No cenário da computação em nuvem, o modelo de serviço FaaS (function as a service, ou função como serviço) vem ganhando adoção rapidamente. Nesse modelo de serviço, os aplicativos em nuvem são estruturados como módulos de código independentes chamados funções, que são instanciadas sob demanda, a tarifação é baseada no número de invocações de funções e no tempo de execução das funções. Os desenvolvedores são atraídos pelo FaaS porque ele promete superar duas desvantagens dos modelos de serviço tradicionais de IaaS e PaaS, a necessidade de provisionar e gerenciar infraestrutura, e a necessidade de pagar por recursos não utilizados. Na prática, no entanto, as coisas são um pouco menos otimistas: os desenvolvedores ainda precisam escolher a quantidade de memória alocada para as funções, e a visibilidade dos custos é baixa, principalmente porque estão atrelados ao desempenho das funções. Este trabalho experimental investiga o comportamento do desempenho e do custo em plataformas FaaS. Foram analisadas as variações de desempenho e custo dentro da mesma plataforma e entre plataformas distintas. Os resultados mostram que o desempenho e o custo podem ser significativamente afetados pela escolha da alocação de memória, do provedor FaaS e da linguagem de programação: foram observadas diferenças de até 8,5× no desempenho e 67× no custo entre os provedores (com a mesma linguagem e tamanho da memória) e 16,8× no desempenho e 67,2× no custo entre linguagens de programação (com o mesmo provedor e memória). Também verificou-se empiricamente que, mesmo quando uma dada configuração apresenta variações significativas de desempenho, a variabilidade do custo permanece pequena. O estudo apresenta ainda uma comparação de custos entre FaaS e IaaS, que revela que a vantagem econômica do modelo FaaS é inversamente proporcional ao volume mensal de requisições processadas.

Orientador: Prof. Dr. Rafael Rodrigues Obelheiro

JOINVILLE, 2019

ANO 2019

DIOGO BORTOLINI

INVESTIGANDO DESEMPENHO E CUSTO EM PLATAFORMAS FAAS

UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA – PPGCA

DISSERTAÇÃO DE MESTRADO

INVESTIGANDO DESEMPENHO E CUSTO EM PLATAFORMAS FAAS

DIOGO BORTOLINI

JOINVILLE, 2019

DIOGO BORTOLINI

INVESTIGANDO DESEMPENHO E CUSTO EM PLATAFORMAS FAAS

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, para a obtenção do grau de Mestre em Computação Aplicada.

Orientador: Dr. Rafael Rodrigues Obelheiro

JOINVILLE

2019

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Setorial do CCT/UDESC,

com os dados fornecidos pelo(a) autor(a)

Bortolini, Diogo
Investigando Desempenho e Custo em Plataformas FaaS / Diogo Bortolini. -- 2019.
61 p.

Orientador: Rafael Rodrigues Obelheiro
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2019.

1. Computação em nuvem . 2. avaliação de desempenho. 3. custo. 4. modelo de serviço FaaS. I. Obelheiro, Rafael Rodrigues. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

Investigando Desempenho e Custo em Plataformas FaaS

por

Diogo Bortolini

Esta dissertação foi julgada adequada para obtenção do título de

Mestre em Computação Aplicada

Área de concentração em "Ciência da Computação",
e aprovada em sua forma final pelo

CURSO DE MESTRADO ACADÊMICO EM COMPUTAÇÃO APLICADA
DO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA
UNIVERSIDADE DO ESTADO DE SANTA CATARINA.

Banca Examinadora:

Prof. Dr. Rafael Rodrigues Obelheiro
CCT/UDESC (Orientador/Presidente)

Prof. Dr. Ricardo Jose Pfitscher
UNISOCIESC

Prof. Dr. Mario Antonio Ribeiro Dantas
UFJF (videoconferência)

Joinville,SC, 09 de dezembro de 2019.

Dedico este trabalho aos meus familiares, amigos, colegas e professores que me acompanharam e me deram forças nessa magnífica trajetória.

AGRADECIMENTOS

Agradeço aos meus pais, por me proporcionarem a educação e pelo incentivo durante minha trajetória acadêmica. À minha esposa e filho pelo apoio, força e amor incondicional. Não serei capaz de retribuir todo carinho, amor e incentivo que recebi de vocês.

Um agradecimento especial ao Professor Doutor Rafael Rodrigues Obelheiro, por compartilhar sua sabedoria e o seu tempo, pelas correções realizadas, por toda a paciência, confiança, empenho e incansável dedicação com que sempre me orientou neste trabalho.

Sou grato a todos os professores que contribuíram com este processo, especialmente aos professores Doutor Guilherme Piêgas Koslovski e Doutor Mauricio Aronne Pillon pelos ensinamentos, sugestões, elogios e críticas durante os seminários do mestrado, na banca de qualificação e banca examinadora.

A esta universidade, seu corpo docente, direção e administração, em especial ao Departamento de Ciência da Computação (DCC) e a Coordenadoria de Informática (CINF), que oportunizaram o horizonte superior que hoje vislumbro.

E a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho.”

Dalai Lama

RESUMO

No cenário da computação em nuvem, o modelo de serviço FaaS (*function as a service*, ou função como serviço) vem ganhando adoção rapidamente. Nesse modelo de serviço, os aplicativos em nuvem são estruturados como módulos de código independentes chamados funções, que são instanciadas sob demanda, a tarifação é baseada no número de invocações de funções e no tempo de execução das funções. Os desenvolvedores são atraídos pelo FaaS porque ele promete superar duas desvantagens dos modelos de serviço tradicionais de IaaS e PaaS, a necessidade de provisionar e gerenciar infraestrutura, e a necessidade de pagar por recursos não utilizados. Na prática, no entanto, as coisas são um pouco menos otimistas: os desenvolvedores ainda precisam escolher a quantidade de memória alocada para as funções, e a visibilidade dos custos é baixa, principalmente porque estão atrelados ao desempenho das funções. Este trabalho experimental investiga o comportamento do desempenho e do custo em plataformas FaaS. Foram analisadas as variações de desempenho e custo dentro da mesma plataforma e entre plataformas distintas. Os resultados mostram que o desempenho e o custo podem ser significativamente afetados pela escolha da alocação de memória, do provedor FaaS e da linguagem de programação: foram observadas diferenças de até 8,5× no desempenho e 67× no custo entre os provedores (com a mesma linguagem e tamanho da memória) e 16,8× no desempenho e 67,2× no custo entre linguagens de programação (com o mesmo provedor e memória). Também verificou-se empiricamente que, mesmo quando uma dada configuração apresenta variações significativas de desempenho, a variabilidade do custo permanece pequena. O estudo apresenta ainda uma comparação de custos entre FaaS e IaaS, que revela que a vantagem econômica do modelo FaaS é inversamente proporcional ao volume mensal de requisições processadas.

Palavras-chaves: Computação em nuvem, avaliação de desempenho, custo, modelo de serviço FaaS.

ABSTRACT

In cloud computing, the function-as-a-service (FaaS) service model has been gaining adoption at a fast pace. In this service model, cloud applications are structured as self-contained code modules called functions that are instantiated on-demand, and billing is based on the number of function invocations and on function execution time. Developers are attracted to FaaS because it promises to remove two drawbacks of the traditional IaaS and PaaS service models, the need to provision and manage infrastructure, and the need to pay for unused resources. In practice, however, things are a little less rosy: developers still have to choose the amount of memory allocated to functions, and costs are less predictable, especially because they are tied to function performance. This experimental work investigates the performance and cost behavior of FaaS platforms. We analyzed performance and cost variations within and across FaaS providers. Our results show that performance and cost can be significantly affected by the choice of memory allocation, FaaS provider, and programming language: we observed differences of up to 8.5× in performance and 67× in cost between providers (with the same language and memory size), and 16.8× in performance and 67.2× in cost between programming languages (with the same provider and memory). We provide empirical evidence that, even in configurations that exhibit significant performance variations, cost variability remains small. Our study also presents a cost comparison between FaaS and IaaS that shows that the economical advantage of the FaaS model is inversely proportional to the monthly volume of requests processed.

Key-words: FaaS, cost and perfomance.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CPU	*Central Processing Unit*
CV	Coeficiente de Variação
EC2	*Elastic Compute Cloud*
FaaS	*Function-as-a-Service*
GCE	*Google Cloud Engine*
GCF	*Google Cloud Functions*
IA	Inteligência Artifical
IaaS	*Infrastructure-as-a-Service*
IoT	*Internet of Things*
kpps	Mil Primos por Segundo
MV	Máquina Virtual
NIST	*National Institute of Standards and Technology*
NRM	Número de Requisições por Mês
PaaS	*Plataform-as-a-Service*
QP	Questão de Pesquisa
SaaS	*Software-as-a-Service*
TCO	*Total Cost of Ownership*

SUMÁRIO

1 INTRODUÇÃO

A computação em nuvem revolucionou como infraestruturas computacionais são gerenciadas, acessadas e utilizadas. Os clientes de nuvem acessam, através da Internet, recursos virtualizados – servidores, armazenamento, bancos de dados, e uma variedade de serviços de aplicação – hospedados em infraestruturas físicas adquiridas e gerenciadas pelos provedores. Esses recursos são alocados e reconfigurados sob demanda, de acordo com a capacidade e a quantidade exigida pelo cliente, que é tarifado de acordo com os recursos utilizados (VAQUERO et al., 2008).

As nuvens computacionais oferecem diversos modelos de serviço, que descrevem como diferentes tipos de recursos são oferecidos como serviços pelo provedor. Atualmente, os modelos mais estabelecidos são *Software-as-a-Service* (SaaS), *Platform-as-a-Service* (PaaS), e *Infrastructure-as-a-Service* (IaaS) (BUYYA et al., 2018). No SaaS, o cliente da nuvem acessa através da Internet aplicações que estão hospedadas em *data centers* de nuvem. Nesse modelo, o cliente é tipicamente um usuário final, não um desenvolvedor, embora também seja possível oferecer serviços que podem ser incorporados em outras aplicações. O modelo PaaS é mais voltado para usuários que demandam maior controle sobre os recursos computacionais utilizados, e oferece um arcabouço para criação e implantação de aplicações de nuvem que incorpora características como modelos de programação e escalabilidade automática. O modelo IaaS oferece acesso a recursos computacionais elementares, como processamento (máquinas virtuais e/ou contêineres), rede e armazenamento, que são configurados e gerenciados pelo cliente. A camada IaaS tem sido a base da computação em nuvem: além de fornecer a infraestrutura para os modelos SaaS e PaaS, ela permite que clientes tenham acesso a recursos computacionais na quantidade e com a capacidade necessárias, no momento adequado, pagando por aquilo que utilizam, e exercendo grande controle sobre esses recursos.

Uma tendência emergente em nuvens computacionais tem sido a adoção do modelo de serviço FaaS (*Function-as-a-Service*), em que aplicações são construídas como conjuntos de funções que executam em instâncias[1] ativadas sob demanda, em resposta a eventos (ROBERTS, 2018). Essas instâncias são *stateless* e temporárias, podendo permanecer ativas durante a execução de uma única requisição, e gerenciadas inteiramente pelo provedor, que fica responsável por garantir sua escalabilidade e disponibilidade. O cliente é cobrado por invocação, sem pagar por recursos ociosos. Com isso, o modelo FaaS pode propiciar uma economia significativa em relação aos modelos mais tradicionais IaaS e PaaS, ao mesmo tempo em que libera o desenvolvedor de preocupações com a gerência da infraestrutura (ADZIC; CHATLEY, 2017).

De maneira mais específica, a tarifação em FaaS é baseada no número de invocações

[1] Seguindo (WANG et al., 2018), o termo *instância* será usado para denotar o contêiner ou máquina virtual em que uma função é executada.

e no produto entre o tamanho da alocação de recursos e o tempo de execução da função. Isto faz com que o custo em FaaS seja mais variável do que o custo nos modelos IaaS/PaaS, em que a tarifação depende da capacidade alocada (EIVY, 2017). Além disso, vários estudos (LEITNER; CITO; STÖCKLI, 2016; EIVY, 2017; JONAS et al., 2017) apontam que a visibilidade dos custos é baixa, ou seja, é difícil para um cliente de nuvem antecipar quanto pagará pelos serviços.

Um desenvolvedor tem poucas opções para influenciar o desempenho e o custo das funções. A mais citada é a quantidade de memória alocada para a função, pois algumas plataformas FaaS, como AWS Lambda e Google Cloud Functions, alocam a CPU proporcional ao tamanho da memória (LLOYD et al., 2018; WANG et al., 2018; FIGIELA et al., 2018). Nesse caso, alocar mais memória do que o necessário para executar mais rapidamente pode fazer sentido não apenas em termos de desempenho, mas também em termos de custo. Outro fator é a escolha da linguagem de programação, que pode ter implicações não apenas em termos de velocidade do código, mas também de sobrecarga do tempo de execução e maturidade da plataforma (LEE; SATYAM; FOX, 2018).

Há um conjunto de trabalhos que exploram o desempenho do modelo FaaS, com foco em escalabilidade, concorrência e discussões com base no tempo de execução das funções (MCGRATH; BRENNER, 2017; LEE; SATYAM; FOX, 2018; LLOYD et al., 2018; WANG et al., 2018; JONAS et al., 2017). Algumas referências (LEITNER; CITO; STÖCKLI, 2016; ADZIC; CHATLEY, 2017; EIVY, 2017; ROGERS, 2017) discutem os custos em FaaS com base principalmente em modelos analíticos, e sem se preocupar com como o custo pode ser afetado por variações no desempenho das funções. Por fim, outros abordam questões de desempenho e custo (VILLAMIZAR et al., 2017; HOROVITZ et al., 2018; FIGIELA et al., 2018). Este trabalho visa a contribuir com esta discussão, explorando de forma mais abrangente de que modo as características das funções afetam o desempenho e o custo percebidos pelo cliente de nuvem.

1.1 OBJETIVOS

O objetivo geral deste trabalho é investigar como a escolha de plataforma FaaS influencia no desempenho e no custo das funções, e como o custo pode ser afetado pela variabilidade de desempenho das funções.

Para alcançar o objetivo principal são definidos alguns objetivos específicos conforme a seguir:

- Medir o desempenho em plataformas FaaS com base no provedor selecionado, memória configurada e linguagem de programação utilizada;
- Avaliar experimentalmente os custos médios com base nas distintas configurações e número de invocações;

- Verificar as razões de custo/desempenho entre os provedores de FaaS;
- Analisar como a variabilidade do desempenho influencia na variabilidade do custo;
- Apresentar um estudo de caso comparando o custo de uma solução FaaS com os de uma configuração IaaS com desempenho similar.

Após a proposta de investigação determinada e os objetivos estabelecidos, um conjunto de questões foram definidos para orientar e amparar a pesquisa deste trabalho:

- **QP1** – *Existem variações significativas de desempenho em configurações fixas?*
  Um dos objetivos consiste em medir o desempenho de FaaS e, portanto, esta questão representa medir a variação do tempo de execução (desempenho) quando a função FaaS é executada diversas vezes mantendo as mesmas configurações de provedor, tamanho de memória e linguagem de programação.
- **QP2** – *Quanto o desempenho varia entre plataformas FaaS com o mesmo tamanho de memória e linguagem de programação?*
  Levando em consideração o mesmo objetivo da questão anterior, esta questão pretender verificar se existem variações de desempenho ao migrar a função entre os provedores, mantendo tando a alocação de memória quanto a linguagem de programação.
- **QP3** – *A escolha de linguagem de programação afeta perceptivelmente o desempenho?*
  Ainda com o objetivo de analisar o desempenho do modelo FaaS, a linguagem de programação é outro fator de configuração importante que deve ser considerado.
- **QP4** – *Como as escolhas de plataforma e linguagem de programação afetam o custo?*
  A quarta questão de pesquisa promove a investigação de uma parte dos fatores que podem influenciar o custo, e a magnitude de tal influência.
- **QP5** – *Nos casos em que alocar uma quantidade maior de memória resulta em um melhor desempenho, isso também gera um melhor custo-benefício?*
  Esta questão complementa os resultados da questão anterior ao combinar custo e desempenho em uma única métrica.
- **QP6** – *As variações de desempenho induzem variações de custo na mesma proporção?*
  Esta questão explora se os custos de FaaS são influenciados diretamente e com a mesma relevância que as variações podem ocorrer no desempenho.
- **QP7** – *Como os custos de FaaS e IaaS podem ser comparados para desempenho equivalente?*
  Uma das vantagens alegadas do modelo FaaS é que ele pode proporcionar uma redução de custos quando comparado aos modelos de serviço mais tradicionais, como IaaS e PaaS.

