (c) e (d) , em que cada caixa representa 10% do tamanho total da sequência base *Y*. Formato [X_S, Y_S, Z_S], em que X_S: caixas amarelas, Y_S: caixas verdes e Z_S: caixas rosas.

As sequências geradas por meio desse método foram adicionadas às sequências que já haviam sido utilizadas e o experimento foi refeito. Os principais resultados advindos dessas observações estão apresentados na Figura 4 da Seção 4, em que os resultados são apresentados.

Nessa etapa inicial do comportamento da aplicação, além da avaliação de similaridade e *pruning*, estudos anteriores de [Sodré et al., 2022] também foram revisados para compreender o comportamento em relação à utilização de memória do MASA-OpenMP, representado pela área em azul da Figura 2.

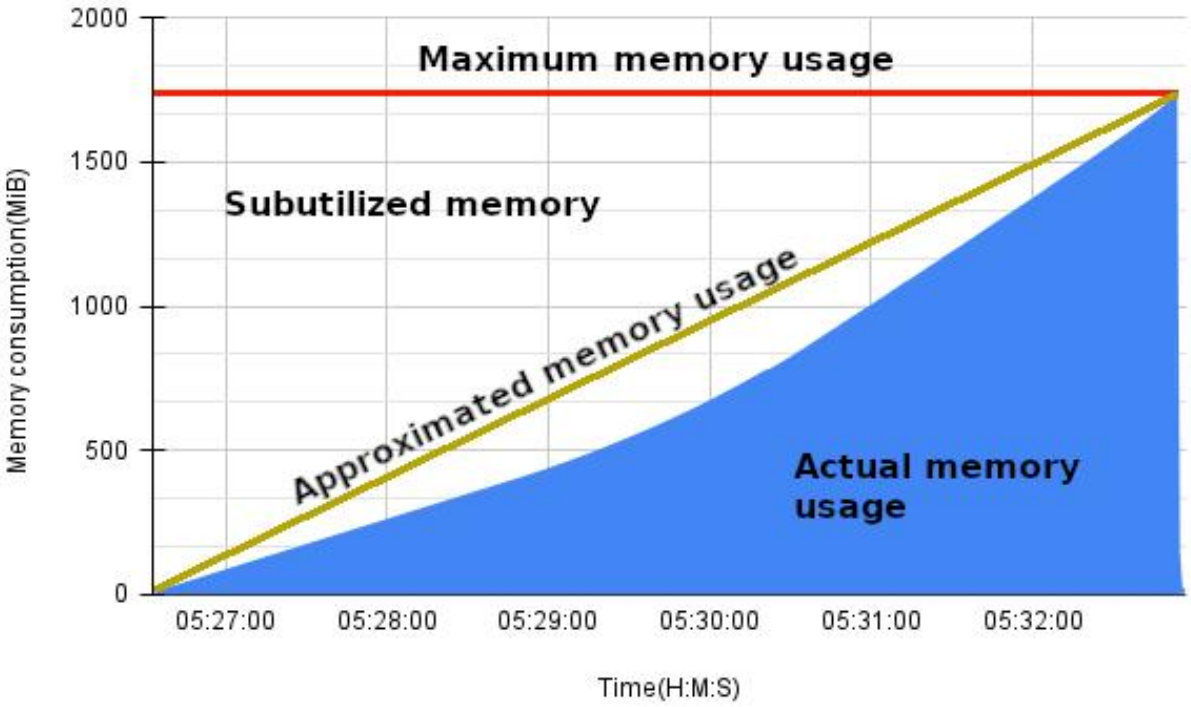

Figura 2: Padrão de consumo de memória de um alinhamento. **Fonte:** Figura 1, [Sodré et al., 2022].

Essas análises viabilizaram o entendimento do comportamento da aplicação em relação ao tempo e memória e, a partir desses estudos, o foco foi voltado ao estudo inicial dos *containers* nas ferramentas disponibilizadas pela AWS.

3.3. Metodologia Seguida neste Projeto para Utilização de Ferramentas de Nuvem em *Container*

Experimentos semelhantes àqueles relatados na versão parcial deste relatório foram realizados, agora com o conhecimento de ferramentas que facilitam a coleta de dados para a estimativa de custo das tarefas, como a biblioteca 'boto3', o SDK (*Software Development Kit*) oficial da AWS para a linguagem Python. Por meio dele, foram implementados meios de automação e interação com serviços da AWS que serão descritos nesta seção. A aplicação estudo de caso se manteve sendo o MASA-OpenMP considerando agora tamanhos específicos para sequências de entrada e números de *threads* determinados. Os conjuntos de sequências a serem alinhados permanecem sendo armazenados em um *bucket* no AWS S3, assim como os dados de saída dos alinhamentos, que são enviados ao *bucket* por meio do AWS CLI (*Command Line Interface*). Além disso, os dados referentes às estimativas de custos também são armazenados nessa ferramenta.

A análise do comportamento da aplicação foi realizada sobre os seguintes serviços: **[ECS no AWS Fargate]** e **[ECS no Amazon EC2].** Para coleta de métricas como tempo de execução, memória consumida/necessária, o

framework de monitoramento utilizado foi implementado por [Sodré, 2022; Sodré, 2024]. A seguir são apresentadas as configurações utilizadas para cada serviço.

- **[ECS no AWS Fargate]** e **[ECS no Amazon EC2]:** para cada um dos casos, uma mesma imagem de *container* foi gerada utilizando um Dockerfile, com a imagem base ubuntu:22.04, a aplicação MASA-OpenMP e as dependências necessárias, como Python, AWS CLI, ferramentas de compilação, entre outras. Essa imagem foi enviada a um repositório privado AWS ECR.
- **[ECS em cluster AWS Fargate]:** Um *cluster* construído para utilização com AWS Fargate que não possui o serviço de monitoramento *Container Insights* da AWS ativado, visto que para monitorar recursos utilizamos o *framework* de [Sodré, 2022; Sodré, 2024]. Vale mencionar que ao criar este *cluster*, seus detalhes de infraestrutura, como em que tipo de instância a aplicação será executada, são abstraídos por ser uma proposta *serverless*.
- **[ECS em cluster Amazon EC2]:** Um *cluster* para instâncias do EC2, que foi definido utilizando o tipo de instância c7g.4xlarge (que possui 16 vCPUs e 32 GiB), com o sistema operacional Amazon Linux 2 (Kernel 5.10/arm64) e com o número máximo de uma instância a ser executada sobre esse "*cluster"*. Também não possui o serviço de monitoramento *Container Insights* da AWS ativado, já que para isso está sendo utilizado o mesmo *framework* utilizado no Fargate.

As '*definições de tarefa'* especificadas tanto para o AWS Fargate quanto para o Amazon EC2 possuem um mesmo tamanho, de 8 vCPUs e 16 GB, e um único *container*, associado à imagem armazenada no ECR. Dessa forma, as '*definições de tarefa'* diferem apenas em requisitos de infraestrutura (como devem ser executadas no AWS Fargate ou no Amazon EC2). O *container* de cada '*definição de tarefa'* também é especificado de forma idêntica: ele possui uma variável de ambiente chamada INPUT_SET, que representa qual dos grupos de sequências será utilizado para realizar os alinhamentos *par-a-par*. O *container* também possui as variáveis 'THREADS' e 'RODADA', que representam respectivamente a quantidade de *threads* que a aplicação utilizará e um contador. Detalhes sobre o motivo da existência dessas variáveis serão expostos posteriormente.

Com base nas configurações descritas acima, a usabilidade e a eficiência em termos de tempo de execução e custo, em ambos os serviços, **[ECS no AWS Fargate]** e **[ECS no Amazon EC2]**, foram analisadas sob o experimento descrito a seguir.

