Web Scraping na Nuvem AWS: Uma Abordagem com Máquinas Virtuais *Burstable*[*]

Matheus Marotti Pereira[1], **Thiago do Prado Silva**[1], **Lúcia Maria de A. Drummond**[1]

[1] Instituto de Computação
Universidade Federal Fluminense (UFF) – Niterói, RJ – Brasil

{matheusmarotti, pradothiago}@id.uff.br, lucia@ic.uff.br

Resumo. *A presente pesquisa tem por objetivo propor um sistema de Web Scraping, baseado na nuvem computacional da AWS, utilizando as máquinas EC2 burstable. O framework define um cluster misto, com instâncias burstable fixas e temporárias, que pode variar o número de instâncias, adicionando ou removendo VMs, para garantir o SLA das mensagens e minimização dos custos. O framework proposto foi avaliado na nuvem AWS, comparado a uma abordagem apenas com instâncias on-demand não expansíveis e também a outra solução baseada em Function as a Service (FaaS). Os resultados mostraram que todos os testes atendem o SLA definido, alcançando uma redução de 96% de custo financeiro, em seu melhor caso quando comparado à abordagem FaaS, e redução de 95,59%, em seu melhor caso quando comparado à abordagem on-demand. Além disso, houve redução de custos de no mínimo 93,26% em todos os demais casos de teste, mostrando que máquinas burstable podem ser um ótimo recurso para esse problema.*

1. Introdução

Web Scraping, ou raspagem da Web, é uma técnica de extração de dados de páginas da internet, que pode ser feita manualmente ou de forma automatizada utilizando ferramentas desenvolvidas para essa tarefa. Esses programas abrem navegadores, transitam entre *sites*, preenchem formulários e até manipulam cabeçalhos para simular um acesso feito de forma manual. Com o crescimento da *World Wide Web* (WWW), em conjunto com a expansão de aplicações *Big Data*, a técnica de *Web Scraping* se tornou muito útil para diversas atividades, como criação de *data sets*, automatização de processos e monitoramento de informações [Zhao 2017].

Através dessa técnica, *scripts* são programados para acessar *websites* e buscar informações para as mais diversas aplicações. Orquestrar uma série de robôs para extrair informações relevantes de *sites* por si só não é trivial, mas quando acrescido da necessidade de fazê-lo em alta velocidade e escala, essa tarefa ganha ainda mais complicações relacionadas a provisionamento de recursos e controle de acessos.

Apesar de anteriormente ser vista apenas como uma opção em potencial para executar essas aplicações, a nuvem agora é uma alternativa segura, confiável e acessível para realizar tarefas de *Web Scraping* [Chaulagain et al. 2017]. Esse paradigma computacional

[*]Essa pesquisa está sendo financiada pelo Projeto Universal/CNPq n^o 404087/2021-3 e pelo Projeto CNE/FAPERJ n^o E-26/201.012/2022(271103)

oferece benefícios valiosos, como o provisionamento rápido de recursos e uma redução significativa nos custos operacionais relacionados a energia, licença de *software* e obsolescência de *hardware*. Devido a essas vantagens, aplicações de *Web Scraping*, antes executadas em infraestruturas dedicadas, passaram a ser executadas mais frequentemente em ambientes computacionais oferecidos por provedores de nuvem.

Mais recentemente, um novo tipo de máquina virtual oferecido por grande parte dos provedores comerciais de nuvens surgiu: as instâncias *burstable*. Segundo a definição utilizada pela AWS em sua documentação[1], máquinas virtuais (VMs) *burstable* são instâncias que fornecem um desempenho base de CPU e podem expandir acima dessa taxa base por quanto tempo for necessário. Dessa forma, esse tipo de máquina se torna útil para sistemas que, no geral, possuem cargas de trabalho mais leves com possibilidade de picos, variando de um estado ocioso para alta demanda. Um exemplo de sistema com essa característica é o estudado neste trabalho, um servidor de *Web Scraping* por demanda. Hoje é possível utilizar instâncias com propriedade expansível nos principais provedores de nuvem, tal como na AWS, instâncias da classe t, ou na Microsoft Azure, instâncias da série B[2].