Esta questão de pesquisa visa comparar o custo de FaaS com o custo de uma solução IaaS que oferece um desempenho equivalente, analisando em que cenários o novo modelo é mais vantajoso sob a perspectiva do custo.

1.2 ORGANIZAÇÃO DO TEXTO

Este trabalho está organizado em cinco capítulos. O Capítulo 1 apresentada o contexto e a introdução do problema e os objetivos do trabalho. O Capítulo 2 discute os modelos de nuvem clássico e o modelo FaaS, bem como seus conceitos e características. No Capítulo 3 estão os trabalhos relacionados e os objetivos e experimentos criados. No Capítulo 4 são apresentadas as avaliações experimentais, comparações de desempenho e custo de FaaS. No Capítulo 5 é possível analisar as conclusões obtidas e as perspectivas de trabalhos futuros. O texto contém ainda três apêndices, que apresentam os ambientes das plataformas FaaS utilizadas nos experimentos: AWS Lambda (Apêndice A), Google Cloud Functions (Apêndice B), e IBM Cloud Functions (Apêndice C).

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo faz uma revisão dos principais aspectos de nuvens computacionais necessários ao entendimento do trabalho. A Seção 2.1 apresenta os conceitos e definições de computação em nuvem e os principais modelos de serviços de nuvens. A Seção 2.2 define o recente modelo de nuvem FaaS, suas características, as configurações e particularidades de três dos principais provedores FaaS, a forma de cobrança e uso do modelo.

2.1 MODELOS DE SERVIÇO EM NUVENS COMPUTACIONAIS

A computação em nuvem permite o uso de recursos de computação compartilhados e é uma alternativa aos servidores locais. A computação em nuvem agrupa um grande número de servidores e outros recursos e, geralmente, oferece sua capacidade conjunta sob demanda, com pagamento por ciclo (BHARDWAJ; JAIN; JAIN, 2010). Segundo a entidade norte-americana NIST (*National Institute of Standards and Technology*), a computação em nuvem é:

> um modelo para permitir acesso ubíquo, conveniente e sob demanda via rede a um agrupamento compartilhado e configurável de recursos computacionais (por exemplo, redes, servidores, equipamentos de armazenamento, aplicações e serviços), que pode ser rapidamente fornecido e liberado com esforços mínimos de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE et al., 2011).

Os provedores de computação em nuvem oferecem diferentes níveis de flexibilidade e controle conforme a necessidade do cliente. Tais abstrações são oferecidas em modelos de serviço que podem ser retratados como camadas em uma pilha: infraestrutura, plataforma e software. Os modelos clássicos que representam cada uma dessas camadas são, respectivamente: IaaS, PaaS e SaaS. Portanto o cliente poderá optar por modelos que fornecem apenas acesso a serviços e/ou aplicações até o gerenciamento de ambientes de desenvolvimento e recursos. As características de cada um desses modelos são:

- Infraestrutura como Serviço (IaaS — *Infrastructure as a Service*) possibilita ao cliente de nuvem provisionar processamento, armazenamento, comunicação de rede e outros recursos de computação fundamentais nos quais o cliente pode instalar e executar softwares em geral, incluindo sistemas operacionais e aplicativos. O consumidor tem controle sobre os sistemas operacionais, armazenamento, e aplicativos instalados, e possivelmente um controle limitado de alguns componentes de rede (MELL; GRANCE et al., 2011). De forma a complementar, Bhardwaj, Jain e Jain (2010) mencionam que IaaS geralmente fornece o hardware e serviços administrativos necessários para armazenar e uma plataforma para executar os aplicativos. Dimensionamento de largura de banda, memória e armazenamento são geralmente incluídos. O provedor oferece o serviço através de um

contrato ou com o pagamento conforme o uso. Exemplos de IaaS são: Amazon EC2, Google Cloud Engine e DigitalOcean.

- Plataforma como Serviço (PaaS — *Platform as a Service*) proporciona ao cliente a possibilidade de instalar na infraestrutura da nuvem aplicativos criados ou adquiridos pelo cliente. Este último tem o controle sobre as aplicações instaladas e possivelmente configurações do ambiente de hospedagem (MELL; GRANCE et al., 2011). A principal diferença operacional entre o PaaS e o FaaS (que será visto com maiores detalhes a seguir) é a escala. Geralmente com um PaaS, é necessário pensar em como escalar. Com um aplicativo FaaS, isso é completamente transparente. Mesmo que o aplicativo PaaS seja configurado para dimensionar automaticamente, isso não será feito no nível de solicitações individuais (ROBERTS, 2018). Exemplos de PaaS são: SAP Cloud Platform, Google App Engine e Red Hat OpenShift.
- Software como Serviço (SaaS — *Software as a Service*) disponibiliza ao cliente o uso de aplicações do fornecedor executando em uma infraestrutura na nuvem. O cliente da nuvem controla apenas algumas configurações da aplicação. (MELL; GRANCE et al., 2011). Exemplos de produtos SaaS são: Google G Suite, Adobe Creative Cloud e Dropbox.

Conforme os estudos de Costello (2019), o mercado mundial de serviços de nuvem pública continuará crescendo nos próximos anos. Entre os modelos clássicos apresentados anteriormente, SaaS ainda é o que produz a maior receita mundial em nuvem, seguido por IaaS. O modelo SaaS é voltado para usuários finais e, portanto, não é o serviço adequado para desenvolvedores que queiram oferecer suas aplicações e funções na nuvem. O estudo indica também que IaaS será o segmento de mercado em nuvem computacional que mais crescerá em receita, portanto IaaS será utilizado para fins de comparação com o modelo FaaS.

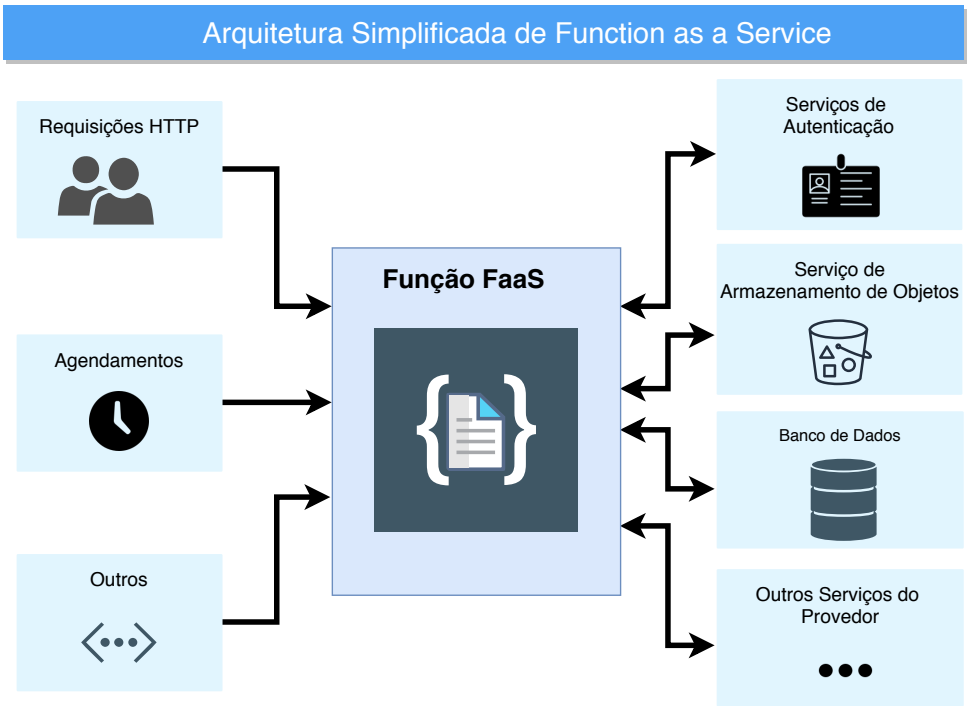
2.2 PLATAFORMAS FAAS

No contexto da nuvem, o conceito de computação *serverless* significa que os desenvolvedores não precisam se preocupar com servidores, ou seja, fornece soluções que abstraem a forma como foram construídas nos servidores ou como o dimensionamento é feito (ROBERTS, 2018). Função como Serviço (FaaS — *Function as a Service*) é um modelo de nuvem computacional emergente com conceito *serverless* que foi anunciado pela primeira vez em novembro de 2014 (AWS, 2014). Esse modelo permite a implantação de software na forma de funções executadas na infraestrutura do provedor em resposta a eventos específicos, como novos arquivos sendo enviados para um armazenamento de dados na nuvem, mensagens chegando em sistemas de filas, ou chamadas HTTP. O desenvolvedor implanta seu código no formato de uma função para o provedor de nuvens, enquanto o gerenciamento dos recursos é fornecido pela plataforma, de forma automatizada e escalável (MALAWSKI et al., 2017).

Um exemplo de uso de FaaS é um aplicativo de servidor que processe apenas uma solicitação a cada minuto, gastando 50 ms para processar cada solicitação, e gerando um uso médio da CPU em uma hora de 0,1%. Se esse aplicativo for implantado em seu próprio servidor dedicado, isso será ineficiente. Com FaaS o aplicativo do exemplo seria tarifado por apenas 100 ms (tempo mínimo de execução da maioria dos provedores FaaS) de computação a cada minuto, ou seja, 0,15% do tempo total. Otimizações de desempenho no código aumentarão a velocidade do aplicativo, mas também fornecerão imediata redução de custos. Por exemplo, se um aplicativo inicialmente leve um segundo para processar e, com a otimização do código, for reduzido para 200 ms, a imediata economia pode ser de 80%, sem alterações na infraestrutura (ROBERTS, 2018).

A Figura 1 mostra a arquitetura simplificada de uma função FaaS sendo executada. O cliente FaaS realiza o *upload* do código ou escreve-o diretamente no editor oferecido pelo provedor, e define a quantidade de memória que será alocada e o tempo máximo que a função poderá levar para executar. A invocação da função é associada a eventos específicos, como requisições HTTP, agendamentos, entre outros. O código pode executar qualquer lógica de negócio, acessar serviços externos ou integrar-se a outros serviços oferecidos pelo provedor, por exemplo. Ao final, o provedor fornece registros com os resultados das execuções de tais funções (AWS, 2017).

Figura 1 – Arquitetura simplificada da execução de uma função FaaS.



Fonte: Elaborada pelo autor, 2019

O modelo de execução e provisionamento de FaaS permite muitos casos de uso, a seguir alguns exemplos típicos:

- **Microsserviços**: são serviços pequenos e autônomos que trabalham juntos. A arquitetura de microsserviços baseia-se na ideia de decompor a aplicação em funções básicas, que podem ser criadas, atualizadas e implantadas de maneira independente (NEWMAN, 2015). A natureza modular e inerentemente escalável do FaaS o torna ideal para implementar partes granulares da lógica da aplicação e podem ser implementadas, gerenciadas e escaladas de forma independente. Além disso, em modelos de nuvem computacional clássico, as equipes pequenas e ágeis de microsserviços gastam tempo com complexidades infraestruturais e operacionais, como tolerância a falhas, balanceamento de carga, ajuste automático de escala e criação de logs (IBM CLOUD, 2019).

- **Processamento de Dados**: FaaS pode ser configurado para reagir a mudanças nos dados e executar ações automaticamente. Essas ações englobam transformar dados, enviar e receber mensagens, chamar outras ações e muitas outras coisas, inclusive processamentos em tempo real (IBM CLOUD, 2019). O Seattle Times[1], empresa de mídia familiar que atende ao Noroeste do Pacífico, usa FaaS para redimensionar imagens para visualização no mais diferentes dispositivos, como computadores, *tablets* e *smartphones* (AWS, 2017).

- ***Back-ends***: desenvolvimento responsável, em termos gerais, pela implementação da regra de negócio. Com FaaS, uma camada de conexão lógica permite integrar e estender os serviços, possibilitando a criação rápida de aplicativos altamente disponíveis, seguros e com boa relação custo-benefício (GOOGLE CLOUD, 2019c). É possível utilizar FaaS em diferentes modelos de *back-ends*, como por exemplo, *back-ends* para integração com outros servidores ou APIs (*Application Programming Interface*), *back-ends* para dispositivos móveis e aplicativos *web*. O Bustle Digital Group[2], detentor de sites e aplicativos de notícias, entretenimento, estilo de vida e moda voltado para as mulheres, com acesso médio mensal de 50 milhões de pessoas, é um exemplo da utilização de FaaS para processamento de grandes volumes de dados estatísticos do site, permitindo que a equipe possa tomar decisões com base neste dados (AWS, 2017).

- **Internet das Coisas**: IoT (*Internet of Things*) é a presença perversiva de uma variedade de coisas ou objetos – como etiquetas de identificação por radiofrequência (RFID), sensores, atuadores, veículos, telefones celulares entre outros dispositivos – que são capazes interagir uns com os outros (GIUSTO et al., 2010). Por exemplo, uma ação no FaaS poderá ser acionada se houver necessidade de reagir a um sensor que excedeu uma determinada temperatura. Como as interações IoT são geralmente *stateless*, com uma possível

[1] <http://www.seattletimes.com/>
[2] <https://www.bustle.com/>

necessidade de manipular vários eventos simultâneos sem aviso prévio, como em desastres naturais ou engarrafamentos. FaaS permite um sistema elástico em que a carga de trabalho normal pode ser pequena, mas que talvez necessite de um escalonamento rápido com tempo de resposta previsível (IBM CLOUD, 2019). Um exemplo é Smart Parking[3], empresa australiana pioneira em estacionamentos inteligentes, que utiliza FaaS combinado com uma rede de sensores, *displays* e *gateways* que registram a chegada de um veículo, geram informações ao vivo sobre o uso do ambiente de estacionamento e gerenciam a capacidade, apresentando orientações e sinalizações automatizadas (GOOGLE CLOUD, 2019c).

- **Aplicativos Inteligentes**: com uso da Inteligência Artificial (IA), aprendizagem de máquina (*machine learning*) e computação cognitiva, ocorre o surgimento de aplicativos poderosos. Exemplos são os assistentes virtuais e *chatbots*, análises de vídeos e imagens e análise de sentimentos. FaaS pode efetivamente ser combinado outros serviços de IA para extrair automaticamente informações úteis dos vídeos sem precisar assisti-los. Outro exemplo seria usar FaaS para análise de sentimento em mensagens de texto e envio de texto ou emoji específico com base no que foi detectado.

As três plataformas FaaS entre as mais populares que foram alvos de experimentos deste trabalho são: AWS Lambda [4], Google Cloud Functions[5] e IBM Cloud Functions[6]. As características mais relevantes de cada plataforma, incluindo os modelos de tarifação, são apresentadas na tabela 1.

Tabela 1 – Características e modelos de precificação das plataformas FaaS.

	AWS	Google	IBM
Memória	$64k$ MB ($k = 2,3,\ldots,47$)	$128k$ MB ($k = 1,2,4,8,16$)	$32k$ MB ($k = 4,5,\ldots,64$)
CPU	proporcional à memória	proporcional à memória	não informado
Rede (bytes transferidos/mês)	1 GB	5 GB	não informado
Armazenamento Local	512 MB	500 MB	não informado
Linguagens Suportadas	Node.js (JavaScript), Python, Java, Ruby C# (.NET Core), Go	Node.js, Python, Go	JavaScript, Swift, Python, PHP, Java, Go, Ruby, .Net Core e outras
Tamanho máximo da função	50 MB (comprimido) e 250 MB (sem compressão)	100 MB (comprimido) e 500 MB (sem compressão)	48 MB
Tempo de Execução máximo	15 minutos	9 minutos	10 minutos
Granuralidade da cobrança	100 ms	100 ms	100 ms
Custo Mensal (USD)	USD 0,20/1M execuções, USD 0,0000166667/GB-s	USD 0,40/1M execuções, USD 0,0000025/GB-s, USD 0,0000100/GHz-s	USD 0,000017/GB-s
Faixa gratuita (por mês)	400,000 GB-s, 1 M execuções	400,000 GB-s, 2 M execuções	400,000 GB-s

Fonte: Elaborada pelo autor, 2019

[3] <https://www.smartparking.com/>
[4] <https://aws.amazon.com/lambda/>
[5] <https://cloud.google.com/functions/>
[6] <https://www.ibm.com/cloud/functions>

O cliente de um provedor FaaS desenvolve suas funções como módulos de código que implementam uma determinada lógica de aplicação. Essas funções são empacotadas e carregadas em um serviço de armazenamento na nuvem. Quando uma função é invocada, seu código é carregado e executado em uma ou mais instâncias (tipicamente contêineres), de acordo com o volume de requisições. Quando há vários eventos simultâneos de invocação, o provedor executa mais cópias da função em paralelo. Depois que a função termina, sua instância torna-se ociosa. Para evitar a latência de ativação de novas instâncias, instâncias ociosas podem ser reaproveitadas para atender a novas requisições, mas não há garantias de que isso ocorra (AWS, 2017). Para armazenamento temporário, cada instância tem acesso a um espaço no disco local, que será removido quando a instância for desativada; para armazenamento persistente, a opção mais comum é usar um dos vários serviços oferecidos pelo provedor, que são cobrados à parte. Cada função tem um tempo máximo de execução, que varia de 9 minutos (Google) até 15 minutos (AWS).