Configurações de entrada da tarefa

Grupos de sequências genéticas foram definidos com base nos seguintes tamanhos: 100k, 200k, 300k, 400k e 500k, em quantidade de nucleotídeos. Cada grupo de sequências foi composto por três sequências com o respectivo tamanho. Ainda, as sequências dentro de cada um dos grupos foram escolhidas arbitrariamente do banco público de dados biológicos *GenBank* do NCBI. Não houve preocupação quanto à proximidade genética entre elas, pois foi dado enfoque para a análise do consumo de recursos da aplicação, uma vez que não possuímos conhecimento neste domínio.

O MASA-OpenMP foi executado considerando um par de sequências de um dado grupo, seguindo a seguinte variação no número de *threads*: 1, 2, 4 e 8 *threads*. Dessa forma, o comportamento da aplicação também foi avaliado quanto ao paralelismo.

À nível de notação do experimento, a tupla (*g, t*) é denotada como a "configuração" associada a uma execução da tarefa. Essa tarefa tem como entrada um grupo de sequências pertencente ao grupo *g*, onde $g \in$ {100k, 200k, 300k, 400k, 500k} e uma quantidade de *threads* $t \in \{1, 2, 4, 8\}$. Assim, a tarefa realizará alinhamentos par-a-par em sequência de forma que cada alinhamento MASA-OpenMP terá como entrada um par de sequências pertencentes ao grupo *g* e será executado com *t threads*. Com esses valores *g* e *t*, um total de 20 configurações podem ser avaliadas em termos de: tempo de execução, custo e eficiência, tanto para o caso de **ECS no AWS Fargate** quanto de **ECS no Amazon EC2** (avaliações distintas).

Descrição da computação da tarefa

O processamento que o *container* executado no **AWS Fargate** e aquele executado em uma instância do **Amazon EC2** é idêntico e possui uma fase inicial, principal e final. Começaremos especificando a parte principal da tarefa.

A parte principal da tarefa a ser executada, dada uma configuração de entrada (*g, t*), consiste na execução, com *t threads* (isto é, com a variável de ambiente OMP_NUM_THREADS=*t*), de alinhamentos individuais par-a-par. Esses alinhamentos par-a-par são executados um após o outro, de modo que, para um grupo *g* de sequências, os pares de sequências de entrada de cada alinhamento MASA-OpenMP são definidos da seguinte forma:

> *Seja (1, 2, ..., n) uma ordenação das sequências do grupo g.*
> *Cada i-ésima sequência desta ordenação será alinhada, em alinhamentos individuais par-a-par, com cada j-ésima sequência em que* $i \leq j$.

Desse modo, cada tarefa (associada a um *container*), cuja configuração de entrada contém o grupo *g* que possui *n* sequências, pode ter sua quantidade de alinhamentos contabilizada deste modo:

Seja *T(n)* o total de alinhamentos obtidos pela formação de pares (*i, j*), com $i \leq n$ e $j \leq n$ em que $i \leq j$.

Para $i = 1$, ocorrerão *n* alinhamentos: (1, 1), (1, 2), …, (1, n).

Para $i = 2$, ocorrerão *n-1* alinhamentos : (2, 2), (2, 3), …, (2, n).

Para $i = 3$, ocorrerão *n-2* alinhamentos : (3, 3), (3, 4), …, (3, n).

…

Para $i = n$, ocorrerá um alinhamento: (n, n).

Assim,

$$T(n) = n + (n-1) + (n-2) + \ldots + (n - (n-2)) + (n-(n-1)).$$

Logo,

$$T(n) = \sum_{k=0}^{n-1} n - k$$

$$= \sum_{k=0}^{n-1} n - \sum_{k=0}^{n-1} k =$$

$$= n^2 - \sum_{k=0}^{n-1} k$$

Portanto, temos:

$$T(n) = n^2 - \frac{n(n-1)}{2}.$$

Assim, como neste trabalho cada grupo *g* possui três sequências, cada tarefa executa T(3) = 6 alinhamentos.

As partes inicial e final da tarefa consistem respectivamente apenas no pré-processamento e pós-processamento da computação principal descrita acima. Abaixo, é apresentado o passo a passo de execução para uma tarefa de configuração (*g, t*). A fase inicial é representada pelos passos 1 e 2, a principal pelo passo 3 e a final pelos passos 4, 5 e 6.

1. Captura-se todas as sequências '.fasta' da pasta que contém as sequências *g*, localizada em um *bucket* no S3, por meio do AWS CLI (*Command Line Interface*), baixando-as para o *container*;
2. Captura-se o ambiente de execução (Fargate ou EC2) por meio da variável $AWS_EXECUTION_ENV e cria-se um arquivo '.csv' de dados no qual serão armazenados dados de cada alinhamento a ser executado;
3. Executa-se o programa que monitora o consumo de memória do *container* que executa o *script* que realiza a sequência de alinhamentos através do MASA-OpenMP, correspondente à parte principal da tarefa, na qual os 6 alinhamentos para a configuração (*g*, *t*) são executados. Durante a execução, o

monitor agrega no arquivo criado no passo anterior, que é preenchido com os seguintes dados de cada alinhamento:

- o dia e hora;
- as sequências de entrada e seus respectivos tamanhos;
- o pico de memória da execução e o tempo demandado para atingir tal pico;
- o tempo de execução deste alinhamento.

Paralelamente, é gravado no arquivo *memória total utilizada*, criado neste passo, o seguinte dado:

- consumo de memória total desta tarefa (que executa a sequência de alinhamentos).

4. Gera-se o gráfico *consumo de memória* × *tempo* relacionado à execução do passo 3 a partir do arquivo *memória total utilizada*;
5. Cria-se uma subpasta no bucket em "EC2" ou "FARGATE" (dependendo do ambiente no qual a tarefa foi executada) associada à execução do passo 3, para onde são enviados o arquivo de dados preenchido, o arquivo de memória total utilizada e o gráfico de consumo de memória gerado, e onde será armazenado o tempo cobrado por esta tarefa posteriormente;
6. Consulta-se a API AWS com requisições HTTP ao endereço $ECS_CONTAINER_METADATA_URI_V4 a fim de coletar dados como o *task_arn* e o *cluster* desta tarefa. Essa consulta ocorre para compor um *payload* que será útil à computação de uma função Lambda explicada posteriormente. Imprime-se uma mensagem final junto a este *payload.*

Estratégias de execução do experimento

Foram avaliados tempos de execução, consumo de memória e custo para os casos **[ECS no AWS Fargate]** e **[ECS no Amazon EC2]** em cada uma das configurações para (g, t) pertencentes a

{(100k, 1), (100k, 2), (100k, 4), (100k, 8),
(200k, 1), (200k, 2), (200k, 4), (200k, 8),
(300k, 1), (300k, 2), (300k, 4), (300k, 8),
(400k, 1), (400k, 2), (400k, 4), (400k, 8),
(500k, 1), (500k, 2), (500k, 4), (500k, 8)}.

Para que os resultados deste experimento fossem mais representativos do desempenho padrão da aplicação, cada configuração foi executada três vezes em cada serviço, a fim de coletar a média de tempo, custo e memória resultantes destas execuções.