Como nas instâncias *burstable*, tipicamente, o usuário utiliza apenas uma fração dos núcleos virtuais (vCPUs) da instância na maior parte do tempo, o custo dessas instâncias é mais baixo se comparado com outras VMs de características parecidas no mercado *on-demand*. Também, como a quantidade de rajadas realizadas pode variar, o custo dessas instâncias é variável e cada provedor tem sua própria maneira de fazer esse cálculo. Na AWS o custo de uma máquina com características expansíveis é calculado a partir de créditos, gerados e gastos, durante a execução[1]. Cada instância, pertencente a uma classe *burstable*, gera uma quantidade de créditos de CPU ao longo de sua execução, onde instâncias mais robustas podem gerar mais créditos. Cada instância também tem um tamanho máximo de armazenamento de créditos, com maior capacidade de acordo com o tamanho da instância. Cada unidade de crédito significa 100% de utilização de uma vCPU por um minuto. Existe a possibilidade de calcular o custo proporcional para percentuais menores de utilização, onde por exemplo, 50% de utilização de uma vCPU por dois minutos também custaria um crédito. Cada instância tem um limiar de utilização total onde a quantidade de créditos gerados é a mesma de créditos consumidos. O nome desse limiar é linha de base (*baseline*). Quando uma instância está com utilização abaixo de sua linha de base, ela acumula créditos (estado de *baseline*), já quando ela está acima do limiar, ela decresce o total de créditos (estado de *burst*).

Apesar do lançamento do Elastic Compute Cloud (EC2) em agosto de 2006, a AWS só anunciou sua primeira classe de instâncias expansíveis em julho de 2014, quase 8 anos depois [3]. Até hoje, vários trabahos da literatura continuam explorando possibilidades para ter mais vantagem com esse tipo de instância [Wang et al. 2017, Baarzi et al. 2019, Jiang et al. 2019, Dantas et al. 2021, Teylo et al. 2021].

---

[1]Key concepts and definitions for burstable performance instances - https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html

[2]B-series burstable virtual machine sizes - https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-b-series-burstable

[3]EC2 Instance History - https://aws.amazon.com/pt/blogs/aws/ec2-instance-history/

Com a evolução das plataformas de nuvem, houve um aumento de soluções de raspagem nesses ambientes [Chaulagain et al. 2017, Woodall et al. 2021]. Como aplicações de *Web Scraping* sob demanda podem atingir momentos de pico de requisições, eventualmente retornando para um estado com número de requisições mais baixo, podemos entender que este tipo de sistema possui carga de trabalho variável. Desta forma, nos parece vantajoso o uso de instâncias *burstable* para criação de um sistema de *Web Scraping* que consiga tirar proveito das capacidades expansíveis dessas instâncias.

Do nosso conhecimento são poucos os trabalhos que tratam *Web Scraping* em nuvens computacionais. No trabalho [Woodall et al. 2021], os autores Woodall *et al* propuseram um modelo de raspagem na Azure, nuvem da Microsoft. Os autores criaram uma arquitetura para obter avaliações no *e-commerce* da Amazon. Foram 17962 avaliações processadas pelo serviço, em uma média de 266 avaliações por hora. É importante ressaltar a recomendação dos autores: uma implementação futura incluindo um sistema próprio de raspagem para diminuir os custos. Em 2017, Chaulagain *et al* fizeram um sistema de *Web Scraping* para aplicações *Big Data* na nuvem [Chaulagain et al. 2017]. O principal argumento para a relevância da solução foi a utilização da raspagem como ferramenta para extração de dados não estruturados da *Web* e, em adição, as características de escalabilidade e flexibilidade garantidas pela nuvem. Em testes, o modelo proposto pelos autores executou até cinco vezes mais rápido do que uma abordagem fora da nuvem.

Neste artigo, apresentamos um *framework* para *Web Scraping*, aproveitando as vantagens das instâncias *burstable*, na nuvem computacional da AWS. Nossa abordagem faz uso de um *cluster* misto de instâncias *burstable* fixas e temporárias para realização das raspagens e consegue garantir uma redução de mais de 90% dos custos quando comparada a abordagens com VMs regulares e *Function as a Service* (FaaS).

Apesar do uso de instâncias com características expansíveis no trabalho de Chaulagain *et al* [Chaulagain et al. 2017], o mesmo não explora essas vantagens, não diferenciando tais instâncias de outras sem esse atributo. Assim, este artigo, do nosso conhecimento, é o primeiro a explorar as características da instâncias *burstable* no tópico de *Web Scraping*, através da gerência de tempos de atendimento dos pedidos de serviços, tamanho das filas de pedidos, e do SLA, tempo máximo de atendimento informado pelo usuário, para criação e eliminação de novas instâncias.

O presente trabalho traz as seguintes contribuições: (i) um *framework* para *Web Scraping* com máquina virtuais *burstable* na nuvem AWS; (ii) a comparação entre o desempenho da arquitetura proposta que utiliza instâncias *burstable* e duas outras abordagens: 1) uma arquitetura que utiliza apenas instâncias *on-demand* sem a característica expansível, e 2) uma arquitetura que utiliza *Function as a Service*, serviço *serverless*, que a maior parte das nuvens comerciais oferece atualmente. As arquiteturas escolhidas para comparação, foram selecionadas a partir da constatação de que o uso das instâncias *on-demand* tem sido a escolha mais natural de infraestrutura para a maior parte das aplicações que usam nuvens, e o *FaaS* por ser um tipo de serviço disponibilizado mais recentemente, que vem ganhando cada vez mais adeptos em uma variedade de aplicações na nuvem e que é apontado pela AWS como uma boa escolha para serviços de alta escalabilidade[4].