Em todas as três plataformas, o cliente precisa especificar a quantidade de memória a ser alocada para uma instância, sendo que o intervalo vai de 128 MB até 2048 MB (Google, IBM) ou até 3008 MB (AWS). A granularidade de alocação também é variável, conforme pode ser visto na Tabela 1. A quantidade de CPU alocada é proporcional à memória alocada para AWS e Google, e não informada para IBM.

De modo geral, a tarifação dos provedores FaaS tem dois componentes. O primeiro é o número de invocações de funções, considerando todas as funções pertencentes ao mesmo usuário. Essa é uma métrica fácil de compreender, ainda que seja dependente da arquitetura da aplicação: uma aplicação subdividida em um número grande de funções com granularidade fina induz um número maior de invocações do que uma aplicação com um número menor de funções com granularidade mais grossa (EIVY, 2017). Dentre os provedores considerados, o único que não cobra pelo número de invocações é a IBM.

O segundo componente de tarifação é o consumo de recursos. Esse consumo é mensurado em GB-s, que é o produto da memória alocada para a função, expressa em GB, pelo seu tempo de execução, expresso em segundos. Por exemplo, uma função de 512 MB de memória que execute durante 200 ms tem um consumo de $0{,}5\,\text{GB} \times 0{,}2\,\text{s} = 0{,}1\,\text{GB-s}$. Um ponto importante é que as granularidades de alocação de memória e de tempo de execução variam entre os provedores. Por exemplo, uma função que necessite de 140 MB de memória irá exigir uma alocação de 160 MB na IBM, 192 MB na AWS e 256 MB no Google. Com relação ao tempo de execução, este é arredondado para cima em múltiplos de 100 ms; portanto, uma função que leve 99 ms para executar será cobrada por 100 ms, e uma função que leve 101 ms será cobrada por 200 ms. No Google, o consumo de recursos tem um componente adicional, o provisionamento de CPU, que é medido em GHz-s e dado pelo produto entre o tempo de execução e a alocação de ciclos de CPU, conforme mostrado na Tabela 2.

Os três provedores proporcionam faixas gratuitas de 400.000 GB-s a cada mês. AWS

Tabela 2 – Provisionamento de CPU no Google Cloud Functions

memória (MB)	128	256	512	1024	2048
CPU (GHz)	0,2	0,4	0,8	1,4	2,4

Fonte: Elaborado pelo autor, 2019

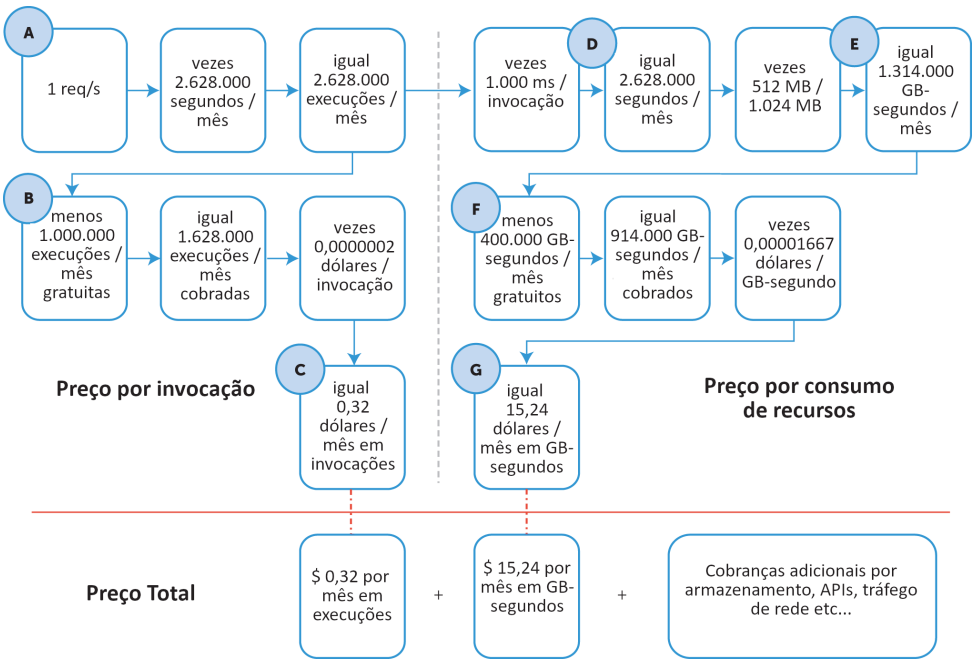
e GCF também incluem um número de invocações gratuitas (1 e 2 milhões, respectivamente, por mês). Como a IBM não cobra pelo número de execuções, sua faixa de gratuidade considera apenas o consumo de recursos.

A Figura 2 mostra um exemplo de cálculo do custo mensal estimado para uma função em AWS Lambda. Os parâmetros de entrada são *(i)* a taxa média de chegada de requisições λ, *(ii)* o tempo médio de execução da função (arredondado para cima em multiplos de 100 ms), e *(iii)* a quantidade de memória alocada. O exemplo considera uma função com tempo médio de execução de 1 s e 512 MB de memória, com taxa média de chegada de $\lambda = 1$ req/s. O cálculo inicia (*A*) multiplicando a quantidade média de segundos em um mês (aproximadamente 2.628.000 s) por λ, resultando no número de requisições por mês *(NRM)*. Esse volume de requisições é usado para calcular o custo por número de invocações e o custo por consumo de recursos. Para o primeiro, *NRM* é subtraído pelo número de execuções gratuitas *(B)* e o resultado multiplicado pelo valor em dólares de cada invocação tarifada, gerando o custo por invocações *(C)*. A etapa do cálculo do consumo de recursos realiza o produto do tempo médio das execuções em segundos pelo *NRM* e a quantidade de memória alocada em gigabytes *(D,E)*. É descontada a faixa gratuita do consumo de recursos (em GB-segundos/mês) *(F)* e o resultado multiplicado pelo custo em dólares, obtendo o custo dos recursos alocados e consumidos. A soma dos dois componentes de preço gera o preço final ao final do mês (USD 15,56). O Google também realiza a cobrança do consumo de CPU, portanto existe um terceiro fator no cálculo que, para as mesmas especificações, seria de USD 21,56. Já a IBM pratica a cobrança apenas dos recursos consumidos *(A,D,E,F,G)*, sendo assim, o custo é de USD 15,56.

Se, por exemplo, o tempo médio de execução exposto no exemplo da Figura 2 sofresse um acréscimo de 20%, os custos totais no mês de AWS, Google e IBM passariam a ser de USD 19,94, USD 26,42 e USD 20,01, respectivamente. Isso representaria um aumento no custo de aproximadamente 28% para AWS e IBM e de 22,5% para o Google; em todos os provedores, o aumento no custo é superior ao aumento no tempo de execução (em termos porcentuais).

Os exemplos apresentados ajudam a reafirmar a relevância deste trabalho. Os clientes FaaS têm dificuldade em entender e planejar os custos devido aos vários componentes de custo (volume de requisições, consumo de recursos), as faixas de gratuidade e as diferenças entre os provedores. Além disso, os custos associados a consumo de recursos podem ser influenciados diretamente pela variabilidade no tempo de execução das funções e pela granularidade de medida desse tempo.

Figura 2 – Exemplo de cálculo do custo total com AWS Lambda considerando uma função com tempo médio de execução de 1 s, 512 MB de memória alocada e taxa média de chegada de requisições de 1 req/s.



Fonte: Adaptado de Eivy (2017, p. 10)

2.3 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram apresentados os conceitos dos modelos tradicionais de computação em nuvem e do emergente FaaS. Para este último foram apresentadas as principais características e diferenças entre os provedores. Além disso, a arquitetura simplificada de invocação e execução de um função, informações e procedimentos dos cálculos da tarifação de FaaS foram exibidos e fatores que compõem o preço: número de invocações, custo por invocação, tempo de execução médio da função, quantidade de memória alocada e custo dos recursos alocados. No Capítulo 3 os trabalhos relacionados são discutidos e a metodologia e detalhes da proposta são apresentados.

3 PROPOSTA

Este capítulo contextualiza e apresenta a proposta do trabalho. A Seção 3.1 faz uma revisão de trabalhos relacionados, identificando a lacuna no estado da arte explorada nesta pesquisa. A Seção 3.2 descreve a metodologia proposta e os objetivos de avaliação deste trabalho.

3.1 TRABALHOS RELACIONADOS

Os trabalhos relacionados podem ser divididos em três grupos de referências. O primeiro grupo se concentra no desempenho de FaaS, o segundo grupo se concentra nos custos de FaaS, o terceiro grupo aborda o desempenho e o custo.

No primeiro grupo, (MCGRATH; BRENNER, 2017) relatam diferenças de escalabilidade (vazão em diferentes níveis de concorrência), latência de *cold start* (tempo necessário para que uma nova instância seja instanciada antes de executar a função), comportamentos de expiração de instância nos provedores FaaS, incluindo AWS Lambda, Azure Functions, Google Cloud Functions, e Apache OpenWhisk (percursor da IBM Cloud Functions). Nos experimentos, a alocação de memória foi fixada em 512 MB, exceto no Azure, onde a alocação de memória é ajustada automaticamente. AWS Lambda apresenta uma escalabilidade linear, com a maior vazão com 15 solicitações simultâneas. O Google tem comportamento sub-linear que diminui conforme aumenta o nível de concorrência, Azure tem comportamento variável e OpenWhisk tem comportamento curioso, com baixa vazão até oito solicitações simultâneas e, apartir daí apresenta comportamento sub-linear.

Lee, Satyam e Fox (2018) realizaram experimentos em quatro provedores de FaaS (AWS, Azure, GCF e IBM) para medir a influência da concorrência no desempenho das funções. Eles encontraram diferenças significativas entre os provedores, com a AWS sendo a mais madura e alcançando uma vazão consideravelmente mais alta do que as outras. Eles também mediram a sobrecarga dos tempos de execução da linguagem de programação para Node.js, C#, Python, Java e Swift, descobriram que *(i)* AWS tinha os custos mais baixos e mais constantes e *(ii)* as diferenças entre as linguagens de programação eram pequenas, exceto no Azure.

Lloyd et al. (2018) mediram vários aspectos de desempenho com AWS Lambda e no Azure Functions. Dentre eles, o que mais se aproxima deste trabalho é como o tamanho de alocação de memória influencia o desempenho na AWS Lambda. Eles descobriram que aumentar a alocação de memória em 12× (de 128 MB para 1536 MB) melhorou o tempo de execução da função em apenas 4× para instâncias que precisam ser instanciadas e 1,55× para instâncias já em execução. Isso está em contraste com o aumento esperado de 12× para uma alocação de CPU perfeitamente proporcional à alocação de memória.

Wang et al. (2018) investigaram o desempenho em AWS, Azure e GCF, e encontraram problema com o isolamento entre funções e *tenants*, levando a desempenho imprevisível e riscos de segurança. Eles mensuraram a utilização de CPU e a relação dos ciclos de CPU conforme a quantidade de memória RAM alocada, demonstraram que os provedores (AWS, GCF e Azure) alocam CPU proporcionalmente ao tamanho da memória. Os autores ainda mostraram como o *cold start* exerce forte influência sobre o tempo de execução, geralmente diminui à medida que a memória configurada aumenta.

Jonas et al. (2017) descrevem PyWren, uma plataforma para análise de dados massivos implementada em AWS Lambda. O artigo conclui que liberar o desenvolvedor de gerenciar a infraestrutura oferece ganhos significativos de produtividade, e que o desempenho das funções, embora aceitável, apresenta uma variabilidade perceptível.

Pawlik, Figiela e Malawski (2019) realizam testes de desempenho em FaaS com um conjunto de 5120 execuções. O estudo relata que AWS apresenta proporção entre memória e desempenho, com exceção dos testes com 3008 MB de memória alocada, que apresentaram resultados semelhantes aos testes realizados com 2048 MB (a justificativa apresentada é que 3008 MB é uma oferta recente, portanto, ainda compartilha os mesmos recursos de CPU que ao alocar 2048 MB). Google também apresenta proporção entre CPU e memória e IBM apresenta o mesmo desempenho, independentemente da configuração da memória.

No segundo grupo, Eivy (2017) apresenta uma discussão abrangente sobre os custos envolvidos na adoção do modelo FaaS. Em particular, ele destaca que: *(i)* a tarifação em FaaS é de difícil compreensão; *(ii)* variações no tempo de execução de funções podem representar variação nos custos; e *(iii)* como funções são *stateless*, elas representam apenas parte do custo de uma aplicação, que provavelmente terá custos adicionais com armazenamento, rede e outros serviços do provedor. O enfoque do trabalho é mais conceitual que quantitativo, custos concretos são discutidos analiticamente em um cenário restrito, com um único provedor (AWS Lambda) e mantendo constantes o tempo de execução, a memória dimensionada e a taxa de requisições de uma função.

Adzic e Chatley (2017) também discutem o custo associado a FaaS. O trabalho aponta que deixar a escalabilidade e a disponibilidade das aplicações a cargo do provedor tende a gerar uma economia significativa, pois evita que o cliente tenha de gastar com o provisionamento de capacidade de reserva e instâncias de *backup*, recursos que muitas vezes permanecem ociosos. O artigo apresenta como estudos de caso duas aplicações que migraram para o modelo FaaS: a primeira, que migrou de PaaS, teve redução de custo da ordem de 66%, enquanto a segunda, que migrou de IaaS, teve custos reduzidos em mais de 95%. O trabalho também apresenta uma comparação limitada de custos, considerando um único provedor (AWS Lambda) e mantendo constantes o tempo de execução e a taxa de requisições de uma função.

Leitner, Cito e Stöckli (2016) discutem a adoção do modelo FaaS na indústria por meio

de revisão de literatura, entrevistas e questionários com desenvolvedores de aplicações para nuvem. Apresenta um modelo de custo abrangente para aplicações baseadas em microsserviços implementados tanto com IaaS/PaaS quanto com FaaS. O modelo é difícil de parametrizar, e para FaaS considera que o custo por requisição é fixo, ignorando a variabilidade de desempenho.

Rogers (2017) introduz um modelo analítico de custos de FaaS e o utiliza para avaliar como os custos são afetados pelo número de chamadas, alocação de memória e tempo de execução em quatro provedores (AWS, Azure, GCF e IBM). Ele mostra que diferentes fornecedores têm o menor custo, dependendo dos outros parâmetros. Ele também discute como o FaaS reduz o custo total de propriedade (*Total Cost of Ownership* - TCO) quando comparado ao IaaS/PaaS e compara os custos de FaaS e IaaS.

No terceiro grupo, Villamizar et al. (2017) faz um estudo experimental comparando o desempenho e o custo de uma mesma aplicação com diferentes arquiteturas: monolítica, microsserviços e baseada em funções. Embora o estudo mostre uma redução de custo na versão FaaS, ele considera um único provedor (AWS Lambda) com dois tamanhos de memória (duas funções com 320 MB e outra duas com 512 MB) e uma taxa fixa de requisições.

Horovitz et al. (2018) propõem FaaStest, uma solução para otimização de custo e desempenho de serviços FaaS. A solução usa aprendizado de máquina para selecionar dinamicamente a plataforma de menor custo (FaaS ou IaaS) para uma aplicação de acordo com o comportamento de chamadas de funções. Além disso, quando o FaaS é a escolhida, uma função de previsão é usada para estimar quando as funções serão chamadas e antecipar a execução da instância para reduzir *coldstart* dos serviços de FaaS. Os experimentos apresentam uma redução de mais de 50% no custo e mais de 90% no tempo de resposta para as chamadas em FaaS.