Cada um dos serviços foi avaliado em termos de custo de acordo com o modelo de cobrança especificado na seção 'Modelos de cobrança do Amazon ECS' [Amazon Web Services Inc, 2024g]:

- **[ECS no AWS Fargate]**

  "*Com o AWS Fargate, você paga pela quantidade de recursos de memória e vCPU solicitados pela sua aplicação em contêineres. Os recursos de memória e vCPU são calculados a partir da hora em que suas imagens de contêiner são extraídas até o término da tarefa do Amazon ECS, arredondados para o segundo mais próximo. Há uma cobrança mínima de um minuto.*"

- **[ECS no Amazon EC2]**

  "*Não há cobrança adicional para o tipo de execução do Amazon EC2. Você paga pelos recursos da AWS (como instâncias do Amazon EC2 ou volumes do Amazon EBS) criados para armazenar e executar sua aplicação. O pagamento é feito conforme o uso. Não há taxas mínimas nem compromissos antecipados*."

Com isso, durante a formulação deste trabalho, notou-se a necessidade de capturar o tempo de execução de uma tarefa de forma externa ao *container*, uma vez que capturar o momento em que as imagens de *container* são extraídas não seria possível de forma interna. Para capturar esse dado e o momento em que a tarefa terminou, o AWS Lambda se mostrou uma opção ideal, dado que se adequa a computações que ocorrem em resposta a eventos; no caso, o término de uma tarefa. O AWS Lambda também assumiu um papel fundamental para realizar as três execuções de cada uma das configurações (g, t). Para isso, as variáveis de ambiente $INPUT_SET, $THREADS e $RODADA foram utilizadas, de forma que $INPUT_SET variou de 100k a 500k, $THREADS nos valores de 1, 2, 4 e 8 e $RODADA de 1 a 3, fazendo com que, ao final de todas as execuções de tarefas, um programa calculasse as médias de tempo demandado, pico de memória e estimasse o custo relacionado a essa média de tempo para cada configuração (g, t).

Dessa forma, foram criadas duas funções Lambda em Python:

Função 1

Responsável primeiro por capturar o tempo total entre o momento em que a extração da imagem do contêiner começou e o momento em que a execução da tarefa foi interrompida e armazená-lo no objeto S3 associado àquela execução. Após isso, incrementa as variáveis $INPUT_SET, $THREADS e $RODADA e invoca a execução de uma nova tarefa, caso esta exista, e, caso contrário, invocar a execução da **Função 2**.

Função 2

Responsável por coletar do S3, dada uma configuração (g, t) e um serviço $s \in$ {Fargate, EC2}, todos os tempos de execução demandados pelas três tarefas executadas com (g, t) no serviço s e calcular o tempo médio de execução. Por meio desse tempo médio, estima-se o custo médio de execução dessa

configuração de acordo com o serviço *s*. Enfim, coleta-se todos os arquivos de consumo de memória para essa configuração (*g*, *t*), selecionando o pico de memória dentre cada arquivo de dados, calculando-se o pico de memória médio para aquela configuração. O produto desta função é um arquivo que contém, em cada linha, para cada configuração, os seguintes dados:

"100K-4threads:: Tempo: 00:00:41.55, Custo estimado: $0.00669 USD, Pico memória: 108.71 MB"

O gatilho para que a **Função 1** seja executada se dá por meio da captura de um *log* específico, verificado no *stream* de *logs* no AWS CloudWatch de cada '*definição de tarefa'*. Como relatado no passo 6 do *'passo a passo de execução da tarefa'*, quando esta termina é impressa uma mensagem junto a um *payload*, direcionada a seu *stream* de *logs*. Este *log*, após o termo "FIM:" segue um formato JSON, deste modo:

```
"FIM:
{
        "task_arn": "id da tarefa encerrada",
        "cluster":"id do cluster no qual a tarefa encerrada foi executada",
        "ambiente": "FARGATE || EC2",
        "dia_inicial": "29_08_2024",
        "hora_inicial": "17:18:02",
        "input_set": "500",
        "rodada": "3",
        "threads": "8"
}"
```

A passagem do *payload* apresentado no JSON e passado para a **Função 1** é indispensável, pois é por meio dela que serão incrementados $INPUT_SET, $THREADS e $RODADA para determinar a próxima tarefa a ser executada. Já as informações relacionadas ao *cluster* são utilizadas para capturar os dados de tempo s ser cobrado.

Em ambas funções, a biblioteca oficial da AWS para Python 'boto3' foi utilizada para interagir com os serviços da AWS diretamente a partir do código em Python contido na função Lambda. Para a consulta de tempos de execução considerados para a cobrança da AWS, foi utilizado o método 'describe_tasks' do cliente 'boto3' do ECS da API AWS, cuja resposta forneceu os seguintes dados, segundo [Amazon Web Services Inc, 2024j]:

- 'pullStartedAt': O carimbo de data/hora Unix para o momento em que a extração da imagem do contêiner começou.
- 'executionStoppedAt': O carimbo de data/hora Unix da hora em que a execução da tarefa foi interrompida.

O envio do arquivo que contém diferença calculada entre esses *timestamps* ao objeto S3 relacionado à execução em questão também foi viabilizada pelo cliente 'boto3', por meio do método 'put_object'. Por fim, o cliente 'boto3' também facilitou o lançamento de novas tarefas no ECS, por meio do método 'run_task'.

Contabilização de recursos utilizados e custos

Foram realizadas tentativas de utilizar 'pullStartedAt' e 'executionStoppedAt' como entrada para os parâmetros TimePeriod.Start e TimePeriod.End do método GetCostAndUsage da API AWS, requisitado via 'boto3' para contabilizar o custo de cada execução da tarefa. Todavia, a granularidade proporcionada não se mostrou fina o suficiente, sendo a menor possível a granularidade horária. Sendo assim, o tempo da execução de cada tarefa foi calculado pela diferença entre 'executionStoppedAt' e 'pullStartedAt' e enviada ao *bucket* S3. Com esses dados, para cada configuração (grupo, *threads*) foi realizada uma média de tempo de execução para os casos **[ECS no AWS Fargate]** e **[ECS no Amazon EC2].** Dessa forma, a estimativa de custo foi calculada para cada serviço a partir dessa média de tempo representada por 'segundos' nas descrições abaixo.

- **Custo Estimado para [ECS no AWS Fargate]**

Dado que o "tamanho da tarefa" foi definido com 8 vCPUs e 16 GB de memória, o cálculo se deu assim:
Se segundos > 60:

$$custo\_ecs\_fargate(segundos) = custo\_memoria(segundos) + custo\_vcpus(segundos)$$

Caso contário:

$$custo\_ecs\_fargate(segundos) = custo\_memoria(60) + custo\_vcpus(60)$$

Onde:

$$custo\_memoria(segundos) = segundos \times 16 \times custo\_gb\_por\_hora \,/\, 3600$$

$$custo\_vcpus(segundos) = segundos \times 8 \times custo\_vcpu\_por\_hora \,/\, 3600$$

Os valores de custo_gb_por_hora e de custo_vcpu_por_hora que foram utilizados datam de 28/08/2024 para a região '*us-east-1*', Norte da Virgínia, utilizando SO/aquitetura como Linux/ARM, e são os seguintes, segundo [Amazon Web Services Inc, 2024d]:

- *custo_gb_por_hora* = 0.00356 USD

- *custo_vcpu_por_hora* = 0.03238 USD

- **Custo Estimado para [ECS no Amazon EC2]**

Dado que o "*cluster*" utilizado foi definido para utilizar apenas uma instância EC2 do tipo c7g.4xlarge, o cálculo se deu assim:

Se segundos > 60:

custo_ecs_ec2(segundos) = segundos × *custo_instancia*(segundos)

Caso contrário:

custo_ecs_ec2(segundos) = 60 × *custo_instancia*(segundos)

Onde:

custo_instancia(segundos) = *custo_por_hora*('c7g.4xlarge') / 3600

O valor de custo por hora para esta instância utilizado também data de 28/08/2024 para a região '*us-east-1*', Norte da Virgínia, e é o seguinte, segundo [Amazon Web Services Inc, 2024b]:

- *custo_por_hora*('c7g.4xlarge') = 0.58 USD

Neste cálculo, vê-se que o custo dos volumes EBS do *cluster* foi desconsiderado, dado que é ínfimo em comparação com o valor cobrado pela instância escolhida. Para explicitar isso, o valor do volume EBS SSD de uso geral (gp3) para a região '*us-east-1*', Norte da Virgínia, para o dia 30/08/2024 é de 0,08 USD/GB-mês [Amazon Web Services Inc, 2024k], e, na verdade, a instância do *cluster* e seu volume são encerrados poucos minutos após o experimento terminar.