[4]What is AWS Lambda? - https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

O restante do artigo está organizado da seguinte forma: a Seção 2 descreve o *framework* proposto e a Seção 3 apresenta o ambiente de testes e seus resultados; por último, a Seção 4 traz as conclusões e considerações para trabalhos futuros.

2. *Framework* proposto

A arquitetura proposta neste trabalho foi projetada para a nuvem computacional da Amazon, a AWS. O projeto utiliza serviço de filas, serviço de monitoramento, serviço de armazenamento e serviço de infraestrutura, SQS, CloudWatch, S3 e EC2, respectivamente. Além disso, foram utilizados o EC2 Image Builder para criação da imagem das instâncias EC2 e o SSM Parameter Store para armazenar parâmetros importantes para a execução da aplicação. Uma visão geral da arquitetura mostrando os recursos empregados e suas interações pode ser visto na Figura 1.

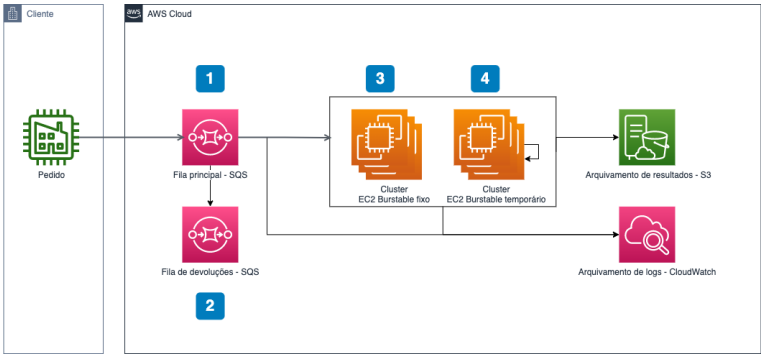

Figura 1. Arquitetura do *framework* para execução de *Web Scraping* na AWS

Os pedidos de raspagem recebidos são alocados em uma fila principal, número 1 da Figura 1, para serem consumidos em seguida pelo *cluster* de instâncias EC2. Associada a essa fila, existe uma fila de devoluções, configurada como *dead-letter queue* da fila principal que recebe mensagens que não conseguiram ser processadas após três tentativas. A fila de devoluções, número 2 no diagrama, é importante para evitar perda de mensagens e facilitar a depuração. Ambas as filas descritas seguem o padrão *First-In-First-Out* (FIFO), importante para a garantia de ordem no atendimento dos pedidos recebidos.

Nos números 3 e 4 da Figura 1, estão os *clusters* de instâncias EC2 *burstable* fixo e temporário, respectivamente. As máquinas são criadas utilizando uma imagem confeccionada através do EC2 Image Builder, contendo todo software necessário para execução da aplicação. Todas as instâncias, executam um serviço constante de consumo da fila SQS para buscar um *batch* de mensagens e, em seguida, executar o processo de raspagem também em *batch*, utilizando a biblioteca Puppeteer, mantida pelo time do Chrome DevTools[5]. A biblioteca Puppeteer simula um acesso humano nos websites requeridos, atuando na aquisição dos recursos Web e extração da informação desejada. Esse tipo de raspagem traz a vantagem de ter mais ferramentas associadas para melhorar a extração do conteúdo demandado. Ao fim da execução, os resultados do processo de *Web Scraping* são salvos no formato JSON no serviço de armazenamento S3, e as métricas sobre os

[5]Getting Started with Headless Chrome - https://developer.chrome.com/docs/puppeteer/overview/

tempos de processamento de mensagens são exportadas para o CloudWatch[6], encerrando aquela execução e dando início a um novo consumo da fila principal SQS.

O *framework* proposto gerencia de forma diferente o *cluster burstable* fixo e o temporário. A quantidade de instâncias no *cluster burstable* fixo (número 3 no diagrama) é definida na criação do ambiente, mantendo-se constante durante toda a execução. Já a quantidade no *cluster* temporário (número 4 no diagrama) é variável durante a execução e depende do número de mensagens na fila. O número de instâncias no *cluster burstable* fixo não é variável para que créditos possam ser gerados em momentos de baixa quantidade de mensagens na fila, criando uma reserva e possibilitando uma rajada em momentos de alta demanda.