Figiela et al. (2018) avaliaram o desempenho e o custo em quatro provedores de FaaS (AWS, Azure, GCF e IBM). Em termos de desempenho, eles avaliaram o tempo de execução em *benchmarks* de ponto flutuante e inteiros, tempos de transferência de arquivos de/para armazenamento em nuvem, tempo de vida da instância e heterogeneidade da infraestrutura (por exemplo, diversidade de *hosts*). Na avaliação de custos no *benchmarks* de inteiros (que basicamente é uma função Node.js que chama um gerador de números aleatórios escrito em C), os autores descobriram que a AWS e a IBM forneciam desempenho consistente e que apenas o primeiro aloca CPU em proporção à memória. Por outro lado, o tempo de execução do Google Cloud Functions possui distribuições bimodais com maior dispersão. Em relação ao IBM Cloud Functions, o desempenho não depende do tamanho da função e a distribuição é bastante estreita, já o Azure tem uma distribuição muito mais ampla e os tempos médios de execução são relativamente mais lentos, o que pode ser atribuído a diferentes hardwares, mas também ao sistema operacional subjacente (Windows) e à tecnologia de virtualização. Em termos de custo, eles concluem que na AWS ele é independente do tamanho da alocação de memória, enquanto no GCF e na IBM as funções menores são mais baratas.

O presente trabalho pode ser comparado mais diretamente aos estudos que analisam o desempenho e os custos no modelo FaaS, como (VILLAMIZAR et al., 2017; FIGIELA et al., 2018), particularmente o último. Villamizar et al. (2017) compararam o desempenho e o custo de um aplicativo em nuvem específico, mas não avaliaram como as diferentes configurações afetam o desempenho e/ou o custo. Figiela et al. (2018) avaliaram o desempenho e o custo em diferentes configurações, mas não consideraram como a escolha da linguagem de programação influencia tais aspectos. Somente Lee, Satyam e Fox (2018) avaliaram o impacto das linguagens de programação no desempenho, mas mediram a sobrecarga dos tempos de execução da linguagem, não sua influência no tempo de execução e/ou custo. Portanto, este é o primeiro estudo que investiga como a escolha de provedor, alocação de memória e linguagem de programação afeta o desempenho e o custo no modelo FaaS.

3.2 METODOLOGIA PROPOSTA

Avaliar desempenho e custo em plataformas FaaS é um problema que requer uma abordagem experimental. Ainda que seja concebível abordar o problema sob uma perspectiva analítica, isso exigiria uma base de dados empíricos que não estão disponíveis. Jain (1991) apresenta uma metodologia amplamente conhecida e adotada para estudos de avaliação de desempenho. A avaliação experimental conduzida neste trabalho está alinhada a essa metodologia, tendo como principais aspectos:

- delimitação do escopo do estudo;
- seleção de métricas;
- planejamento dos experimentos.

O estudo tem como objetivo avaliar o desempenho e o custo em plataformas FaaS, e como a variabilidade do desempenho pode afetar o custo. Dada a inexistência de *benchmarks* específicos para FaaS, torna-se necessário definir um componente (ou conjunto de componentes) que seja objeto de experimentação. Esse objeto de experimentação deve ter as seguintes características:

- **Independência de serviços adicionais**: aplicações FaaS tipicamente combinam funções fornecidas pelo cliente (que implementam a lógica da aplicação) com serviços fornecidos pelo provedor (como bancos de dados e armazenamento de arquivos). Assim, o desempenho e o custo de uma aplicação são determinados tanto pelo desempenho/custo de suas funções quanto pelo desempenho/custo dos serviços do provedor (incluindo o tráfego de rede, normalmente tarifado por volume). Por outro lado, os mesmos serviços de provedor são costumeiramente usados por aplicações em nuvem que adotam outros modelos de serviço, como PaaS e IaaS. Tendo em vista o objetivo do estudo, faz-se necessário

restringir o seu escopo à medição do desempenho e do custo de funções que não dependam de outros serviços do provedor, para considerar apenas os aspectos intrínsecos das plataformas FaaS. O desempenho e o custo dos serviços do provedor são ortogonais ao modelo de serviço adotado, e podem ser mensurados separadamente.

- **CPU-*bound* com baixo consumo de memória**: como em algumas plataformas FaaS a alocação de CPU é influenciada pela alocação de memória (LLOYD et al., 2018; WANG et al., 2018; FIGIELA et al., 2018), surge a hipótese de que, em determinados casos, pode ser vantajoso alocar mais memória do que o necessário, para obter melhor desempenho com o mesmo custo ou a um custo inferior (caso o desempenho superior se traduza em um menor consumo de recursos). Para examinar essa hipótese, deve-se escolher uma função que requeira pouca memória (podendo assim ser executada com qualquer uma das alocações de memória disponíveis) e cujo desempenho seja altamente dependente de processamento (ou seja, uma função CPU-*bound*), para que seja possível observar a influência da alocação de CPU.

- **Portabilidade**: para avaliar como a escolha de linguagem de programação afeta o desempenho e o custo, é conveniente que a função possa ser facilmente escrita em qualquer linguagem, e não tenham dependência de recursos específicos de nenhuma linguagem.

As métricas adotadas para análise de desempenho serão tempo de execução e vazão (*throughput*). Com relação ao custo, será considerado o custo mensal, em dólares, para uma dada taxa média de chegada de requisições. Para avaliar a relação entre custo e desempenho, será usada uma métrica derivada que relaciona custo e vazão; essa métrica será detalhada na Seção 4.1.

Os fatores considerados nos experimentos serão o provedor FaaS, a quantidade de memória alocada para a função, e a linguagem de programação adotada. Desempenho e custo serão mensurados sob duas perspectivas. A primeira perspectiva analisa o quanto essas métricas podem variar dentro de uma mesma configuração, isto é, considerando os fatores fixos. A segunda perspectiva considera a variação das métricas em função da variação dos fatores, mostrando como a escolha de configuração pode afetar desempenho e custo.

A metodologia adotada neste trabalho oferece uma visão abrangente do desempenho e do custo de plataformas FaaS. Ela pode ser facilmente replicada e adaptada para uso com outras funções, cargas de trabalho, e provedores FaaS.

3.3 CONSIDERAÇÕES DO CAPÍTULO

O modelo de nuvem computacional FaaS é recente e desde seu efetivo surgimento alguns trabalhos relacionados têm se dedicado a estudar as características, comportamentos e sugerir melhorias e adequações para este modelo. As referências bibliográficas apresentadas

neste capítulo podem ser agrupadas em três grupos: estudos relacionados ao desempenho, estudo de custos de FaaS e trabalhos que analisam desempenho e custo. Dada a inexistência de uma abordagem mais específica da avaliação de desempenho e custo e como a variabilidade do desempenho pode afetar o custo, estes serão os objetos de estudo deste trabalho.

O capítulo apresentou ainda a metodologia proposta para o estudo, discutindo o seu escopo, as métricas utilizadas e os experimentos adotados. O Capítulo 4 traz uma descrição detalhada dos experimentos e a análise dos resultados obtidos.

4 AVALIAÇÃO EXPERIMENTAL

Este capítulo apresenta a avaliação experimental das plataformas FaaS. A Seção 4.1 descreve em detalhes os experimentos realizados. A Seção 4.2 compara o desempenho em diferentes configurações para um mesmo provedor e entre os provedores de FaaS, e a Seção 4.3 faz a comparação dos custos. A Seção 4.4 traz uma discussão geral dos resultados.

4.1 DESCRIÇÃO DOS EXPERIMENTOS

Conforme discutido na Seção 2.2, um aplicativo no modelo FaaS combina funções fornecidas pelo cliente com serviços oferecidos pelo provedor (como armazenamento e mensagens). Assim, o desempenho do aplicativo tem dois componentes principais, desempenho da função e desempenho do serviço, e este trabalho concentra-se no primeiro. A maioria dos serviços em nuvem não é específica para o FaaS e, portanto, seu desempenho e custo podem ser avaliados separadamente para obter um panorama completo do desempenho/custo do aplicativo.

Para atingir o modelo proposto na Seção 3.2, foi necessário o desenvolvimento de uma função que dependesse mais de CPU (*CPU-bound*), utilizando pouca memória RAM e que pudesse ser facilmente migrada para qualquer linguagem de programação. A alocação de memória é o único parâmetro que o cliente FaaS consegue configurar de forma a melhorar o desempenho da instância, levando em consideração os provedores que alocam ciclos de CPU em relação à quantidade de memória configurada. Por isso, uma função *CPU-bound* faz sentido na análise de desempenho.

A função implementada usa um algoritmo ineficiente para calcular o enésimo número primo. O Algoritmo 1 mostra a versão Python da função. O código testa todos os números inteiros a partir de 1, incluindo os números pares. O teste de primalidade (função `isPrime()`) também é ineficiente, verificando se existe um divisor do número candidato entre os inteiros no intervalo $[2, \sqrt{n}]$. A própria função mede o seu tempo de execução, tornando essas medições independentes de *cold/warm start*. A função satisfaz os requisitos de dependência de CPU, portabilidade e baixo consumo de memória, e permite que o tempo de execução seja controlado ajustando o valor de n.

Para minimizar imprecisões na medição do tempo e evitar grandes flutuações de custo devido ao arredondamento dos tempos de execução, n foi ajustado para obter um tempo de execução da ordem de alguns segundos, na melhor das hipóteses. Os resultados apresentados neste capítulo são para $n = 80.000$, ou seja, a função retorna o octogésimo milésimo número primo.

Avaliou-se o impacto no desempenho e no custo de três fatores: provedor FaaS, me-

Algoritmo 1: Função enésimo número primo em Python

```
import json
import math, sys, time
def main(dict):
    def isPrime(n):
        for i in range(2, int(math.sqrt(n)+1)):
            if n % i == 0:
                return False;
        return n>1;
    tini = time.time()
    n = 80000
    i = 1
    cont = 0
    while (cont < n):
        i = i+1
        if isPrime(i):
            cont = cont+1
    print("RunTime: %.5f s" %(time.time()-tini,n,i))
```

mória alocada e linguagem de programação. Os tamanhos de memória foram restringidos pelo Google, que possui o menor número de opções disponíveis: 128, 256, 512, 1024 e 2048 MB. Quanto à linguagem de programação, implementou-se o algoritmo nas linguagens suportadas pelos três provedores: Go (versão 1.11), Python (versão 3.7) e Node.js (versão 8 no GCF e versão 10 na AWS e IBM). Com cinco tamanhos de memória e três linguagens de programação, há 15 configurações para cada um dos três provedores, ou 45 configurações no total.

Os experimentos foram realizados entre abril e outubro de 2019, nas regiões `us-east (N. Virginia)` (AWS), `us-central-1` (GCF) e `Dallas (Based in CF)` (IBM). Para invocação de funções foram usadas chamadas agendadas. As funções foram agendadas para execução em intervalos de 15 minutos. Os ambientes dos três provedores são descritos em mais detalhes nos Apêndices A (AWS), B (GCF) e C (IBM).

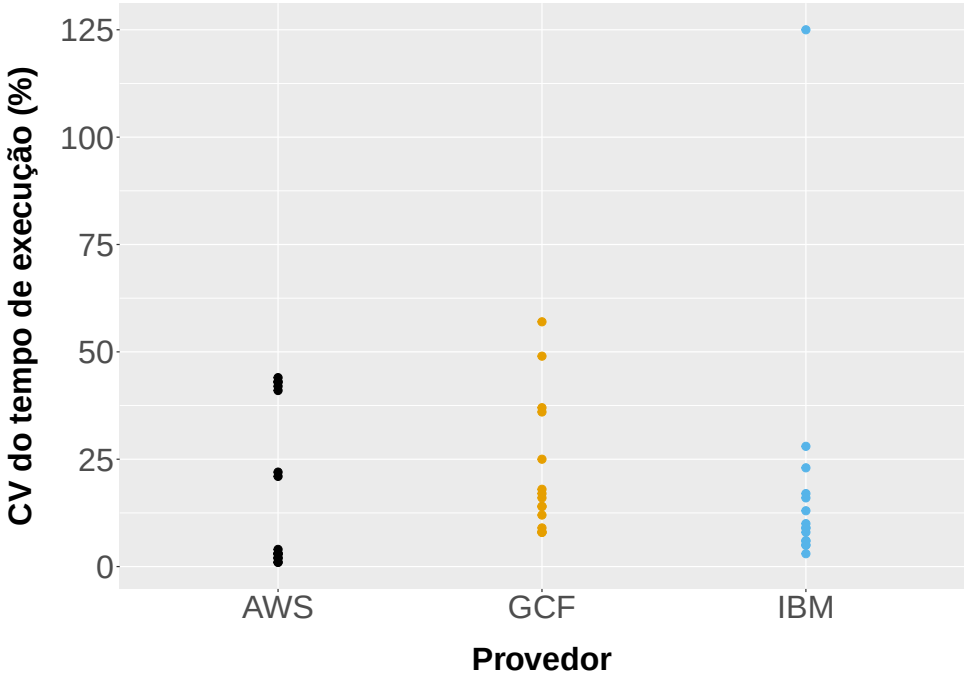
Como o tempo de execução é medido pela própria função, ele não inclui o tempo necessário para ativar uma instância. Em outras palavras, as medições de tempo de execução não são afetadas pelo *cold start* e *warm start*. Além disso, aspectos de isolamento entre instâncias não são levados em consideração.

4.2 COMPARAÇÃO DE DESEMPENHO

Primeiramente foi analisado como o tempo de execução das funções varia para configurações fixas, ou seja, para o mesmo provedor, alocação de memória e linguagem de programação. Foram realizadas 300 invocações da função *benchmark* em cada configuração. A Figura 3 mostra o coeficiente de variação (CV, o desvio padrão dividido pela média) para o tempo de execução, com cada ponto representando o CV para uma determinada configuração.

A AWS tem 8 configurações com CVs entre 1% e 4%, e 7 configurações entre 21% e 44%. O Google tem CVs bem distribuídos entre 8% e 57%. A maioria das configurações da IBM (14 de 15) tem um CV abaixo ou igual a 28%, mas uma das configurações (Node.js, 128 MB) possui um CV de 125%. No geral, houve uma variação significativa de desempenho em todos os provedores: para um terço das configurações (15 de 45), o tempo de execução da função apresentou um coeficiente de variação acima de 20%.

Figura 3 – Coeficiente de variação do tempo de execução da função para todas as configurações.

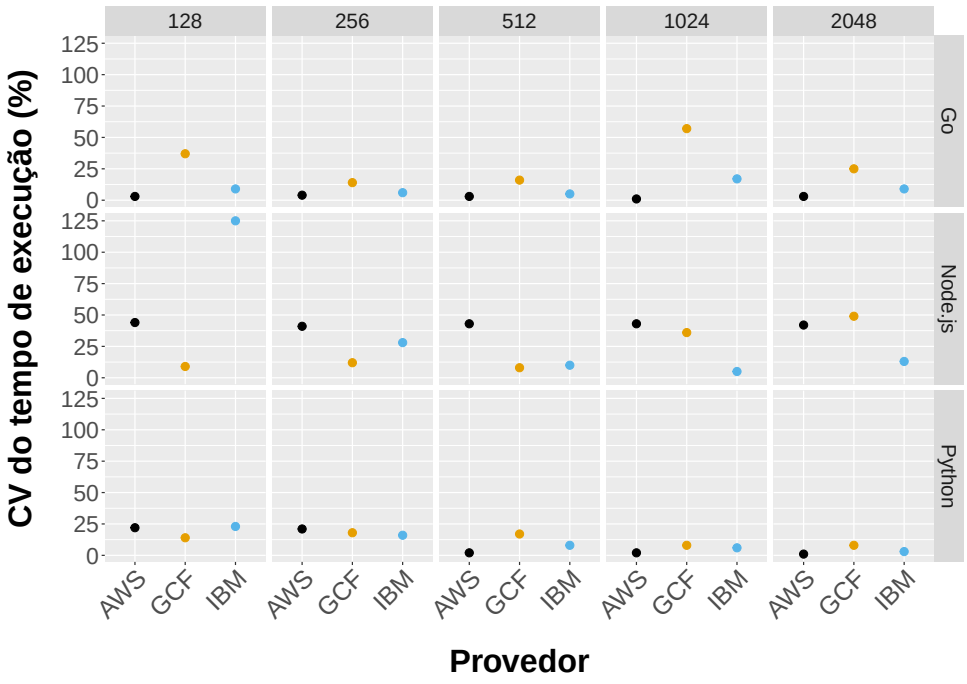


Fonte: Elaborado pelo autor, 2019

A Figura 4 separa o CV do tempo de execução por configuração. Na AWS, a variabilidade foi afetada principalmente pela linguagem de programação, com o Python apresentando o CV mais baixo e o Node.js o mais alto, enquanto a memória apenas apresentou variações do CV quando a linguagem de programação era o Python. O GCF não teve uma tendência discernível. É possível notar uma relação inversa entre o tamanho da memória alocada e a variabilidade quando o Python é utilizado e uma variação maior em Go. Na IBM, a variabilidade foi maior para o Node.js do que para as outras linguagens de programação, incluindo o maior CV que é de 125% com Node.js e 128 MB, conforme mencionado anteriormente.