Um fato a ser apontado é que foram feitas tentativas para utilizar uma máquina de menor porte, 'c7g.2xlarge', que possui 8 vCPUs e 16 GiB, o que diminuiria o custo de utilização do EC2, já que a máquina possui menor valor de custo por hora e possuiria exatamente os recursos necessários para o "tamanho da tarefa" definido. Entretanto, ao tentar lançar uma tarefa de 8 vCPUs e 16 GB nesta máquina, essa tarefa fica em estado de "em provisionamento" constantemente, provavelmente por causa de um *overhead* de outros processos utilizados pelo ECS.

Vale salientar que o 'Caso contrário' de **Custo Estimado para [ECS no AWS Fargate]** e **Custo Estimado para [ECS no Amazon EC2]** ocorre devido à cobrança mínima de um minuto de execução, documentadas respectivamente em [Amazon Web Services Inc, 2024g] e [Amazon Web Services Inc, 2024b].

4. RESULTADOS

Esta seção exibe os principais resultados obtidos. A Seção 4.1 contém gráficos que reportam o quanto o tempo de execução é influenciado pelo tamanho da sequência, pela semelhança e pela porcentagem de *pruning*, conforme a metodologia descrita na Seção 3.2. Já a Seção 4.2 possui resultados relacionados à execução dos containers na AWS, conforme a metodologia descrita na Seção 3.3.

4.1. Resultados do Estudo Inicial de Comportamento da Aplicação Estudo de Caso MASA

Foi produzido o gráfico da Figura 3, que correlaciona o produto dos tamanhos das sequências de entrada ao tempo de execução do alinhamento. A reta azul da Figura 3 foi gerada com base nos pontos de similaridade total (isto é, alinhamentos de sequências iguais) e a reta vermelha com base nos pontos de similaridade não total (isto é, alinhamentos de sequências diferentes). É possível notar que as retas aproximam bem o comportamento do tempo de execução de ambas classes de pontos.

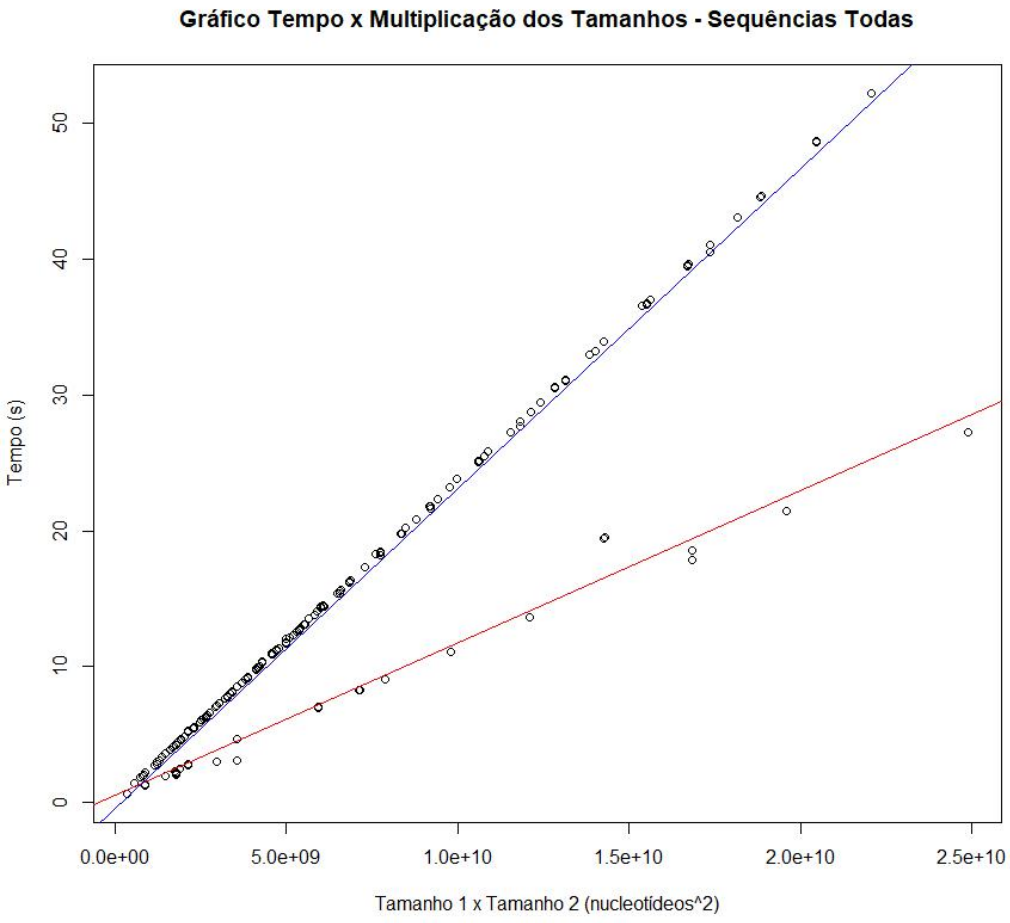

Figura 3: Alinhamentos das 18 sequências iniciais. A reta azul é formada com base em alinhamentos de semelhança total e a reta vermelha formada com base em alinhamentos de semelhança não total.

Além disso, o gráfico da Figura 4 foi gerado, com base na execução que recebeu as 18 sequências iniciais somadas às sequências sintéticas geradas no formato [X_S, Y_S, Z_S], em que é possível observar pontos sobre boa parte do intervalo de porcentagem de *pruning* (que costumeiramente não passa de 60%). A geração desses pontos possibilitou a percepção de uma relação confiável entre a porcentagem de *pruning* e a inclinação da reta do tempo de execução.