A seguir, apresentaremos o modelo utilizado para adicionar ou eliminar VMs temporárias. Note que a partir de agora usaremos o termo *cluster* para nos referirmos ao *cluster* contendo os dois *clusters* definidos anteriormente, tanto o fixo como o temporário. Entretanto, quando mencionarmos a necessidade de acrescentar ou eliminar VMs, estaremos falando apenas do *cluster* temporário.

O *framework* proposto calcula a quantidade ideal de instâncias no *cluster* a cada um quarto do SLA (*Service Level Agreement*), que é o tempo máximo acordado com o usuário para que cada requisição (mensagem) seja processada, e será chamado de Ciclo de Alocação. Assim, a cada Ciclo de Alocação, o número ideal de máquinas virtuais para o *cluster* inteiro (considerando tanto o fixo e o temporário) (N) é calculado através de uma equação que relaciona: o número de mensagens na fila (M), o tempo médio de serviço do *cluster* (S), o tempo máximo de processamento de mensagens ($Tsla$) acordado com o usuário, e a quantidade de mensagens consumidas paralelamente em cada máquina (P). Essa equação é encontrada a partir de três outras equações que definem a taxa de serviço do *cluster* (μ), o número de consumidores (Q) e o número de mensagens na fila (M). Se o número obtido for maior do que a quantidade de instâncias ativas, novas instâncias temporárias são iniciadas. Se o resultado da equação for um número menor do que a quantidade de instâncias ativas e, se o tempo de espera da mensagem mais antiga da fila for menor que o tempo do SLA, máquinas virtuais são encerradas. Por exemplo, tendo em vista o SLA de 300 segundos, se o *cluster* tem 3 instâncias ativas e a equação indicou que apenas 2 instâncias são necessárias, mas a mensagem mais antiga está esperando para ser executada há 320 segundos, nenhuma instância será encerrada. A Tabela 1 apresenta a notação utilizada.

Assim, podemos definir:

$$\mu = Q \div S \tag{1}$$

Ou seja, se, por exemplo, no Ciclo de Alocação, existirem 10 consumidores (Q) e o *cluster* for capaz de atender cada mensagem em 5 segundos (S), a taxa de serviço será igual a 2 (μ), o que representará 2 mensagens sendo atendidas pelo *cluster* por segundo.

O número de consumidores é definido pela Equação 2. Assim, por exemplo, se existirem 2 instâncias ativas (N), cada uma responsável por um conjunto de 5 mensagens, *batch* (P), o número de consumidores será 10.

$$Q = N * P \tag{2}$$

[6]Amazon CloudWatch - https://aws.amazon.com/cloudwatch

Variável	Significado
N	Número de máquinas virtuais ativas no Ciclo de Alocação
μ	Taxa de serviço do *cluster*, número de mensagens atendidas a cada Ciclo de Alocação
Q	Número de consumidores, número total de mensagens a serem atendidas a cada Ciclo de Alocação
M	Número de mensagens na fila no Ciclo de Alocação
S	Tempo médio de serviço do *cluster* no Ciclo de Alocação
$Tsla$	Tempo máximo acordado para processamento de cada mensagem
P	Tamanho do *batch* de mensagens definido para consumo por cada máquina virtual

Tabela 1. Notação utilizada

A próxima equação define o número de mensagens na fila, que é calculado pelo produto do tempo limite de serviço de cada mensagem, definido pelo usuário, e a taxa de serviço. Assim, se o usuário definir que o tempo serviço desejado é de 1 segundo e a taxa de serviço for de 2 mensagens por segundo, a fila terá duas mensagens.

$$M = Tsla * \mu \tag{3}$$

Consequentemente, realizando algumas substituições nas equações (1), (2) e (3), obtemos:

$$N = (M * S) \div (Tsla * P)$$

Logo, o número de VMs necessárias (N) para atender cada mensagem respeitando o acordo de serviço dado pelo tempo $Tsla$ é igual ao número de mensagens na fila (M) vezes o tempo médio de serviço do *cluster* (S), dividido pelo produto entre o tempo $Tsla$ e o tamanho do *batch* P.

Note que $Tsla$, tempo máximo acordado para o processamento de cada mensagem, e P, tamanho do *batch* de mensagens definido para consumo, são dados de entrada pelo usuário no *framework*. Salienta-se que o tamanho do *batch* escolhido deve ser o número máximo de casos concorrentes suportados pela VM escolhida. Dessa forma, as máquinas virtuais serão aproveitadas ao máximo durante sua execução independente de sua capacidade.

Tanto o cálculo quanto as operações de iniciar e encerrar instâncias são procedimentos realizados por uma função orquestradora do *framework*. A função orquestradora é uma rotina executada em uma instância do *cluster burstable* fixo. A cada $Tsla/4$ a função é executada, calculando o número de instâncias ideal para o *cluster* e adaptando-o, através da inicialização ou do encerramento das instâncias temporárias.