Para ilustrar a distribuição dos tempos de execução, a Figura 5 mostra os histogramas das configurações com 128 MB e Node.js, usando intervalos de classes (*bins*) de 100 ms. Essas configurações exibiram alta variabilidade na IBM e AWS (CVs de 125% e 44%, respectivamente), e baixa variabilidade no GCF (um CV de 9%). Para todos os provedores os tempos ficaram agrupados em torno de uma única moda, mas com distribuições assimétricas: IBM e AWS apresentaram caudas longas à direita, enquanto GCF teve cauda à esquerda (na realidade,

Figura 4 – Coeficiente de variação do tempo de execução da função para cada configuração.



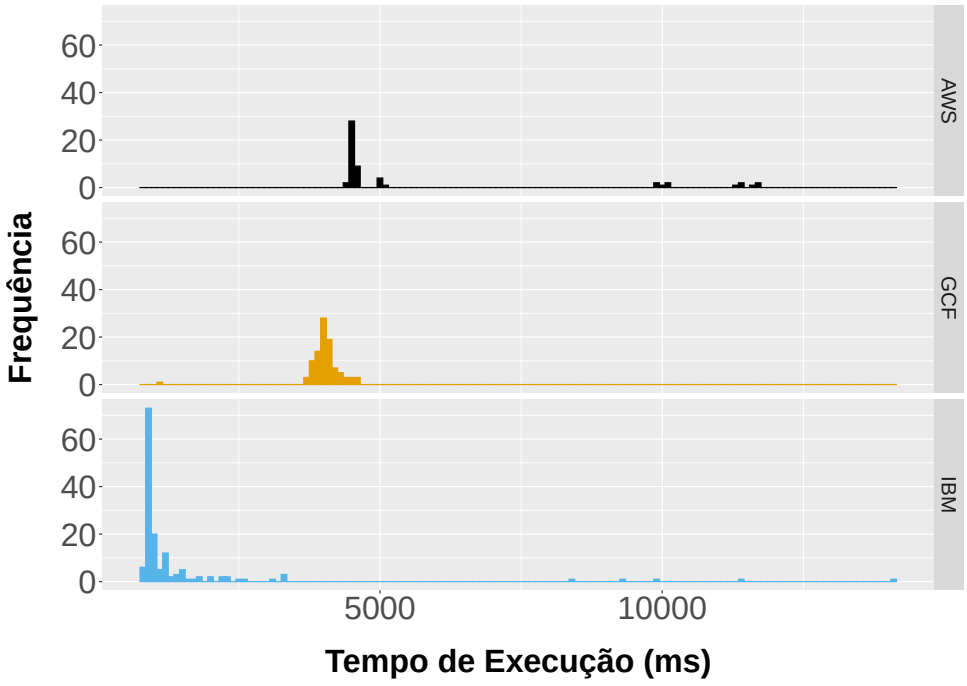
Fonte: Elaborado pelo autor, 2019

apenas uma observação dentre 96). Os histogramas de outras configurações mostram distribuições semelhantes, portanto é possível dizer que, para uma determinada configuração, embora a maioria dos tempos de execução permaneça dentro de um intervalo definido, há uma pequena probabilidade de que eles possam ter grandes variações.

Na sequência ocorre a análise da variação do tempo de execução conforme o provedor FaaS, a memória alocada e a linguagem de programação selecionada; os resultados são mostrados na Figura 6. A IBM teve um desempenho melhor que o AWS e o GCF para todas as linguagens de programação com tamanhos de memória de 128 MB e 256 MB, para Go e Python com 512 MB. Para alocações de memória de 1024 MB o desempenho foi semelhante para todos os provedores e linguagens de programação, a AWS teve melhor desempenho para 2048 MB. Os resultados mostram que a IBM fornece a mesma quantidade de CPU, independentemente do tamanho da memória alocada (exceto com Node.js, onde 128 MB tem desempenho pior que os demais tamanhos), enquanto a AWS e o GCF alocam a CPU na proporção do tamanho da memória (conforme o esperado). Isso é explícito para o GCF, como visto na Tabela 2, e já foi relatado para a AWS (FIGIELA et al., 2018; LEE; SATYAM; FOX, 2018; WANG et al., 2018). Portanto, levando em consideração apenas o desempenho (tempo de execução), a IBM oferece a melhor opção para alocação de memória até 512 MB, enquanto a AWS é a melhor para as maiores alocações de memória.

A Figura 6 também mostra diferenças visíveis de desempenho entre as linguagens de

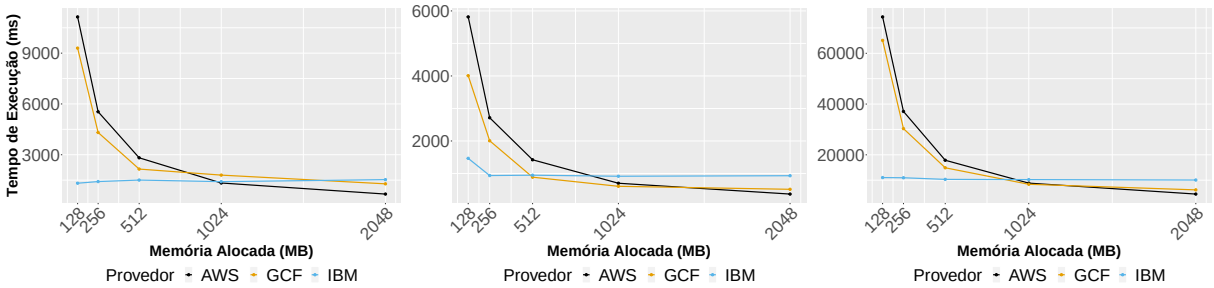
Figura 5 – Histograma do tempo de execução com 128 MB RAM e Node.js.



Fonte: Elaborado pelo autor, 2019

Figura 6 – Tempo de execução da função para diferentes configurações. Observar as diferentes escalas do eixo *y*.

(a) Go (b) Node.js (c) Python



Fonte: Elaborado pelo autor, 2019

programação. A Figura 7 fornece uma melhor visualização para isso, representando as razões do tempo de execução do Go/Python para o Node.js para todas as configurações. Para fins de comparação, também foram realizadas medições em uma máquina de teste com um processador Intel Core i5 6600 de 3,3 GHz e 8 GB de RAM, executando o Ubuntu 18.04.2 LTS; nesta máquina, as razões medidas foram de 1,1 (Go) e 6,3 (Python) em relação a Node.js.

Os resultados mostram que:

1. As razões são diferentes para cada provedor e tamanho de memória;

2. Node.js superou em desempenho o Go e o Python em 14 das 15 combinações de provedor e tamanho de memória alocada (a exceção foi a IBM com 128 MB, onde o Go foi mais

rápido). Nas plataformas FaaS, as razões dos tempos médios de execução de Go/Python para Node.js foram 1,9 e 12,7, respectivamente;

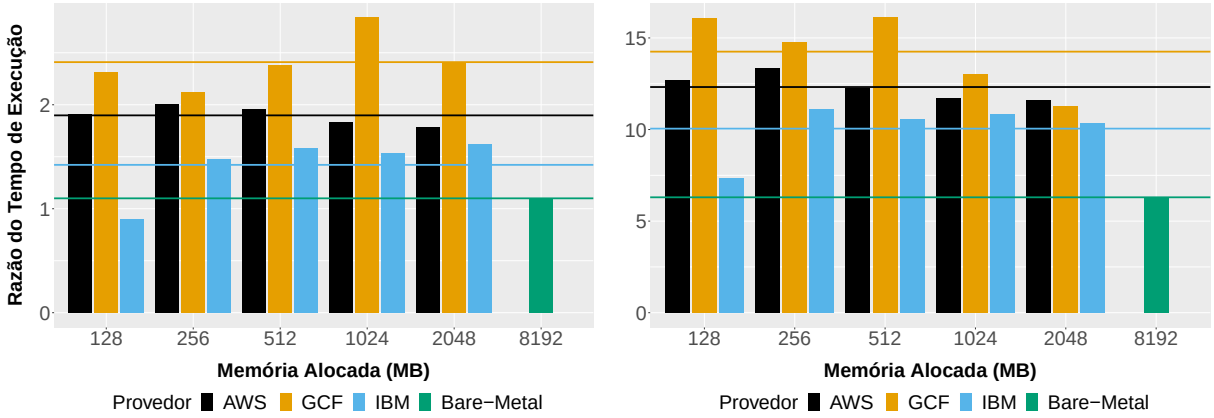
3. Em média, a IBM apresentou menores razões (1,4 e 10,4), enquanto o GCF apresentou as razões mais altas (2,5 e 14,8);

4. As diferenças entre as linguagens de programação são maiores nas plataformas FaaS do que no *bare metal* (com exceção da IBM com 128 MB, onde a versão com Go é mais rápida que a versão Node.js).

Como um todo, esses resultados revelam que *(i)* não é razoável esperar que as diferenças relativas de desempenho observadas em um ambiente de teste permaneçam as mesmas quando o código for executado em uma plataforma FaaS; *(ii)* os provedores FaaS possuem níveis de maturidade distintos no suporte às diferentes linguagens de programação, principalmente quando comparados com os resultados do ambiente *bare metal*; e *(iii)* uma escolha criteriosa da linguagem de programação pode produzir economias significativas nos custos do provedor.

Figura 7 – Razão do tempo de execução médio de Go e Python para Node.js.

(a) Go para Node.js (b) Python para Node.js



Fonte: Elaborado pelo autor, 2019

A maior diferença entre os provedores com a mesma linguagem de programação e tamanho de memória foi de $8,5\times$, para IBM/AWS com Go e 128 MB. A diferença máxima entre linguagens de programação no mesmo provedor e com o mesmo tamanho de memória foi de $16,8\times$, para Go/Python no Google Cloud Functions com 512 MB.

As Figuras 6 e 7 demonstram que o desempenho depende muito da combinação específica de provedor, linguagem de programação e alocação de memória escolhida pelo cliente. Para analisar estatisticamente as contribuições de cada um desses aspectos, foi executado uma ANOVA fatorial (FIELD; MILES; FIELD, 2012, Cap. 12) nos dados coletados e calculada a alocação de variação. Todos os fatores principais (provedor, linguagem programação e memória) e suas interações são estatisticamente significativos ($p < 0,001$). Os fatores principais

respondem por 52,8% da variação observada no tempo de execução, enquanto as interações respondem por 40,3% da variação (os 6,9% restantes são resíduos). O alto percentual de variação devido às interações fornece forte evidência estatística de que os fatores devem ser considerados juntos e não separadamente.

4.3 COMPARAÇÃO DE CUSTO

Esta seção examina como os custos em plataformas FaaS variam de acordo com o provedor, a alocação de memória e a linguagem de programação. Considera-se o custo mensal para uma taxa fixa de requisições λ, expressa em requisições por segundo (req/s). Essa taxa fixa pode ser interpretada como a taxa média de requisições ao longo de um mês, não necessariamente como uma taxa constante. Conforme apresentado na Figura 2 da Seção 2.2 (página 24), a taxa de requisições foi multiplicada pelo número médio de segundos em um mês para obter *NRM*, o número de requisições por mês. O custo devido ao número de invocações da função por mês é obtido multiplicando *NRM* pelo custo por invocação, enquanto o custo devido ao consumo de recursos é calculado usando *NRM*, o tempo médio de execução (arredondado para cima em múltiplos de 100 ms) e a alocação de memória para cada configuração. Para GCF ainda é somando o consumo de CPU, que é calculado usando *NRM*, o tempo médio de execução (também arredondado) e a alocação de ciclos de CPU para cada configuração; em ambos os casos, as faixas gratuitas aplicáveis foram levadas em consideração.

4.3.1 Custo em função da configuração

A Figura 8 mostra o comportamento do custo para uma função com 128 MB de memória quando o tempo de execução varia entre 50 e 5000 ms (em incrementos de 25 ms) em dois cenários, $\lambda = 1$ req/s (fig. 8a) e $\lambda = 150$ req/s (fig. 8b). Em ambos os cenários, as curvas para AWS, GCF e IBM exibem um comportamento serrilhado, devido ao arredondamento do tempo de execução para múltiplos de 100 ms.

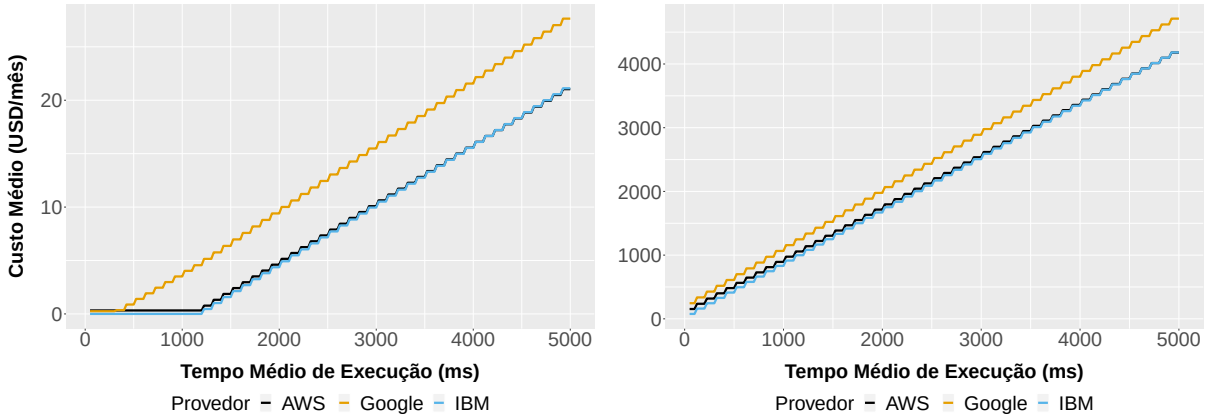
No cenário da Figura 8a, a taxa de requisições é baixa (1 req/s), e é possível observar o efeito das faixas de gratuidade: o custo fica próximo a zero até um determinado tempo de execução. A cobrança por recursos alocados começa a ocorrer primeiro no Google (GCF), que também torna-se o provedor mais caro, com a diferença aumentando com o tempo de execução; em contrapartida, é o segundo provedor mais barato para tempos de execução de até 300 ms. Até 4.000 ms, a IBM apresenta o menor custo entre os provedores, uma vez que não efetua cobrança pelo número de invocações das funções, somente pelos recursos alocados. Por volta de 4.000 ms o custo da AWS passa a acompanhar o custo da IBM, posteriormente se torna mais barato.

O segundo cenário (Figura 8b) tem uma taxa de requisições mais alta, de 150 req/s: essa taxa é a mesma usada em (EIVY, 2017), e foi adotada para facilitar comparações. Este

cenário tem uma tendência semelhante à do anterior: GCF é o provedor com o maior custo e IBM é o mais barato para tempos de execução de função baixos, mas torna-se mais custoso que AWS com tempos de execução maiores.

Figura 8 – Custo médio mensal em relação ao tempo médio de execução.

(a) $\lambda = 1$ req/s (b) $\lambda = 150$ req/s



Fonte: Elaborado pelo autor, 2019

É importante ressaltar que o comportamento relativo demonstrado nos gráficos da Figura 8 repete-se para outros cenários em termos de taxa de requisições e/ou tamanho de memória alocada, mudando apenas de magnitude. O próprio tempo de execução foi variado até 300 s (valor padrão de alguns provedores), mas, como o comportamento segue a tendência observada, esses dados foram omitidos da visualização para dar maior clareza.