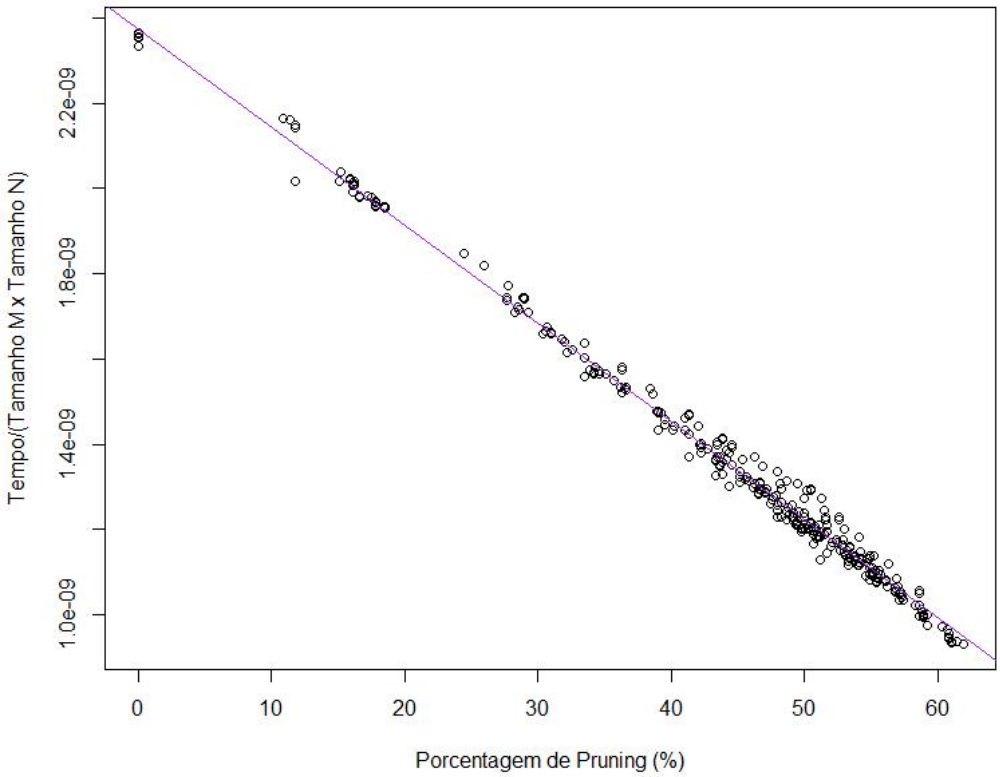

Figura 4: Inclinação da reta do tempo de execução formada com base no produto dos tamanhos das sequências em função da porcentagem de *pruning*, com a adição das sequências sintéticas geradas.

4.2. Resultados do Estudo dos Serviços em Nuvem de *Container*

Consoante os tempos a serem considerados para a cobrança, capturados pela **Função 1** do AWS Lambda a cada execução da tarefa, a Tabela 1 foi construída de modo que cada célula dos campos de tempo contém uma média entre as três execuções daquela configuração no respectivo serviço, enquanto o custo estimado é calculado por esta média de tempo, em segundos, como especificado em **Custo Estimado para [ECS no AWS Fargate]** e **Custo Estimado para [ECS no Amazon EC2]**.

Tabela 1: Comparação de Tempo e Custo Estimado entre Fargate e EC2.
Valores do EC2 calculados para a instância c7g.4xlarge.

Configuração		Fargate		EC2	
Tamanho	Threads	Tempo (HH:mm:ss.SS)	Custo Estimado (USD)	Tempo (HH:mm:ss.SS)	Custo Estimado (USD)
100k	1	00:03:23.31	0.01785	00:02:05.95	0.02029
	2	00:01:48.42	0.00952	00:01:05.14	0.01050
	4	00:01:15.72	0.00665	**00:00:41.55**	**0.00967**
	8	**00:00:56.30**	**0.00527**	**00:00:23.58**	**0.00967**
200k	1	00:10:50.09	0.05706	00:07:08.41	0.06902
	2	00:04:44.94	0.02501	00:03:45.27	0.03629
	4	00:03:06.87	0.01640	00:02:00.11	0.01935
	8	00:02:13.93	0.01176	00:01:11.53	0.01152
300k	1	00:18:53.35	0.09948	00:16:03.00	0.15515
	2	00:08:40.81	0.04572	00:08:16.33	0.07996
	4	00:06:13.39	0.03278	00:04:14.39	0.04099
	8	00:03:47.93	0.02001	00:02:19.07	0.02241
400k	1	00:37:10.22	0.19576	00:28:17.39	0.27347
	2	00:19:19.07	0.10174	00:14:21.51	0.13880
	4	00:10:13.38	0.05384	00:07:21.09	0.07106
	8	00:06:13.75	0.03281	00:03:53.00	0.03754
500k	1	00:50:36.15	0.26651	00:43:32.21	0.42086
	2	00:32:26.40	0.17085	00:22:10.42	0.21434
	4	00:15:02.78	0.07924	00:11:19.29	0.10944
	8	00:08:51.95	0.04669	00:05:52.03	0.05672

É importante destacar que o **Custo Estimado para [ECS no AWS Fargate]**, utilizando a configuração (g, t) = (100k, 8), cujos resultados estão destacados em negrito na Tabela 1, corresponde à cobrança mínima de um minuto de **[ECS no AWS Fargate]**. O mesmo ocorre para o **Custo Estimado para [ECS no Amazon EC2]**, utilizando as configurações de (100k, 4) e (100k, 8), cujos custos foram estimados com o valor de tempo de 60 segundos mesmo que a tarefa tenha demandado menos tempo.

De forma geral, foi observado que as tarefas executadas em **[ECS no Amazon EC2]** terminaram em um tempo de execução menor, em todos os casos, porém com um custo estimado maior, utilizando o mesmo "tamanho de tarefa" de 8 vCPUs e 16 GB. Para entender se o custo adicional justifica a melhora de tempo observada no EC2 em relação ao Fargate, foram coletados dados de diminuição percentual do tempo e aumento percentual do custo estimado, bem como um Fator-Benefício determinado pela razão

diminuição_percentual_tempo / aumento_percentual_custo.

Essas métricas estão apresentadas na Tabela 2. Basicamente, a interpretação do Fator Custo-Benefício indica o quão eficiente em termos de tempo por cada unidade adicional de custo o EC2 é em relação ao Fargate.

Tabela 2: Diminuição Percentual de Tempo, Aumento Percentual de Custo e Fator Custo-Benefício do EC2 em Relação ao Fargate.
Dados do EC2 capturados a partir da instância c7g.4xlarge.

Configuração		Métricas do EC2		
Tamanho	Threads	Diminuição Tempo (%)	Aumento Custo (%)	Fator Custo-Benefício
100k	1	38.05	13.67	2.78
	2	39.92	10.29	3.88
	4	45.13	45.41	0.99
	8	58.12	83.49	0.70
200k	1	34.10	20.96	1.63
	2	20.94	45.10	0.46
	4	35.73	17.99	1.99
	8	46.59	**-2.04**	**-22.83**
300k	1	15.03	55.96	0.27
	2	4.70	74.89	0.06
	4	31.87	25.05	1.27
	8	38.99	11.99	3.25
400k	1	23.89	39.70	0.60
	2	25.67	36.43	0.70
	4	28.09	31.98	0.88
	8	37.66	14.42	2.61
500k	1	13.96	57.92	0.24
	2	31.65	25.46	1.24
	4	24.76	38.11	0.65
	8	33.82	21.48	1.57

Nota-se que nos dados destacados em negrito na Tabela 2 que, para a configuração (200k, 8), o custo do EC2 foi menor que o custo do Fargate, causando um Fator Custo-Benefício negativo.

Outro produto do experimento explicitado na Seção 3.3 foi a coleta dos picos de memória que culminou na geração da Tabela 3. A célula da configuração (*g, t*) no serviço s {Fargate, EC2} foi preenchida desta forma: coletou-se os três arquivos de dados para a configuração (*g, t*) no serviço *s*, selecionando o pico de memória de cada arquivo de consumo de memória total. Com o pico de cada arquivo, calculou-se o pico de memória médio para aquela configuração.

Tabela 3: Comparação de Pico de Memória entre Fargate e EC2.
Dados do EC2 capturados a partir da instância c7g.4xlarge.

Configuração		Pico de Memória (MiB)	
Tamanho	Threads	Fargate	EC2
100k	1	100.68	103.41
	2	105.88	105.83
	4	108.03	108.71
	8	101.22	106.92
200k	1	333.49	333.37
	2	332.17	333.70
	4	323.17	312.00
	8	337.09	342.16
300k	1	707.16	706.82
	2	711.68	708.19
	4	713.49	712.70
	8	712.57	705.64
400k	1	1236.24	1235.68
	2	1237.94	1240.84
	4	1244.57	1240.73
	8	1231.67	1247.26
500k	1	1909.38	1909.23
	2	1915.43	1915.15
	4	1921.46	1912.49
	8	1922.13	1902.76

Não apenas os picos no consumo de memória, mas também o comportamento de consumo ao longo do tempo também foi analisado. Os gráficos *consumo de memória x tempo* mantiveram um desempenho semelhante ao padrão observado no relatório parcial (quando foram gerados a partir de diferentes conjuntos de sequências de entrada). Eles possuem tantos "picos" quanto alinhamentos executados pela tarefa em questão. Esses gráficos demonstraram comportamentos semelhantes entre **[ECS no AWS Fargate]** e **[ECS no Amazon EC2]**, e, visto que foram gerados 120 gráficos deste tipo gerados pela combinação 20 configurações x 3 rodadas de execuções x 2 serviços, são exibidos abaixo resultados para configurações aleatórias para comprovar sua semelhança.