3. Experimentos computacionais

Para criar os recursos AWS para realização dos testes, utilizamos Serverless Framework[7] e Cloudformation[8], serviço de *Infrastructure as Code* (IaC) da AWS. Serverless Framework

[7]Serverless Framework Documentation - https://www.serverless.com/framework/docs
[8]AWS CloudFormation - https://aws.amazon.com/pt/cloudformation/

é um *framework* desenvolvido pela empresa Serverless Inc em JavaScript com objetivo de ajudar a criar recursos, principalmente do tipo *serverless*, nas plataformas de computação na nuvem mais relevantes.

Em nosso caso, o Serverless Framework foi utilizado para auxiliar na criação e implantação de uma pilha Cloudformation, unidade que agrupa recursos da nuvem AWS, contendo os recursos necessários para a criação do *framework* proposto. Além disso, para preparação das máquinas virtuais e ambiente, foram utilizados o EC2 Image Builder, para criação de uma imagem com AWS Linux 2 e dependências para executar o processo de *Web Scraping*, e o Parameter Store do AWS Systems Manager para guardar variáveis necessárias para execução correta do sistema.

Definimos apenas um *site* para ser acessado pelo sistema em nosso teste. O *site* escolhido é o Conselho Regional de Enfermagem do Rio de Janeiro (COREN-RJ)[9]. Escolhemos esse *website* como fonte, porque ele fornece informações de interesse público, no caso, a verificação de cadastro no COREN-RJ, e não disponibiliza acesso das informações via API, tornando *Web Scraping* uma boa escolha para buscar tais informações de maneira automatizada.

Para realizar as raspagens, definimos um *script* utilizando a biblioteca Puppeteer. O Puppeteer utiliza o Chrome DevTools Protocol para manipular um navegador *headless*, que não possui interface gráfica do usuário, e então realizar o acesso automatizado ao *site* desejado[5]. O *script* descreve uma série de passos a serem executados no navegador, como o acionamento de um botão para trocar de páginas e preenchimento de uma barra de pesquisa com o documento escolhido, para ao fim, extrair o HTML da página com a informação sobre o registro no COREN, encerrando a raspagem.

Para simular os pedidos de *Web Scraping*, criamos um *script* para popular a fila de entrada do sistema. O *script* recebe como entrada os seguintes parâmetros: (i) um tamanho de *batch*, (ii) um número de *batches* total, (iii) e um tempo em milissegundos que representa o tempo entre cada envio de *batch*. Variando esse parâmetros, o sistema pode ser avaliado em vários cenários de utilização, com muitas ou poucas requisições. Em nossos experimentos, são explorados três níveis de utilização do sistema: alta, média e baixa. A definição dos parâmetros utilizados em cada teste está descrita na próxima seção. Para viabilizar a reprodutividade da pesquisa o código fonte do framework está disponível no repositório https://github.com/PradoTPS/aws-scraper-cost-optimization.

3.1. Configuração dos cenários de testes

As configurações dos *scripts* de população da fila para definição dos três cenários de pedidos de raspagem estão apresentados na Tabela 2. Através de vários experimentos, observou-se que o tempo médio de resposta do *site* era de aproximadamente 10 segundos. Baseado neste valor, definimos os tempos entre *batches* para os três cenários da seguinte forma: (i) *Utilização alta*: é igual ao tempo médio de resposta do site, de forma a termos, em grande parte do tempo, pedidos a serem processados, independentemente do sistema consumidor; (ii) *Utilização média*: é 50% maior que o cenário *Utilização alta*, levando assim à uma menor taxa de chegada de pedidos; e (iii) *Utilização baixa*: é o dobro do

[9]COREN-RJ: Acesse Sua Inscrição - https://servicos.coren-rj.org.br/appcorenrj/incorpnet.dll/Controller?pagina=pub_mvcLogin.htm&conselho=corenrj

cenário *Utilização alta*. Dessa forma conseguimos analisar como o sistema se comporta em diferentes cenários de chegada de pedidos.

Já o tamanho do *batch*, quantidade de casos executados ao mesmo tempo por uma única máquina, foi definido de forma a ser o maior número de pedidos de raspagem que uma máquina consegue executar ao mesmo tempo. Em todos os cenários o número de *batches* foi escolhido para termos um tempo total de execução do *script* de população da fila igual a 15 minutos (*Tempo entre batches* x *Número de batches*).

Nome do cenário	**Tamanho do *batch***	**Tempo entre *batches***	**Número de *batches***
Utilização alta	20	10 s	90
Utilização média	20	15 s	60
Utilização baixa	20	20 s	45

Tabela 2. Cenários utilizados para testes

3.1.1. Configuração dos ambientes na nuvem

Foram escolhidas três configurações de sistema, descritas na Tabela 3, onde N *máximo* é o número máximo de máquinas virtuais ativas em um ciclo de ativação, ou seja o maior tamanho possível do *cluster* durante a execução dos testes.