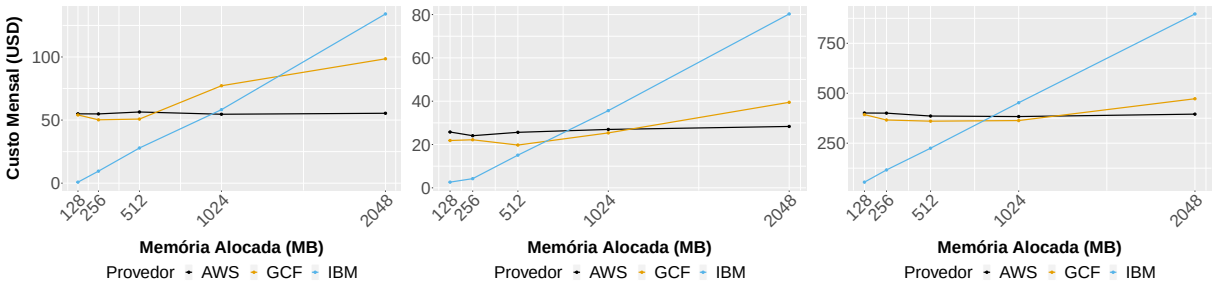
A Figura 9 apresenta o custo mensal médio para a função usada como *benchmark* nas diferentes configurações (escolha de provedor, memória e linguagem de programação). O custo foi calculado com base no tempo médio de execução para cada configuração (resultados da Seção 4.2).

Com relação aos provedores de FaaS, os gráficos mostram que:

1. Na AWS, os custos permanecem praticamente constantes, independentemente do tamanho da memória, pois o custo é inversamente proporcional à memória alocada. Isso con-

Figura 9 – Custo mensal médio ($\lambda = 1$ req/s). Observar as diferentes escalas do eixo y.

(a) Go (b) Node.js (c) Python



Fonte: Elaborado pelo autor, 2019

firma que o poder computacional é proporcional à memória alocada (conforme a Seção 4.2).

2. Na IBM, os custos são proporcionais ao tamanho da memória.

3. No GCF, os custos são mais baixos para 256 e 512 MB do que para 128 MB (ao menos para Go e Python), aumentam para alocações de memória maiores que 512 MB. Uma possível explicação é que, de acordo com a Tabela 2, os aumentos na alocação de memória após 512 MB correspondem a aumentos menores no provisionamento da CPU (por exemplo, passar de 512 MB para 1024 MB fornece o dobro da memória, mas apenas 75% mais ciclos de CPU).

4. A IBM tem o menor custo até 512 MB, AWS e GCF compartilham o menor custo para 1024 MB e a AWS tem o menor custo com 2048 MB.

As variações no custo são significativas. A maior diferença de custo entre os provedores de FaaS com a mesma linguagem de programação e tamanho de memória foi de $67,0\times$ (USD 54,13), para IBM/AWS com Go e 128 MB.

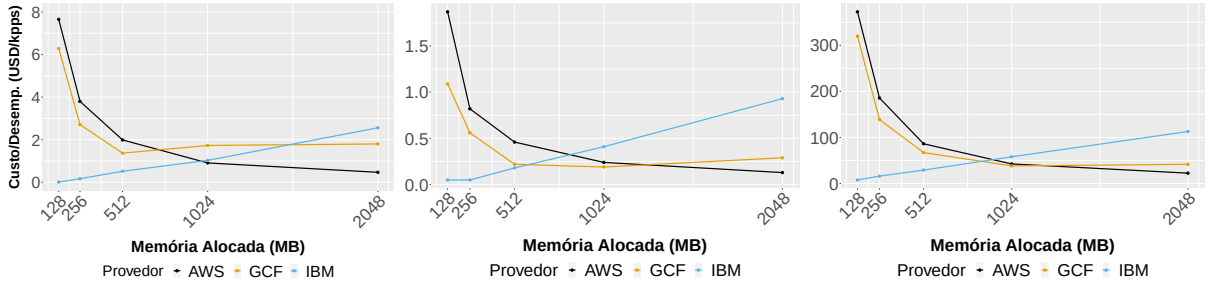
Com relação às linguagens de programação, a Figura 9 mostra que o Node.js tem o menor custo, exceto na IBM com 128 MB, onde Go é mais barato. Os gráficos também mostram que a diferença de custo entre Go e Node.js. ($1,8\times$ em média, com um máximo de $2,8\times$) é muito menor que a diferença entre Python e Node.js ($11,8\times$ em média, com um máximo de $17\times$). A maior diferença entre linguagens de programação no mesmo provedor e com o mesmo tamanho de memória foi de $67,2\times$ (USD 54,25), para Go/Python na IBM com 512 MB.

Os dados de desempenho e custo foram combinados em uma única métrica de custo/desempenho. Um cliente FaaS consciente dos custos se esforçará para minimizar a relação custo/desempenho dentro do conjunto de configurações viáveis para uma determinada função. Como a razão custo/desempenho é expressa em dólares por vazão (JAIN, 1991, Cap. 3), o número de primos calculados pela função (80.000) é dividido pelo tempo médio de execução para obter a vazão de cada configuração em *mil primos por segundo (kpps)*. As curvas de custo/desempenho, mostradas na Figura 10, permitem concluir que: *(i)* as plataformas FaaS com a melhor relação custo/benefício são IBM até 512 MB (todas as linguagens de programação), GCF para 1024 MB (quando utilizado Node.js e Python) e AWS para 1024 MB (quando utilizado Go) e 2048 MB (todas as linguagens de programação); *(ii)* O GCF é menos competitivo para Go do que para Node.js ou Python; e *(iii)* Node.js é a linguagem de programação que possui melhor relação custo/benefício, enquanto o Python apresenta a pior relação custo/benefício.

A melhor relação custo/desempenho (mais baixa) foi de USD 0,01/kpps (na configuração IBM/Go/128 MB), enquanto a pior (mais alta) taxa foi de USD 372,71/kpps (na configuração AWS/Python/128 MB). Tamanha diferença é uma evidência notável de quanto desempenho e custo podem variar entre as plataformas FaaS.

Figura 10 – Razão custo/desempenho. Observar as diferentes escalas do eixo *y*.

(a) Go (b) Node.js (c) Python



Fonte: Elaborado pelo autor, 2019

4.3.2 Impacto das variações de desempenho sobre o custo

Os resultados da Seção 4.3.1 mostram como o custo é afetado pela escolha de provedor FaaS, alocação de memória e linguagem de programação, mas não como o custo de uma dada função efetivamente varia em uma dada configuração. Para avaliar essa variação, realizou-se um experimento no qual foram mensurados o tempo de execução e o tempo de tarifação para 10.000 invocações em cada um dos provedores, com alocação de 128 MB de memória RAM e linguagem de programação Node.js. Os dados coletados foram agrupados em lotes de 1.000 invocações, e calculou-se o CV do tempo de execução e o custo para cada lote, sem levar em consideração as faixas gratuitas. Com isso, é possível observar o quanto a variabilidade de desempenho influencia na variabilidade de custo. A mesma análise foi realizada com lotes de 500 e 2.000 invocações e os resultados foram similares, indicando que a análise é pouco influenciada pelo tamanho do lote; por essa razão, os resultados para outros tamanhos de lotes foram omitidos.

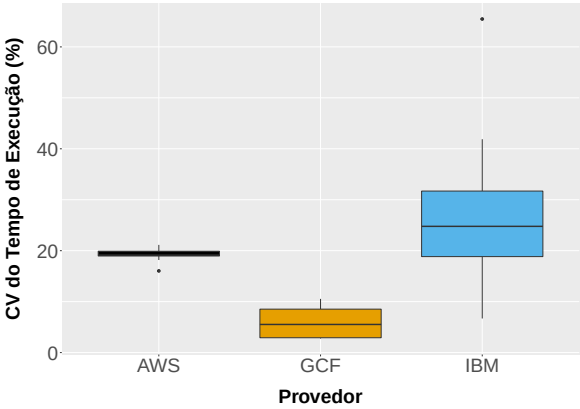
A Figura 11 apresenta *boxplots* para os CVs do tempo de execução e para os custos, sendo um CV e um custo para cada lote de 1.000 execuções. Conforme a Figura 11(a), ao verificar o coeficiente de variação em relação ao tempo de execução, a IBM foi a que apresentou a maior variabilidade e os maiores índices, com um CV máximo de 65,5%, mínimo de 6,7%, terceiro quartil de 31,7% e a média em 27,8%. AWS sugere CVs mais constantes, sendo que o menor CV como sendo 16%, o maior 21,1% e uma média igual a 19,1%. Por fim, GCF demonstra menores valores de variabilidade, sendo 2,8%, 10,5% e 5,9% para o menor, o maior e a média, respectivamente. Tais valores reafirmam a existência de uma variabilidade no desempenho das aplicações em FaaS.

A segunda etapa consistiu em calcular o preço efetivamente tarifado pelos provedores para cada um dos lotes, descartando-se as faixas gratuitas e utilizando os tempos de tarifação (arrendondados). A Figura 11(b) apresenta a distribuição dos custos, sendo possível observar que os custos entre os provedores são diferentes. Entretanto, a variabilidade do custo mensal para cada provedor é baixa. Para AWS, GCF e IBM os CVs foram de 1,69%, 1,24% e 4,92%, respectivamente. Sendo assim, mesmo a IBM atingindo um CV de quase 5% no custo, esse

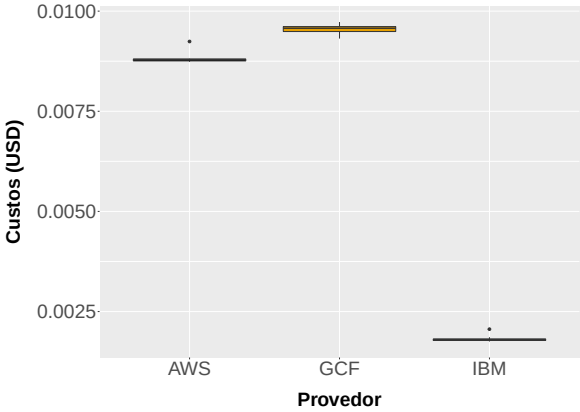
patamar é bem inferior ao CV atingido pelo desempenho. Portanto, a influência que a variação de desempenho provoca no custo mensal de FaaS é pequena. Isso, provavelmente, ocorre devido aos arredondamentos que os provedores realizam no tempo de execução para os múltiplos de 100 ms, minimizando seu efeito nos custos.

Figura 11 – *Boxplots* dos CVs do tempo de execução e dos custos dos provedores.

(a) CVs do tempo de execução



(b) Custos dos provedores



Fonte: Elaborado pelo autor, 2019

Após analisados os aspectos com relação aos custos de FaaS, é importante enfatizar que os cálculos dos exemplos são baseados em um tempo médio de execução, mas que na prática o cliente é cobrado de acordo com o tempo de execução de cada invocação, ou seja, o custo efetivo cobrado pelo provedor consiste no somatório de todas as execuções mensais e recursos utilizados. De forma a verificar se o modelo estimado é válido e pode ser utilizado para representar o custo efetivo de todas as execuções mensais e recursos utilizados. A Tabela 3 apresenta o custo efetivo (isto é, o custo real tarifado pelos provedores) de um experimento com 10 mil execuções de uma determinada função e o custo estimado considerando o mesmo número de invocações e tomando por base o tempo médio de todas as execuções. A diferença entre o custo estimado e o custo efetivo é bastante pequena, precisando ser representada com quatro casas decimais. Isso permite concluir que o modelo de custo ilustrado na Figura 2 (página 24) fornece valores bastante próximos dos reais.

Tabela 3 – Comparação do custo efetivo e do custo estimado conforme o exemplo da Figura 2 (pág. 24)

Provedor	Custo Efetivo (USD)	Custo Estimado (USD)	Diferença (Efetivo − Estimado)
AWS	0,0882	0,0874	+ 0,0008
Google	0,1673	0,1682	− 0,0009
IBM	0,0182	0,0191	− 0,0009

Fonte: Elaborado pelo autor, 2019

4.3.3 Comparação com IaaS

O modelo de serviço FaaS surgiu como uma oferta para um segmento em computação em nuvem que é a construção da solução, no desenvolvimento, sem a necessidade de provisionamento do servidor e pagamento apenas pelo uso efetivo. Ou seja, é importante ressaltar que FaaS não surge para substituir efetivamente, mas como um alternativa para determinadas situações. Portanto, é importante comparar o desempenho e custo de FaaS com outro modelo. Devido às características de uso e importância de mercado, conforme descrito na Seção 2.1, IaaS é o modelo clássico que será comparado com FaaS para verificar situações em que a migração de modelo para outro pode acarretar vantagens.

Como os modelos IaaS e FaaS possuem características específicas de funcionamento, cobranças e limitações, existe a necessidade de equiparar o desempenho de ambos de forma a obter uma comparação de custo mais apropriada e igualitária. Com isso, a função do enésimo primo foi utilizada na linguagem de programação Node.js. As configurações IaaS utilizadas foram MVs de 1 vCPU com 1 GB de memória, com modelos de cobrança mensais e com uso *on demand*, ou seja, utilização sem contratação prévia. O modelo FaaS teve a memória configurada de forma a proporcionar desempenho (tempo de execução) próximo ao obtido pelo produto IaaS de cada um dos provedores.

Uma vez definidas as configurações e garantida a igualdade de desempenho dos modelos de nuvem computacional, o número de requisições por segundo das funções foi variado de forma ascendente. Rogers (2017) propôs e apresentou tal comparação, entretanto, não levou em consideração as limitações de execuções de ambos os modelos. Os provedores FaaS possuem limitação de 1000 execuções concorrentes por padrão (AWS Lambda possui limitações de 500 a 3000, conforme a região). A Equação 4.1 apresenta o cálculo para obter a taxa máxima de requisições por segundo para uma função em um provedor FaaS, respeitando o número máximo de execuções concorrentes e levando em consideração o tempo médio de execução.

$$\lambda_{\text{máx}} < \frac{\text{máximo de execuções concorrentes}}{\text{duração em segundos da função}} \tag{4.1}$$

Em uma máquina virtual com apenas uma vCPU, a execução de uma função *CPU-bound* não pode ser paralelizada, uma vez que tal função utiliza 100% do processador. Isso estabelece um limite para a capacidade de processamento da MV, o qual pode ser modelado usando teoria de filas (JAIN, 1991, cap. 30). Em um modelo de filas, a capacidade de processamento de um servidor é a taxa de serviço $\mu = \frac{1}{E(s)}$, onde $E(s)$ é o tempo de serviço esperado para uma requisição, e a condição de estabilidade $\lambda < \mu$ deve ser respeitada. Embora as medidas obtidas de tempo de execução não permitam precisar $E(s)$, como o objetivo é garantir que o número de execuções fique em um limite factível, $E(s)$ pode ser aproximado pelo menor tempo de execução observado. Assim, a Equação 4.2 apresenta o cálculo para obter o número máximo de execuções por mês *NMEM*, no qual; o valor de 2.628.000 é a quantidade média de segundos

em um mês.

$$NMEM = \frac{1}{\text{menor tempo de execução em segundos}} \times 2.628.000 \tag{4.2}$$

O produto de IaaS da AWS utilizado nos experimentos de comparação foi o EC2 (AWS, 2019a). Em seu catálogo de tipos de instância possui o t2.micro, que oferece algumas horas de instâncias gratuitas por um ano. O tipo testado proporciona no máximo 1 vCPU de até 3,3 GHz[1] e 1 GB de memória RAM com contratação por demanda (existem contratações anuais e trienais que apresentam menos custos). Para AWS Lambda, a configuração de memória que apresenta o desempenho mais próximo do EC2 t2.micro foi a de 640 MB (tempo de execução médio de 808 ms em EC2 contra 796 ms do AWS Lambda, uma diferença de 1,5%).

A região onde AWS Lambda foi executado o limite de concorrência é 3000, como o tempo de execução médio é de 796 ms, conforme a Equação 4.1, o número de requisições deve ser inferior a 3769 req/s. Para AWS EC2 t2.micro, o número máximo de execuções por mês *NMEM*, conforme a Equação 4.2, é de aproximadamente 3.379.195 execuções mensais, uma vez que o menor tempo de execução foi 0,7777 s.