- Consumo de memória para **configuração (*g*, *t*) = (200k, 2)**

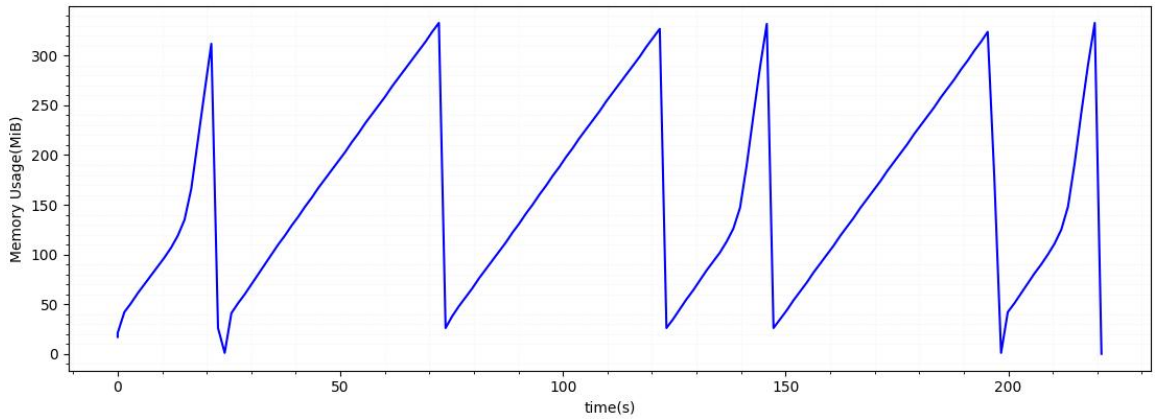

Figura 5: Consumo de memória da rodada 1 para **[ECS no AWS Fargate]**

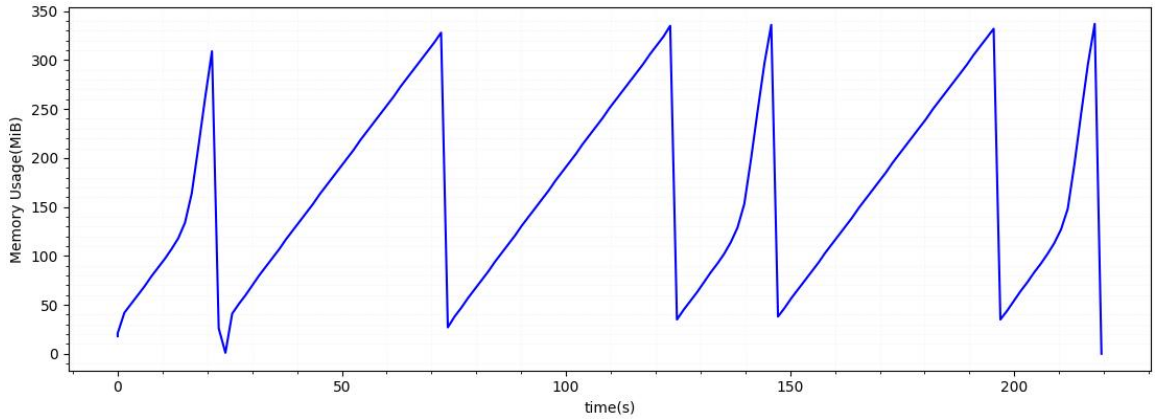

Figura 6: Consumo de memória da rodada 1 para **[ECS no Amazon EC2]**

- Consumo de memória para **configuração (grupo, *threads*) = (300k, 1)**

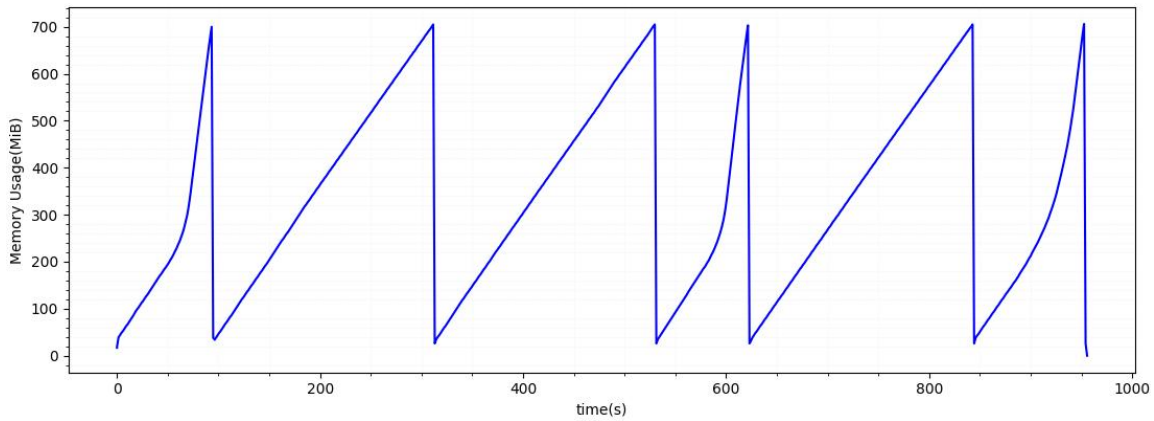

Figura 7: Consumo de memória da rodada 2 para **[ECS no AWS Fargate]**

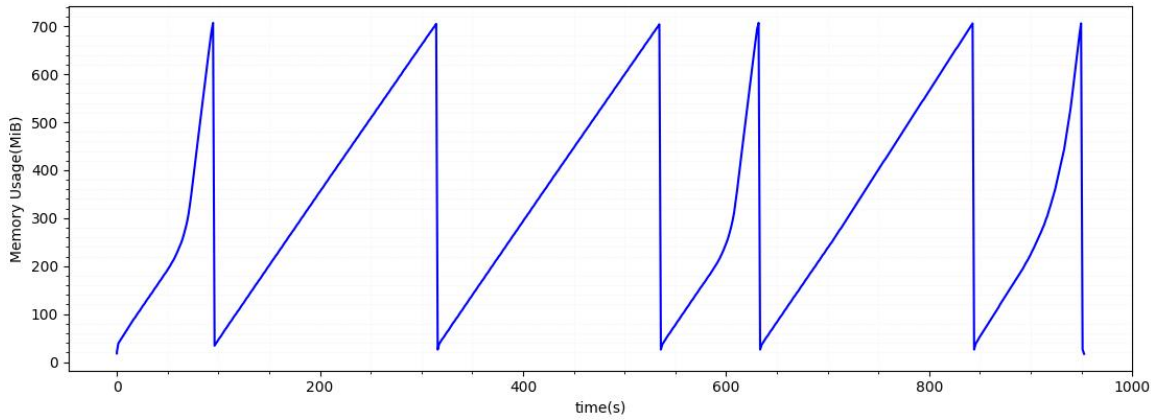

Figura 8: Consumo de memória da rodada 2 para **[ECS no Amazon EC2]**

Além da análise de pico de memória para cada configuração, o *speed-up* e a eficiência no paralelismo também foram medidos da seguinte forma: *speed-up(n) = T(1)/T(n)* , onde *n* é o número de *threads* e *T(n)* é o tempo demandado pela tarefa utilizando *n threads*, enquanto que *eficiência(n) = speed-up(n)/n.* Essas métricas foram calculadas usando como referência, para cada tamanho, o tempo de execução demandado utilizando um *thread*. Por exemplo, para as configurações (200k, 2), (200k, 4) e (200k, 8), o tempo que baseou o cálculo do *speed-up* foi o da configuração (200k, 1) no serviço em questão, enquanto que para as configurações (400k, 2), (400k, 4) e (400k, 8) o tempo base foi o de (400k, 1) no serviço em questão.