Na Tabela 3 também estão definidos os tipos de instâncias utilizadas no *cluster* EC2 *burstable* fixo e temporário. Os tipos de instâncias consistem em várias combinações de CPU, memória, armazenamento e capacidade de rede[10].

Os sistemas *Burstable - 1* e *Burstable - 2* se diferenciam apenas no tamanho do *cluster burstable* fixo, possuindo uma e duas instâncias, respectivamente. Nos dois sistemas foram escolhidas instâncias do tipo *t2.micro*[11] para o *cluster* temporário. As instâncias t2 são uma das opções de instância do Amazon EC2 de menor custo, com capacidade expansível, sendo ideais para uma variedade de aplicativos de uso geral. Já para o *cluster* fixo foram utilizadas instâncias do tipo *t3.micro*[12], um tipo de instância mais otimizada para capacidade *burstable*. Para o sistema *On-demand* foram utilizadas instâncias do tipo *m1.small*, um tipo de baixo custo que não possui capacidade de intermitência, sendo assim um sistema 100% *on-demand*. A taxa por hora das instâncias *t2.micro*, *t3.micro* e *m1.small* são, respectivamente, 0,0116 USD, 0,0104 USD[13] e 0,044 USD[14].

[10]Amazon EC2 Instance Types - https://aws.amazon.com/ec2/instance-types

[11]Amazon EC2 T2 Instances - https://aws.amazon.com/ec2/instance-types/t2

[12]Amazon EC2 T3 Instances - https://aws.amazon.com/ec2/instance-types/t3

[13]Preço sob demanda do Amazon EC2 - https://aws.amazon.com/pt/ec2/pricing/on-demand/

[14]Preço sob demanda do Amazon EC2 - https://aws.amazon.com/pt/ec2/pricing/on-demand/

Nome do sistema	**Tamanho do *cluster* fixo**	N **máximo**	P	$Tsla$	**Tipo das instâncias do *cluster* fixo**	**Tipo das instâncias do *cluster* temporário**
Burstable - 1	1	10	5	5 min	t3.micro	t2.micro
Burstable - 2	2	10	5	5 min	t3.micro	t2.micro
On-demand	1	10	5	5 min	m1.small	m1.small

Tabela 3. Sistemas utilizados para testes

3.2. Resultados obtidos

Combinando os diferentes cenários e sistemas, os resultados obtidos podem ser encontrados na Tabela 4. Os custos abaixo são referentes apenas aos recursos de processamento, EC2 ou Lambda. O cálculo do custo das máquinas EC2 foi realizado através da taxa por segundo, proveniente da taxa por hora informada pela AWS. Para cada cenário o sistema que obteve o melhor resultado, em relação ao custo, está destacado em negrito.

Cenário	**Sistema**	**Tempo médio de processamento do pedido**	**Custo**
Utilização alta	**Burstable - 1**	**3,58 min**	**0.0088 USD**
	Burstable - 2	2,64 min	0.0124 USD
	On-demand	5.13 min	0.1705 USD
	Lambda	∼7 s	0.16 USD
Utilização média	**Burstable - 1**	**2,17 min**	**0.0044 USD**
	Burstable - 2	0.46 min	0.0073 USD
	On-demand	4.82 min	0.0999 USD
	Lambda	∼7 s	0.11 USD
Utilização baixa	**Burstable - 1**	**1,96 min**	**0.0050 USD**
	Burstable - 2	0.55 min	0.0069 USD
	On-demand	4.59 min	0.0742 USD
	Lambda	∼7 s	0.08 USD

Tabela 4. Resultados obtidos nos testes

Comparamos o *framework* proposto, também, com um sistema baseado em *Function as a Service* (FaaS), utilizando o serviço Lambda[15] da AWS. Com intuito de estimar seus custos financeiros, primeiramente foi necessário obter o tempo de execução do processo de raspagem utilizando este ambiente e a melhor configuração de memória para execução da função Lambda. O Lambda aloca capacidade de CPU na proporção da quantidade de memória configurada, onde esta memória é a quantidade de memória disponível para a função do Lambda ao longo da execução[16]. Encontramos estes valores utilizando a ferramenta AWS Lambda Power Tuning[17].