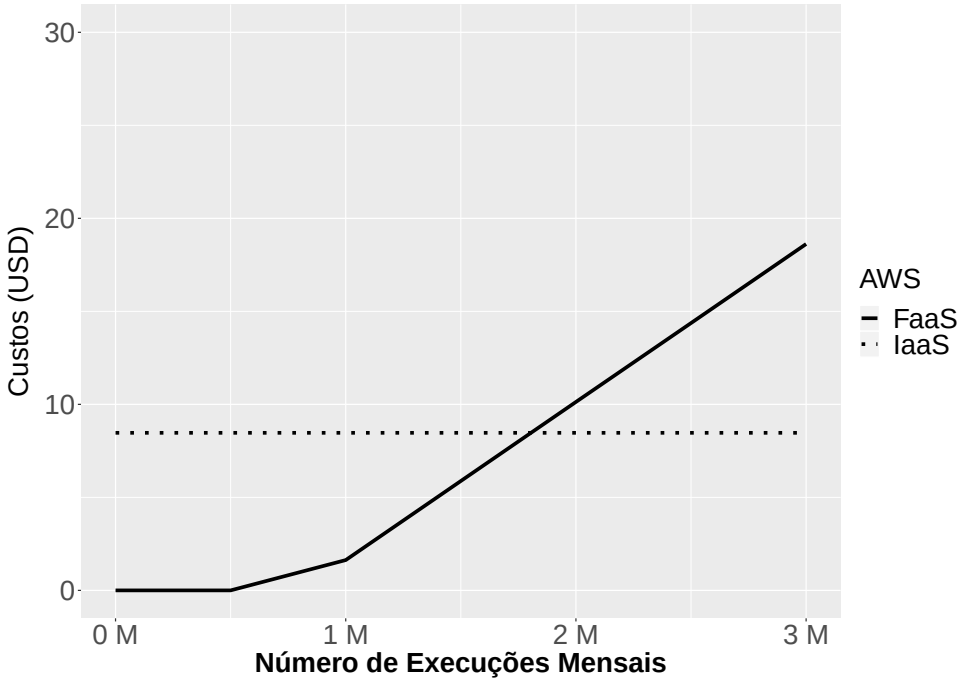
A Figura 12 mostra os custos estimados para AWS Lambda e EC2. Observa-se que o custo em FaaS varia com o número mensal de requisições, enquanto que IaaS tem custo fixo mensal de USD 8,50. O ponto de equilíbrio entre as soluções ocorre entre 1.808.700 e 1.808.800 execuções mensais, o que corresponde a aproximadamente $\lambda = 0{,}7$ req/s. Isso indica que antes desta faixa os custos de AWS Lambda serão melhores, e que após isso, utilizar EC2 t2.micro é mais vantajoso.

Para o provedor Google a comparação foi entre o Google Cloud Functions (FaaS) e o Google Compute Engine (IaaS) do Google Cloud Plataform (GOOGLE CLOUD, 2019a). A instância de MV deste último foi configurada com 1 vCPU (com ciclos de CPU a partir de 2 GHz[2]) e 1 GB de memória, obtendo um tempo médio de execução de 702 ms (menor tempo execução é 696 ms). A configuração de memória em FaaS do Google que mais se aproxima em tempo de execução do Google Compute Engine (GCE) é a de 1024 MB, que executa a função em 656 ms; o tempo de execução em FaaS é 7,0% maior que em IaaS. A Figura 13 apresenta um gráfico comparativo do custo mensal para ambos conforme o número de execuções mensais da função *benchmark*. O ponto de equilíbrio (onde os custos são iguais) está entre 2.051.400 e 2.051.500 execuções (cerca de 0,8 requisições por segundo). Portanto, abaixo desta faixa GCF apresenta custos inferiores e, após o intervalo, se torna mais caro quando comparado com Cloud Engine, que possui um custo mensal fixo de USD 19,22.

O limite máximo de execuções mensais para a instância de Cloud Engine utilizada nos testes é de 3.775.862 (conforme a Equação 4.2). Caso o cliente de IaaS necessite de quantidades

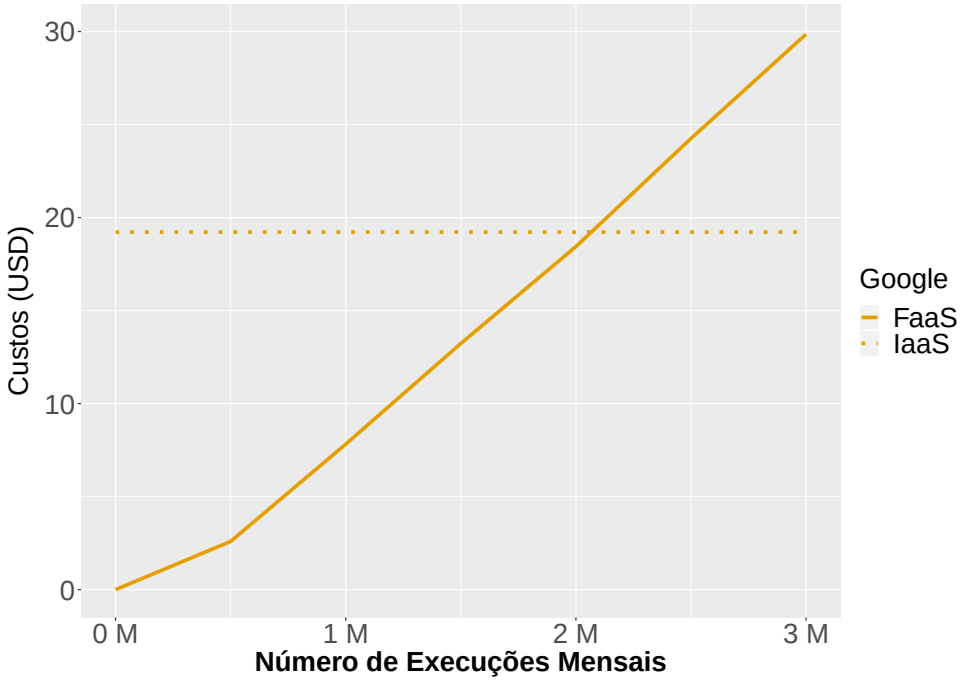
[1] <https://aws.amazon.com/pt/ec2/instance-types/>
[2] <https://cloud.google.com/compute/docs/cpu-platforms>

Figura 12 – Comparação de Custos de AWS Lambda (FaaS) alocado com 640 MB de memória e EC2 (IaaS) t2.micro de 1 vCPU e 1 GB de memória



Fonte: Elaborado pelo autor, 2019

Figura 13 – Comparação de custos de Google Cloud Functions (FaaS) com 1024 MB de memória e Google Compute Engine (IaaS) com MV de 1 vCPU e 1 GB de memória.



Fonte: Elaborado pelo autor, 2019

maiores de requisições será necessário alocar mais recursos para a MV, o que impactará no desempenho e custo. Já em GCF o limiar são as 1.000 execuções concorrentes possíveis, o que indica que neste exemplo o GCF poderia executar até 1.524 req/s (conforme Equação 4.1).

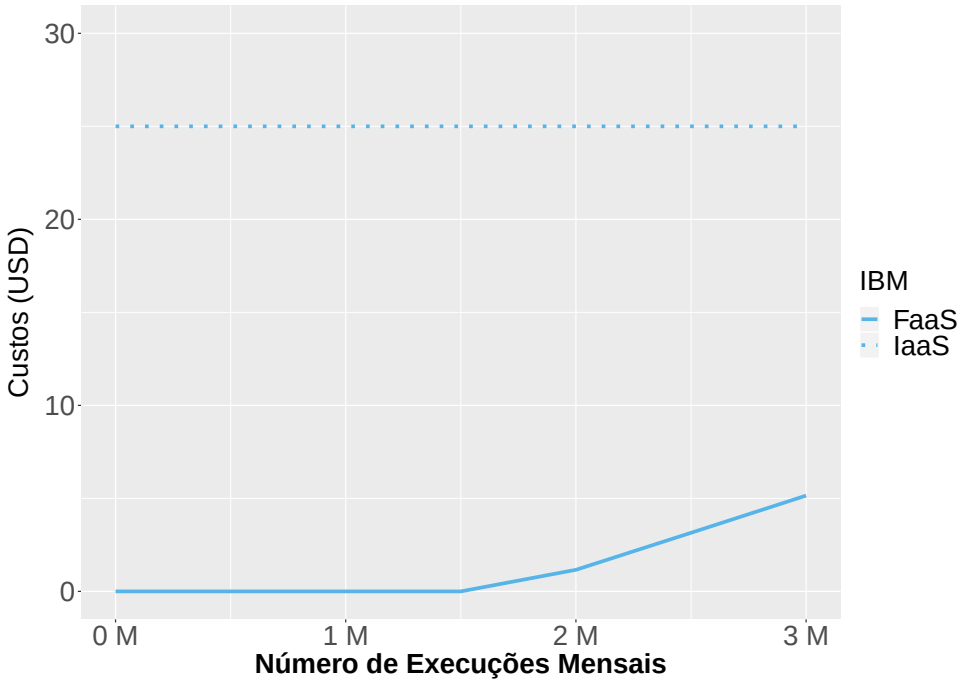
A IBM possui o Virtual Server como o seu serviço de IaaS (IBM, 2019b). Para os testes foi alocada uma instância de MV com 1 vCPU (equivalente a 2 GHz ou mais [3]) e 1 GB de memória, obtendo um tempo médio de execução de 822 ms (o menor tempo execução foi de 814 ms). O *NMEM* é de aproximadamente 3.228.501. Como IBM Cloud Functions não apresenta alterações claras de capacidade de CPU em relação à quantidade de memória alocada, não é possível selecionar uma configuração de FaaS que possua desempenho semelhante ao Virtual Server. Além disso, reconfigurar o Virtual Server para equiparar seu desempenho ao obtido com Cloud Functions também não é possível, uma vez que somente é possível adicionar ou remover vCPUs na MV. Na IBM, aumentar ou diminuir o poder de processamento sem aumentar a contagem de núcleos virtuais é possível apenas em servidores *bare metal*.

Na IBM Cloud Functions, o *benchmark* executado na linguagem Node.js com 128 MB de memória possui um tempo de execução médio de 1465 ms, portanto, seu custo mensal permanece zero, inclusive logo após o *NMEM* do Virtual Server. Ao executar o mesmo *benchmark* com 256 MB, o tempo médio de execução fica em 937 ms (13,9% maior que em IaaS). Conforme apresentado na Figura 14, o custo mensal é de USD 5,15 para 3.000.000 execuções, inferior ao custo mensal fixo de USD 25,00 do Virtual Server. Além disso, não ocorre um ponto de equilíbrio entre o serviço de FaaS com IaaS, antes deste último atingir o *NMEM*. O limiar de IBM Cloud Functions com 256 MB com suas 1000 execuções concorrentes possíveis é de 1.067 req/s. Um ponto de equilíbrio vai aparecer quando FaaS de IBM é configurado com 1024 MB de memória (tempo médio ficou em 916 ms): o custo no modelo FaaS ultrapassa o da solução IaaS entre 2.043.800 e 2.043.900 invocações (aproximadamente $\lambda = 0{,}7$ req/s).

É importante ressaltar que a análise está focada exclusivamente nos custos relacionados à execução da função *benchmark* deste trabalho. Os custos mensais de IaaS podem ser diferentes para contratações anuais do serviços, descontos com base no tempo de uso e, obviamente, conforme a configuração da MV instanciada. Para o modelo FaaS, os custos estão baseados nos tempos médios de execução, embora os provedores FaaS efetuem a cobrança baseado no tempo de cada invocação. Outros custos podem ocorrer, caso exista necessidade de persistência de dados, por exemplo. Em contrapartida, como no modelo FaaS o provedor é responsável por provisionar, configurar e gerenciar a infraestrutura, o cliente não incorre em custo de pessoal para assumir essas tarefas, o que pode representar um custo total de propriedade (TCO – *total cost of ownership*) menor do que no modelo IaaS, como discutido em (ROGERS, 2017).

[3] <https://cloud.ibm.com/docs/vpc-on-classic-vsi?topic=vpc-on-classic-vsi-creating-virtual-servers>

Figura 14 – Comparação de custos de IBM Cloud Functions (FaaS) com 256 MB de memória e IBM Virtual Servers (IaaS) com MV de 1 vCPU e 1 GB de memória.



Fonte: Elaborado pelo autor, 2019

4.4 DISCUSSÃO DOS RESULTADOS

Com base nos resultados apresentados, é possível retomar as questões de pesquisa elencadas na Seção 1.1 (página 15) e examinar as respostas encontradas para elas:

- **QP1** – *Existem variações significativas de desempenho em configurações fixas?*

  Sim. Os resultados apresentados na Seção 4.2 mostram que, para 1/3 das configurações testadas (15 de 45), o tempo de execução apresentou um coeficiente de variação superior a 20%, atingindo um máximo de 125%.

- **QP2** – *Quanto o desempenho varia entre plataformas FaaS com o mesmo tamanho de memória e linguagem de programação?*

  A Seção 4.2 mostrou que a IBM teve desempenho claramente superior às demais plataformas para funções com até 256 MB de memória. Foram observadas diferenças de desempenho de até 8,5× entre plataformas e 16,8× entre linguagens de programação. Isso sugere, por exemplo, que mudar de provedor pode trazer benefícios para o cliente. No entanto, a dependência de outros serviços do provedor (para armazenamento persistente, por exemplo) e a heterogeneidade de suporte a linguagens de programação podem dificultar a migração entre plataformas. Uma forma de minimizar esse *lock-in* de fornecedor

e facilitar a migração é usar ferramentas como o Serverless Framework, um *framework* que visa permitir a portabilidade entre provedores FaaS (SERVERLESS, 2019).

- **QP3** – *A escolha de linguagem de programação afeta perceptivelmente o desempenho?*

  Sim. Em média, Node.js foi quase duas vezes mais rápido que Go, e cerca de 13× mais rápido que Python. Além disso, as diferenças relativas entre linguagens observadas nas plataformas FaaS foram quase o dobro das mensuradas em *bare metal*. Isso sugere que as diferenças de desempenho precisam ser mensuradas nas próprias plataformas, não sendo possível confiar em dados obtidos em experimentos em laboratório.

- **QP4** – *Como as escolhas de plataforma e linguagem de programação afetam o custo?*

  A Seção 4.3 mostra que, no geral, IBM tem o menor custo até 512 MB, e AWS Lambda é o mais econômico para alocações maiores de memória. A influência da linguagem de programação sobre o custo segue a sua influência sobre o desempenho. Além disso, foram observadas diferenças de custo de até 67× entre plataformas e entre linguagens.

- **QP5** – *Nos casos em que alocar uma quantidade maior de memória resulta em um melhor desempenho, isso também gera um melhor custo-benefício?*

  Os experimentos realizados ratificam que o desempenho em AWS e GCF é proporcional à memória alocada. No entanto, os resultados da Seção 4.3.1 mostram que alocar memória adicional nem sempre se traduz em um melhor custo-benefício: isso acontece na AWS (a relação custo/desempenho decresce monotonicamente com o aumento da memória, com todas as linguagens), mas não em GCF, onde tamanhos intermediários de memória apresentam o menor custo/desempenho.

- **QP6** – *As variações de desempenho induzem variações de custo na mesma proporção?*

  Não. Na Seção 4.3.2, o maior CV de tempo de execução é de 65,6% ao passo que o maior CV de custo foi de aproximadamente 5%. Portanto, os resultados apresentados evidenciam que, embora a variabilidade do tempo de execução possa ser significativa, ela gera uma variação inexpressiva nos custos para o cliente.

- **QP7** – *Como os custos de FaaS e IaaS podem ser comparados para desempenho equivalente?*

  Os resultados da Seção 4.3.3 mostram que, para AWS e GCF, o custo em FaaS é inferior ao de IaaS até um determinado limite de invocações mensais. A comparação para IBM fica um pouco prejudicada pela impossibilidade de selecionar configurações FaaS e IaaS que ofereçam desempenho equivalente, mas ainda assim é possível afirmar que a alocação de CPU independente da quantidade de memória induz custos em FaaS menores do que os custos em IaaS.

Uma característica que chama a atenção de desenvolvedores para o modelo FaaS são as faixas de gratuidade oferecidas pelos provedores. O presente estudo mostra que essas faixas podem manter o custo próximo a zero até determinados limiares de uso. Em contrapartida, como o custo é diretamente proporcional à utilização das funções, é preciso atentar para a possibilidade de receber altas cargas de requisições, que levarão a despesas significativas. Regular o acesso a funções, seja por *rate limiting* ou mesmo evitando que elas possam ser invocadas diretamente por usuários finais (ou por ações desses usuários), pode ser uma forma de manter o custo sob controle.

4.5 CONSIDERAÇÕES DO CAPÍTULO

A utilização de uma função simples e facilmente portável possibilitou os experimentos com diferentes linguagens de programação, conforme descrito na Seção 4.1. O primeiro experimento que avaliou as variações no tempo de execução para uma mesma condição de configuração (provedor, linguagem e memória) apresenta CVs de até 125%, sendo que um terço apresentou patamares de CV superiores a 20%. Em um segundo momento as configurações foram variadas e mostraram que Node.js é a linguagem de programação com melhor desempenho em mais de 90% dos testes. As razões dos tempos médios de execução de Go/Python com Node.js sugerem valores distintos conforme o tamanho de memória e provedor, sendo que são maiores quando comparados com um *bare metal*. Na sequência uma comparação de custos sugere que AWS tem custos praticamente constantes independente da memória, IBM possui custo proporcional ao tamanho da memória alocada e tem menor custo até 512 MB e no GCF os custos são mais baixos para memórias configuradas em 256 MB e 512 MB do que as demais configurações. Todos os parâmetros de linguagem de programação, memória alocada e provedor utilizado influenciam nos custos. Entretanto, as grandes variações de desempenho não provocam grandes variações no custo, uma vez que a maior variação atingiu apenas um CV 4,92%. Por fim, a Seção 4.3.3 sugere que FaaS proporciona custos inferiores ao modelo clássico IaaS quando o número de execuções por mês é pequeno, e passa a ser mais custoso conforme aumenta.