Tabela 4: Comparação de Speed-up e Eficiência entre Fargate e EC2. Dados do EC2 capturados a partir da instância c7g.4xlarge.

Configuração		Fargate		EC2	
Tamanho	Threads	Speed-up	Eficiência	Speed-up	Eficiência
100k	2	1.88	0.94	1.93	0.97
	4	2.69	0.67	3.03	0.76
	8	3.61	0.45	5.34	0.67
200k	2	**2.28**	**1.14**	1.90	0.95
	4	3.48	0.87	3.57	0.89
	8	4.85	0.61	5.99	0.75
300k	2	**2.18**	**1.09**	1.94	0.97
	4	3.04	0.76	3.79	0.95
	8	4.97	0.62	6.92	0.87
400k	2	1.92	0.96	1.97	0.99
	4	3.64	0.91	3.85	0.96
	8	5.97	0.75	7.28	0.91
500k	2	1.56	0.78	1.96	0.98
	4	3.36	0.84	3.85	0.96
	8	5.71	0.71	7.42	0.93

5. PRODUÇÃO TÉCNICO-CIENTÍFICA

- [Resumo Expandido Publicado] Gabriel C. Mills, Sara M. Cavalcante, Cristina Boeres, Vinod E. F. Rebello, "Estimativa de Tempo de Alinhamentos Biológicos na Nuvem", Escola Regional de Alto Desempenho do Rio de Janeiro (ERAD-RJ 2023) 21-23 junho 2023, Rio de Janeiro, RJ, Brasil, SBC.

- [Premiação] Gabriel C. Mills, Sara M. Cavalcante, Cristina Boeres, Vinod E. F. Rebello, "Estimativa de Tempo de Alinhamentos Biológicos na Nuvem". Prêmio de melhor artigo fórum de Iniciação Científica na Escola Regional de Alto Desempenho do Rio de Janeiro (ERAD-RJ 2023) 21-23 junho 2023, Rio de Janeiro, RJ, Brasil, SBC.

Além disso, durante este projeto foram produzidos artefatos de código e documentações que podem ser úteis para a reprodução dos experimentos descritos, compartilhados no repositório abaixo:

- Estudos em Container na Nuvem AWS, disponível publicamente em: https://github.com/saramcav/Estudos-em-container-na-nuvem-AWS.

6. CONCLUSÕES

A análise inicial do estudo de caso MASA-OpenMP revelou que o tempo de execução depende não apenas do tamanho das sequências de entrada, mas também da similaridade entre elas. Isso é evidenciado pela inclinação das retas nas Figuras 3 e 4, que indica que pares de sequências semelhantes têm tempos de execução menores e taxas de *pruning* maiores.

No contexto do estudo das ferramentas em nuvem para *container*, foi observado por meio da Tabela 1 que o tempo de execução usando **[ECS no Amazon EC2]** é, em média, 31.43% menor do que com **[ECS no AWS Fargate]**, mas com um custo médio 33.41% maior. Ainda nesse contexto, a Tabela 2 evidencia que o Fator Custo-Benefício, *F,* é equilibrado, com 9 de 20 configurações justificando o custo extra do **[ECS no Amazon EC2]** em relação ao **[ECS no AWS Fargate]**, nos casos em que $F > 1$. No entanto, algo a ser considerado é que a diferença de custo é influenciada pela instância utilizada no **[ECS no Amazon EC2]**, c7g.4xlarge, que, por possuir mais recursos do que os que a tarefa necessita, pode se mostrar mais custo-efetiva com a utilização de mais *containers* sobre esse mesmo *host* pagando-se ainda pelo mesmo custo estimado. Esse aspecto não foi explorado durante o projeto, o que limita uma conclusão mais geral.

O consumo de memória entre **[ECS no AWS Fargate]** e **[ECS no Amazon EC2]** se mostrou semelhante, tanto nos valores máximos demandados para cada configuração, apresentados na Tabela 3, quanto no padrão visual do comportamento de consumo ao longo do tempo, notado ao comparar as Figuras 5 com 6 e as Figuras 7 com 8.

No âmbito do paralelismo, dados de *speed-up* e eficiência apresentados na Tabela 4 expõem que **[ECS no Amazon EC2]** geralmente (em 18 das 20 configurações) oferece melhor desempenho de tempo que **[ECS no AWS Fargate]**, aproveitando melhor o aumento de *threads* para diminuir o tempo de execução. A ocorrência de dois casos em que a eficiência é maior do que 1 no **[ECS no AWS Fargate],** para as configurações (200k, 2) e (300k, 2), é curiosa. Isso porque, como não se pode conhecer o *host* sobre o qual as execuções das configurações (200k, 1) e (300k, 1) aconteceram, apenas a arquitetura dos processadores, levanta-se a dúvida sobre se essa eficiência pode ter sido gerada por uma troca de infraestrutura sobre a qual as tarefas executaram.

Em termos de facilidade de uso, **[ECS no AWS Fargate]** é mais simples para iniciar tarefas, mas pode gerar dúvidas que são mais facilmente resolvidas com **[ECS no Amazon EC2]**. Por exemplo, para entender o motivo pelo qual uma tarefa não entra em estado "*em execução*", pode-se acessar remotamente as instâncias do Amazon EC2 e averiguar o problema, o que não é possível ao utilizar o AWS Fargate. Apesar da simplicidade, como o fato de não precisar selecionar o tipo de instância em **[ECS no Amazon EC2]**, a configuração de tarefas em **[ECS no AWS Fargate]** pode ser complexa para usuários não especializados, dado que é necessário haver uma previsão de recursos, em vCPU e GiB, que a computação a ser executada demandará.

Em resumo, o estudo cumpriu seus objetivos ao avaliar a execução da aplicação MASA-OpenMP em ambientes de nuvem, comparando o desempenho, custo e facilidade de uso entre **[ECS no AWS Fargate]** e **[ECS no Amazon EC2]**. As comparações mostraram nuances importantes e contribuirão para futuras decisões sobre essas soluções de infraestrutura na nuvem.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[Amazon Web Services Inc, 2024a] Preço do Amazon Elastic Container Registry. Disponível em: https://aws.amazon.com/pt/ecr/pricing/. Acesso em: 21 ago. 2024.

[Amazon Web Services Inc, 2024b] Preço sob demanda do Amazon EC2. Disponível em: https://aws.amazon.com/pt/ec2/pricing/on-demand/. Acesso em: 28 ago. 2024.

[Amazon Web Services Inc, 2024c] Preço do Amazon EC2. Disponível em: https://aws.amazon.com/pt/ec2/pricing/. Acesso em: 21 ago. 2024.

[Amazon Web Services Inc, 2024d] Definição de preço do AWS Fargate. Disponível em: https://aws.amazon.com/pt/fargate/pricing/. Acesso em: 28 ago. 2024.

[Amazon Web Services Inc, 2024e] Definições de tarefa do Amazon ECS. Disponível em: https://docs.aws.amazon.com/pt_br/AmazonECS/latest/developerguide/task_definitions.html. Acesso em: 25 ago. 2024.