[15]AWS Lambda - https://aws.amazon.com/lambda

[16]Configuring Lambda function options - https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html

[17]Profiling functions with AWS Lambda Power Tuning - https://docs.aws.amazon.com/lambda/latest/operatorguide/profile-functions.html

Utilizamos a configuração indicada pela ferramenta como *Best Cost* (melhor custo) com 768 MB de memória e tempo de execução de aproximadamente 7 segundos. Com esses valores e com o número de execuções extraído dos cenários de teste (*Tamanho do batch* x *Número de batches*), utilizando a calculadora de preços da AWS[18] conseguimos estimar os custos quando empregamos FaaS.

O tempo médio de processamento do pedido nos sistemas foi calculado utilizando as métricas exportadas por cada instância do *cluster* para o Cloudwatch, onde cada instância ativa exporta sua média atual e o número de mensagens processadas (requisições de raspagens) com sucesso ao fim de cada *batch*. A função orquestradora é responsável por agregar esses dados, gerando assim o tempo final. Já para o Lambda, considerando que o tempo médio de execução ficou em torno de 7 segundos, e que podemos desconsiderar seu limite de concorrência por se tratar de um serviço FaaS, definimos o tempo médio para este mesmo valor.

Analisando a Tabela 4 observa-se que o sistema *Burstable - 1* conseguiu se manter dentro do $Tsla$ e atingir uma redução de custos de 94,5%, 96,0% e 93,75% em relação ao *Lambda* nos cenários de alta, média e baixa utilização, respectivamente. Também obtivemos uma redução de 94,83%, 95,59% e 93,26% quando comparado ao *On-demand*.

Verificamos que *Burstable - 2* atinge marcas bem menores no tempo médio de processamento do pedido do que o sistema *Burstable - 1* devido ao uso de duas instâncias fixas. Além do aumento na capacidade de lidar com rajadas durante a execução dos testes, existiram momentos onde o valor de N era 1, porém o *Burstable 2* possui no mínimo duas instâncias e, por conta desta máquina a mais, conseguiu processar os pedidos em menor tempo.

3.2.1. Análise do melhor caso

Nas Figuras 2, 3, 4 e 5 encontramos mais detalhes sobre os resultados da execução no sistema *Burstable - 1* durante o cenário *Utilização alta*. Este sistema obteve o melhor custo dentro deste cenário. Todos os gráficos estão relacionados ao tempo de execução do teste em segundos.

Analisando a Figura 2 é perceptível a tendência do sistema de aproximar o tempo médio de processamento ao $Tsla$, isso ocorre pois ele é configurado para calcular o número de instâncias necessárias para que as mensagens sejam atendidas exatamente cumprindo esse *deadline*. Isto também justifica o acúmulo de mensagens, antes de seu declínio na Figura 3, pois o *framework* verifica que consegue manter um certo número de pedidos na fila sem que o tempo de processamento ultrapasse o *deadline*.

A relação entre o número de instâncias no *cluster* (N) e o número de mensagens na fila (M) pode ser encontrada nas Figuras 3 e 4, onde o *framework* permite o acúmulo de um certo número de mensagens até o momento em que verifica a necessidade de aumentar a quantidade de instâncias para que estas mensagens sejam resolvidas dentro do $Tsla$.

O consumo de créditos na instância do *cluster burstable* fixo em relação ao tempo de execução está representado na Figura 5, sendo notável o decréscimo dos mesmos com

[18] AWS Pricing Calculator - https://calculator.aws/#/createCalculator/Lambda

o decorrer do caso de teste, o que reafirma a aptidão desse tipo de instância para problemas com fluxo intermitente.

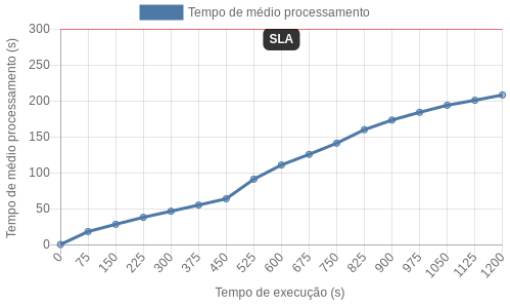

Figura 2. Tempo médio de processamento (s) x Tempo (s), sistema *Burstable - 1* em Utilização alta

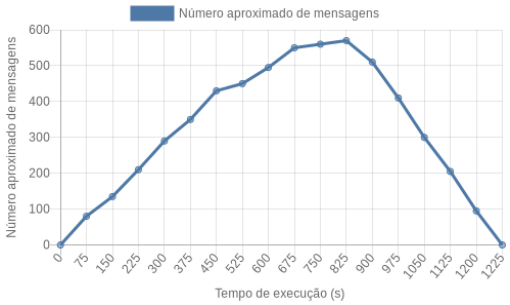

Figura 3. Número de mensagens x Tempo (s), sistema *Burstable - 1* em Utilização alta