[Amazon Web Services Inc, 2024f] Como usar o Container Insights. Disponível em: https://docs.aws.amazon.com/pt_br/AmazonCloudWatch/latest/monitoring/ContainerInsights.html. Acesso em: 24 ago. 2024.

[Amazon Web Services Inc, 2024g] Definição de preço do Amazon ECS. Disponível em: https://aws.amazon.com/pt/ecs/pricing/. Acesso em: 24 ago. 2024.

[Amazon Web Services Inc, 2024h] Definição de preço do Amazon S3. Disponível em: https://aws.amazon.com/pt/s3/pricing/. Acesso em: 24 ago. 2024.

[Amazon Web Services Inc, 2024i] Definição de preço do AWS Lambda. Disponível em: https://aws.amazon.com/pt/lambda/pricing/. Acesso em: 24 ago. 2024.

[Amazon Web Services Inc, 2024j] Describe_tasks. Disponível em: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ecs/client/describe_tasks.html. Acesso em: 01 set. 2024.

[Amazon Web Services Inc, 2024k] Preço do Amazon EBS. Disponível em: https://aws.amazon.com/pt/ebs/pricing/. Acesso em: 25 ago. 2024.

[De O. Sandes et al., 2016] SANDES, E. F. O.; MIRANDA, G.; MARTORELL, X.; AYGUADÉ, E.; TEODORO, G.; MELO, A. C. M. A. MASA: A multiplatform architecture for sequence aligners with block pruning. ACM Transactions on Parallel Computing, v. 2, n. 4, 2016. Código disponível em: https://github.com/edanssandes/MASA-OpenMP.

[Docker Inc., 2024] Docker: Container Engine . Disponível em: https://www.docker.com/. Acesso em: 25 ago. 2024.

[Garfinkel, 2011] GARFINKEL, S. Before Facebook and Google — even before Internet — scientists at MIT had a radical vision they called the computer utility. MIT Technology Review, Cambridge, 03 out. 2011. Disponível em: https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/. Acesso em: 25 ago. 2024.

[Google Cloud, 2024] O que é o Kubernetes? Disponível em: https://cloud.google.com/learn/what-is-kubernetes?hl=pt-br. Acesso em: 25 ago. 2024.

[Myers and Miller, 1988] MYERS, E. W.; MILLER, W. Optimal alignments in linear space. Bioinformatics, oxford, v. 4, n. 1, p. 11–17, 1988.

[Sharma et. al., 2016] SHARMA, P.; CHAUFOURNIER, L.; SHENOY, P.; TAY, Y. C. Containers and Virtual Machines at Scale: A Comparative Study. *In:* PROCEEDINGS OF THE 17TH INTERNATIONAL MIDDLEWARE CONFERENCE (MIDDLEWARE '16), 17., 2016, New York. Anais… New York: ACM, 2016. p. 1–13. DOI: https://doi.org/10.1145/2988336.2988337.

[Sodré, D., 2024] SODRÉ, D. Extraindo o máximo de instâncias da nuvem ao atrasar tarefas para melhorar a utilização de recursos. Projeto Final de Curso, Ciência da Computação, Instituto de Computação, Universidade Federal Fluminense, 2024.

[Sodré et al., 2022] SODRÉ, D.; BOERES, C.; REBELLO, V. (2022). Making the most of what you pay for by delaying tasks to improve overall cloud instance performance. *In:* XXIII SIMPÓSIO EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO - SBC, 23., 2022, Florianópolis. Anais estendidos… Florianópolis: SBC, 2022. p. 9–16.

8. AUTO-AVALIAÇÃO DO ALUNO

Por meio da ajuda e das revisões da orientadora, tenho aprendido sobre a minuciosidade necessária para expressar a informação que o comprometimento com a ciência demanda, e, assim, venho adaptando minha escrita de modo a evitar palavras imprecisas e vagas. A clareza e a reprodutibilidade assumiram, durante as escritas relacionadas a essa iniciação científica, papéis mais importantes em minha avaliação, e as tenho buscado durante este projeto ao descrever tudo que tenho feito. Por exemplo, comentários em *scripts* passaram a ser vistos como indispensáveis, pois favorecem a reprodutibilidade e a colaboração de diferentes pessoas. Outros exemplos são vistos na documentação das configurações e sequências genéticas utilizadas, e na produção de documentos ‘passo a passo’ para futuros alunos. Tenho me atentado à necessidade de especificar as fontes dos fatos que escrevo, e não

pressupô-los como senso comum. Além disso, também tenho aprendido sobre a importância de quantificar os dados, isolar as variáveis e decidir as métricas relevantes para formar um experimento científico sério.

No quesito acadêmico, obtive um conhecimento que agrega positivamente. Pude aprender sobre o surgimento da Computação em Nuvem, da virtualização, dos contêineres e de todos os conceitos relacionados a este projeto e descobrir que remetem a definições mais antigas do que eu esperava. Isso me fez entender a importância de estudar a história da computação e de saber reaproveitar conhecimentos já vistos para implementar soluções novas. Tive a oportunidade de aprender a escrever *shell scripts*, algo que pode parecer simples, mas que, para alguém que nem mesmo conhecia comandos UNIX, é representativo. Também tenho aprendido a utilizar o Docker e conhecido ferramentas muito utilizadas no mercado, de interesse de empresas importantes, como o Amazon ECS, o AWS Fargate, o AWS Lambda e todas as ferramentas mencionadas neste relatório. Aprendi a utilizar bibliotecas com ferramentas estatísticas disponíveis para as linguagens Python e R que me ajudarão a lidar com dados durante a jornada acadêmica. Pude praticar a interpretação dos resultados experimentais, o que também será fundamental para meus futuros trabalhos. Enfim, muitas ferramentas foram exploradas e incrementadas a meus recursos para pensar soluções computacionais.

No âmbito pessoal, minha paciência quanto ao processo de aprendizado aumentou, pois aprendi sobre a necessidade de construir conhecimento aos poucos. Por ser meu primeiro contato com as ferramentas de *container* em nuvem, enfrentei dificuldades quanto às configurações necessárias para executar os experimentos, e isso me fez perceber que “dar passos atrás” para entender as ferramentas a fundo seria importante. Isso também me fez perceber a importância de uma boa documentação. Além disso, meu senso crítico tem sido aprimorado durante a avaliação dos serviços em nuvem, devido à percepção da disparidade entre a facilidade vendida para utilizá-los e como isso é na prática.

Ainda no âmbito pessoal, ao final do projeto voltei minhas atenções para ensinar passos iniciais na Computação em Nuvem para dois alunos de graduação interessados em começar seus estudos no tema que futuramente provavelmente farão parte de nosso laboratório de estudos . Com isso, tenho refletido sobre a melhor forma de expor esse conhecimento adquirido ao passo que esse tipo de responsabilidade tem fixado melhor os conceitos para mim. Essa interação é interessante, pois quando comecei também tive alunos do laboratório que me ajudaram.

Acredito que nos três meses finais do projeto eu não tenha desempenhado da melhor forma possível em termos experimentais, pois foi um pouco desanimadora a quantidade de configurações que a interação com essas ferramentas demanda. Isso impediu que os serviços fossem analisados de uma forma mais profunda, utilizando mais *containers* sobre um mesmo *host* ou mesmo avaliando o Amazon EKS, elencados como trabalhos futuros no

relatório parcial. Além disso, também teria sido interessante explorar a configuração do *cluster* com mais instâncias. Contudo, de forma geral, creio que fiz um bom trabalho ao enfrentar a dificuldade de coletar informações e de ter o primeiro contato com muitas ferramentas.