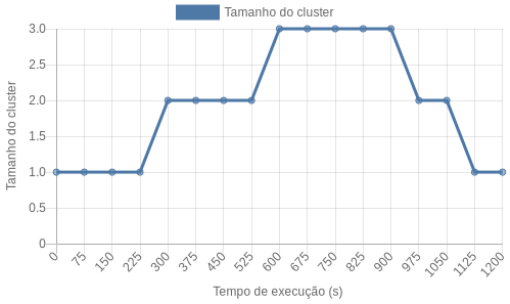

Figura 4. Tamanho do cluster x Tempo (s), sistema *Burstable - 1* em Utilização alta

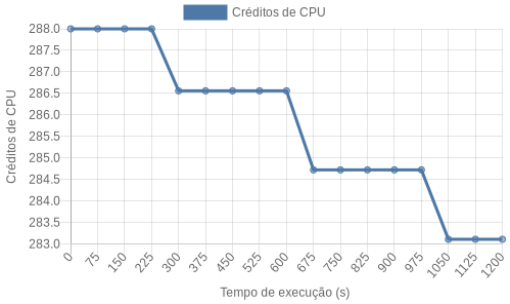

Figura 5. Créditos de CPU x Tempo (s), sistema *Burstable - 1* em Utilização alta

4. Conclusão

Este trabalho propõe um *framework* de *Web Scraping* na nuvem AWS aproveitando a qualidade expansível das instâncias *burstable* para processar pedidos de raspagem dentro do tempo limite definido e ainda otimizando o custo financeiro. O *framework* define um *cluster* misto, com instâncias fixas e temporárias, com possibilidade de variação da quantidade de instâncias de maneira a garantir o SLA das mensagens recebidas e obter o menor custo possível. O *framework* proposto foi avaliado na nuvem AWS, com uma e duas instâncias *burstable* fixas e comparado a uma abordagem apenas com instâncias *on-demand* não expansíveis e também a outra solução baseada em FaaS. Os resultados mostraram que todos os testes atendem o SLA definido, alcançando uma redução de 96% de custo financeiro, em seu melhor caso quando comparado à abordagem FaaS, e redução de 95,59%, em seu melhor caso quando comparado à abordagem *on-demand*. Além disso, houve redução de custos em todos os demais casos de teste, com menores reduções para o modelo com duas instâncias *burstable* fixas em comparação com o modelo de apenas uma máquina.

Para futuros trabalhos, seria interessante adicionar mecanismos de controle de rajada das instâncias *burstable*, a fim de aproveitar ainda mais os créditos disponíveis dessas VMs. Dessa forma, o *framework* poderia categorizar as instâncias expansíveis em estados de geração ou utilização de créditos e controlar, por exemplo, a quantidade de casos processados em paralelo pela instância. Por fim, buscando também minimizar custo, é interessante o estudo do problema combinatório gerado para entender quais os valores ótimos dos parâmetros de configuração do *framework*. A taxa de entrada pode ser descrita como uma distribuição para que, a partir disto, alguns parâmetros, como o de número de máquinas virtuais ativas N e o tamanho do *batch* de mensagens, possam ser mais precisamente definidos.

Referências

Baarzi, A. F., Zhu, T., and Urgaonkar, B. (2019). Burscale: Using burstable instances for cost-effective autoscaling in the public cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 126–138, New York, NY, USA. Association for Computing Machinery.

Chaulagain, R. S., Pandey, S., Basnet, S. R., and Shakya, S. (2017). Cloud based web scraping for big data applications. In *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 138–143.

Dantas, J., Khazaei, H., and Litoiu, M. (2021). Bias autoscaler: Leveraging burstable instances for cost-effective autoscaling on cloud systems. In *Proceedings of the Seventh International Workshop on Serverless Computing (WoSC7) 2021*, WoSC '21, page 9–16, New York, NY, USA. Association for Computing Machinery.

Jiang, Y., Shahrad, M., Wentzlaff, D., Tsang, D. H., and Joe-Wong, C. (2019). Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1576–1584.

Teylo, L., Arantes, L., Sens, P., and Drummond, L. (2021). Scheduling bag-of-tasks in clouds using spot and burstable virtual machines. *IEEE Transactions on Cloud Computing*, pages 1–1.

Wang, C., Urgaonkar, B., Nasiriani, N., and Kesidis, G. (2017). Using burstable instances in the public cloud: Why, when and how? *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1).

Woodall, R., Kline, D., Modaresnezhad, M., and Vetter, R. (2021). A cloud-based system for scraping data from amazon product reviews at scale. In *Proceedings of the Conference on Information Systems Applied Research*, Washington DC, USA.

Zhao, B. (2017). Web scraping. In Schintler, L. A. and McNeely, C. L., editors, *Encyclopedia of Big Data*, pages 1–3, Cham. Springer International Publishing.