5 CONCLUSÃO

A computação em nuvem disponibiliza, sob demanda, recursos computacionais como serviços. Em vez de efetuar investimentos fixos significativos em infraestruturas computacionais físicas, tendo de responsabilizar-se pelo seu gerenciamento, os clientes de nuvem alugam os recursos computacionais que necessitam, pagando conforme seu uso. O modelo emergente de serviço FaaS vem sendo adotado por desenvolvedores de aplicações para nuvens computacionais motivados por dois fatores preponderantes, redução das preocupações com gerenciamento e provisionamento da infraestrutura virtualizada e a possibilidade de economia de custos. Para o primeiro, o único parâmetro de configuração é a quantidade de memória que será alocada: as funções são ativadas sob demanda, em resposta a eventos, a responsabilidade de propiciar a disponibilidade e escalabilidade é do provedor de FaaS. O segundo fator é motivado pelo fato do cliente ser cobrado por invocação, pagando efetivamente apenas pelo uso, em contraste com outros modelos de serviço no qual ele pode ter de pagar por recursos provisionados mas não utilizados.

Conceitualmente a cobrança efetuada em FaaS pode parecer simples, uma vez que o cliente paga apenas pelo número de invocações, tempo de execução e recursos alocados. No entanto, a visibilidade dos custos é baixa, uma vez que depende do volume de requisições e do produto entre a quantidade de recursos consumidos e o tempo em que eles são utilizados, que pode ser variável. Diversas características das funções podem influenciar no desempenho e, consequentemente, nos custos de FaaS, como por exemplo, a linguagem de programação adotada, o provedor escolhido, a quantidade de memória RAM alocada e suas combinações. Outros fatores que prejudicam essa visibilidade do custo são a existência de faixas de gratuidade e a dependência de outros serviços do provedor de nuvem, cobrados à parte. Esses fatores não foram amplamente explorados pela literatura e foram foco deste trabalho (à exceção dos serviços de provedor). Além disso, em situações onde FaaS pode ser adotado como um modelo alternativo aos modelos clássicos de nuvem computacional, como por exemplo IaaS, é importante comparar os custos e exemplificar situações em que um modelo, em relação ao custo, é mais vantajoso que o outro. Nessa comparação ainda foram levados em consideração os limiares de execuções mensais destes dois modelos.

Neste contexto, o objetivo do trabalho foi investigar a influência que a escolha do provedor e seus parâmetros podem ocasionar no desempenho e custo, como o custo é afetado pela variabilidade do desempenho. Para isso, uma abordagem experimental foi utilizada, restringindo a medição do desempenho e do custo de funções sem considerar demais serviços do provedor, apenas considerando os aspectos inerentes de FaaS. Como em alguns provedores FaaS a alocação de memória influencia na alocação de CPU (e no desempenho), uma função *CPU-bound* foi utilizada nos experimentos para testar a hipótese de que alocar mais memória do que

o necessário pode proporcionar melhor desempenho com um mesmo custo ou até mesmo um custo menor. Por fim, essa aplicação é simples para ser portável e facilitar o teste com diferentes provedores e linguagens de programação. Assim foi possível medir o desempenho com base no provedor FaaS selecionado, memória configurada e linguagem de programação utilizada, avaliar de forma experimental os custos médios conforme as configurações especificadas e número de invocações, analisar a razão de custo/desempenho e como a variabilidade do desempenho pode influenciar na variabilidade do custo e por fim, apresentar um estudo de caso comparando o custo de uma solução FaaS com os de uma configuração IaaS com desempenho similar.

A avaliação experimental de desempenho e custo foi realizada em três plataformas FaaS: AWS Lambda, Google Cloud Functions e IBM Cloud Functions. As contribuições propostas foram alcançadas e o resultados experimentais demonstraram que: *(i)* ocorrem significativas diferenças de desempenho em todos os provedores, inclusive conforme o recurso de memória alocado, linguagem de programação adotada e, inclusive as combinações entre todos estes fatores; *(ii)* o custo também é impactado significativamente pelas configurações, em algumas situações com diferenças superiores a $67\times$; *(iii)* a relação custo/desempenho apresenta diferenças que sugerem uma evidência notável de quanto desempenho e custo variam entre as plataformas FaaS; *(iv)* a variação que o desempenho provoca no custo mensal de FaaS é pequena, contrariando o que a literatura sugeria; *(v)* ao comparar o provedor IaaS e FaaS em situação específica, o ponto de equilíbrio entre os modelos fica em torno de 2 milhões de execuções e fatores como o número máximo de execuções mensais e custo de configurações devem ser levados em consideração. Os experimentos levam a concluir que os clientes do FaaS devem usar tal metodologia e resultados experimentais como orientação e avaliar o desempenho e o custo de suas próprias funções antes efetivamente implantá-las em FaaS em larga escala.

Cabe mencionar algumas dificuldades que foram encontradas no decorrer deste trabalho. Primeiramente, a falta de um *benchmark* específico para FaaS, o que exigiu o desenvolvimento de uma ferramenta própria, além de dificultar uma comparação direta com os resultados de outras publicações. Em segundo lugar, a função utilizada neste trabalho foi escolhida de forma a facilitar a migração entre os provedores, mas houve a necessidade de alguns ajustes na forma de entrada e chamada da função e saída dos dados nas plataformas (a saída deveria ser uma resposta JSON em alguns provedores). Por fim, os *logs* com os resultados das invocações, objeto importante deste estudo, são armazenados de forma limitada e temporária pelos provedores e, portanto, tinham que ser extraídos do provedor. Para isso um *script* foi criado com a tarefa de fazer a coleta dos registros a cada hora e armazená-los localmente (isto é, fora do ambiente de nuvem). Embora os provedores ofereçam serviços de armazenamento permanente dos *logs*, estes não foram utilizados para evitar cobranças adicionais.

Este trabalho de mestrado resultou nas seguintes publicações:

- Quer Pagar Quanto? Uma Comparação de Custos de Provedores de Nuvem FaaS, publi-

cado nos Anais do X Computer on the Beach, Florianópolis, Brasil, 2019 (BORTOLINI; OBELHEIRO, 2019); e,

- Investigating Performance and Cost in Function-as-a-Service Platforms, publicado no International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Antuérpia, Bélgica, 2019 (BORTOLINI; OBELHEIRO, 2020).

Algumas perspectivas podem ser elencadas para a continuidade deste trabalho. A primeira envolveria analisar outros fatores que influenciam os custos de FaaS, como tráfego de rede e armazenamento em nuvem. A segunda seria propor um modelo analítico para guiar o desenvolvedor FaaS nas escolhas de alocação, linguagem de programação e provedor mais adequados às características da função pretendida. Uma terceira perspectiva seria incorporar outros provedores de FaaS à análise, considerando também outras modalidades de tarifação em FaaS, como planos mensais de assinatura. A quarta perspectiva seria realizar experimentos ao longo de um período mais extenso, permitindo a análise de tendências do comportamento das plataformas FaaS e a caracterização das distribuições dos tempos de execução das funções.

REFERÊNCIAS

ADZIC, G.; CHATLEY, R. Serverless computing: Economic and architectural impact. In: **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. New York, NY, USA: ACM, 2017. (ESEC/FSE 2017), p. 884–889. Disponível em: <http://doi.acm.org/10.1145/3106237.3117767>.

AWS. **Release: AWS Lambda on 2014-11-13 - AWS Release Notes**. 2014. Disponível em: <https://aws.amazon.com/pt/releasenotes/release-aws-lambda-on-2014-11-13/>. Acesso em: 30 set. 2019.

____. **Serverless Architectures with AWS Lambda**: Overview and best practices. 2017. Disponível em: <https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf>. Acesso em: 10 nov. 2019.

____. **Amazon EC2**. 2019. Disponível em: <https://aws.amazon.com/ec2/>. Acesso em: 01 dez. 2019.

____. **AWS Lambda**. 2019. Disponível em: <https://aws.amazon.com/lambda>. Acesso em: 03 dez. 2019.

BHARDWAJ, S.; JAIN, L.; JAIN, S. Cloud computing: A study of infrastructure as a service (iaas). **International Journal of engineering and information Technology**, v. 2, n. 1, p. 60–63, 2010.

BORTOLINI, D.; OBELHEIRO, R. R. Quer pagar quanto? uma comparação de custos de provedores de nuvem faas. **Anais do Computer on the Beach**, p. 123–131, 2019.

____. Investigating performance and cost in function-as-a-service platforms. In: BAROLLI, L.; HELLINCKX, P.; NATWICHAI, J. (Ed.). **Advances on P2P, Parallel, Grid, Cloud and Internet Computing**. Cham: Springer International Publishing, 2020. p. 174–185. ISBN 978-3-030-33509-0.

BUYYA, R. et al. A manifesto for future generation cloud computing: Research directions for the next decade. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 51, n. 5, p. 105:1–105:38, nov. 2018. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3241737>.

COSTELLO, K. **Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019**. [S.l.], 2019. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g>.

EIVY, A. Be wary of the economics of "serverless" cloud computing. **IEEE Cloud Computing**, v. 4, n. 2, p. 6–12, March 2017. ISSN 2325-6095.

FIELD, A.; MILES, J.; FIELD, Z. **Discovering Statistics Using R**. [S.l.]: SAGE Publications, 2012.

FIGIELA, K. et al. Performance evaluation of heterogeneous cloud functions. **Concurrency and Computation: Practice and Experience**, v. 30, n. 23, p. e4792, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4792>.

GIUSTO, D. et al. **The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications**. Springer New York, 2010. ISBN 9781441916747. Disponível em: <https://books.google.com.br/books?id=vUpiSRc0b7AC>.

GOOGLE CLOUD. **Compute Engine**. 2019. Disponível em: <https://cloud.google.com/compute/>. Acesso em: 01 dez. 2019.

____. **Google Cloud Functions**. 2019. Disponível em: <https://cloud.google.com/functions>. Acesso em: 13 nov. 2019.

____. **What you can build with Cloud Functions**. 2019. <https://cloud.google.com/functions/use-cases/>. Disponível em: <https://cloud.google.com/functions/use-cases/>. Acesso em: 14 nov. 2019.

HOROVITZ, S. et al. FaaStest – machine learning based cost and performance FaaS optimization. In: **15th GECON**. Pisa, Italy: Springer, 2018. p. 171–186. ISBN 978-3-030-13341-2.

IBM. **IBM Cloud Functions**. 2019. Disponível em: <https://cloud.ibm.com/functions>. Acesso em: 23 nov. 2019.

____. **IBM Cloud Virtual Servers**. 2019. Disponível em: <https://www.ibm.com/cloud/virtual-servers>. Acesso em: 03 dez. 2019.

IBM CLOUD. **Common use cases**. 2019. Disponível em: <https://cloud.ibm.com/docs/openwhisk?topic=cloud-functions-use_cases>. Acesso em: 14 nov. 2019.

JAIN, R. **The Art of Computer Systems Performance Analysis**. [S.l.]: Wiley, 1991.

JONAS, E. et al. Occupy the cloud: Distributed computing for the 99%. In: **Proceedings of the 2017 Symposium on Cloud Computing**. New York, NY, USA: ACM, 2017. (SoCC '17), p. 445–451. ISBN 978-1-4503-5028-0. Disponível em: <http://doi.acm.org/10.1145/3127479.3128601>.

LEE, H.; SATYAM, K.; FOX, G. C. Evaluation of production serverless computing environments. **IEEE 11th International Conference on Cloud Computing**, p. 442–450, 2018.

LEITNER, P.; CITO, J.; STÖCKLI, E. Modelling and managing deployment costs of microservice-based cloud applications. In: **2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)**. [S.l.: s.n.], 2016. p. 165–174.

LLOYD, W. et al. Serverless computing: An investigation of factors influencing microservice performance. In: IEEE. **2018 IEEE International Conference on Cloud Engineering (IC2E)**. [S.l.], 2018. p. 159–169.

MALAWSKI, M. et al. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. **Future Generation Computer Systems**, 2017. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X1730047X>.

MCGRATH, G.; BRENNER, P. R. Serverless computing: Design, implementation, and performance. In: **2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)**. Los Alamitos, CA, USA: IEEE Computer Society, 2017. p. 405–410. ISSN 2332-5666. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/ICDCSW.2017.36>.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

NEWMAN, S. **Building microservices: designing fine-grained systems**. [S.l.]: "O'Reilly Media, Inc.", 2015.

PAWLIK, M.; FIGIELA, K.; MALAWSKI, M. Performance considerations on execution of large scale workflow applications on cloud functions. **arXiv preprint arXiv:1909.03555**, 2019.

ROBERTS, M. **Serverless Architectures**. 2018. Disponível em: <https://martinfowler.com/articles/serverless.html>. Acesso em: 30 set. 2019.

ROGERS, O. **Economics of Serverless Cloud Computing**. [S.l.], 2017. Disponível em: <http://bit.ly/2mVA3XW>. Acesso em: 30 out. 2019.

SERVERLESS. **The complete solution for building & operating serverless applications**. 2019. Disponível em: <https://serverless.com/>. Acesso em: 01 dez. 2019.

VAQUERO, L. M. et al. A break in the clouds: Towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>.

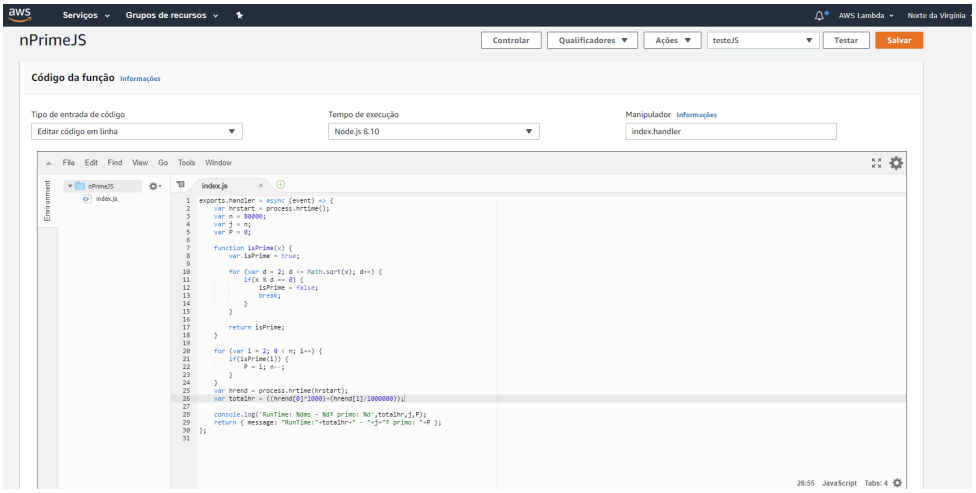
VILLAMIZAR, M. et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. **Service Oriented Computing and Applications**, v. 11, n. 2, p. 233–247, Jun 2017.

WANG, L. et al. Peeking behind the curtains of serverless platforms. In: **2018 USENIX Annual Technical Conference (USENIX ATC 18)**. Boston, MA: USENIX Association, 2018. p. 133–146. ISBN 978-1-931971-44-7. Disponível em: <https://www.usenix.org/conference/atc18/presentation/wang-liang>.

APÊNDICE A – AMBIENTE FAAS AWS LAMBDA

Este apêndice apresenta o ambiente FaaS no provedor AWS Lambda onde os experimentos foram executados, suas características e funcionalidades. A Figura 15 mostra que o usuário pode escolher a linguagem de programação, o módulo (arquivo) a ser executado, e tem à sua disposição um editor para a inclusão do código da função que está sendo criada. O provedor permite também verificar a região do provedor AWS Lambda onde a função FaaS será executada e realizar o *upload* de um diretório compactado com os códigos da função. É possível também configurar o versionamento das funções, o que permite criar uma nova função de teste e manter a versão atualmente publicada.

Figura 15 – Ambiente de inserção do código FaaS em AWS Lambda.

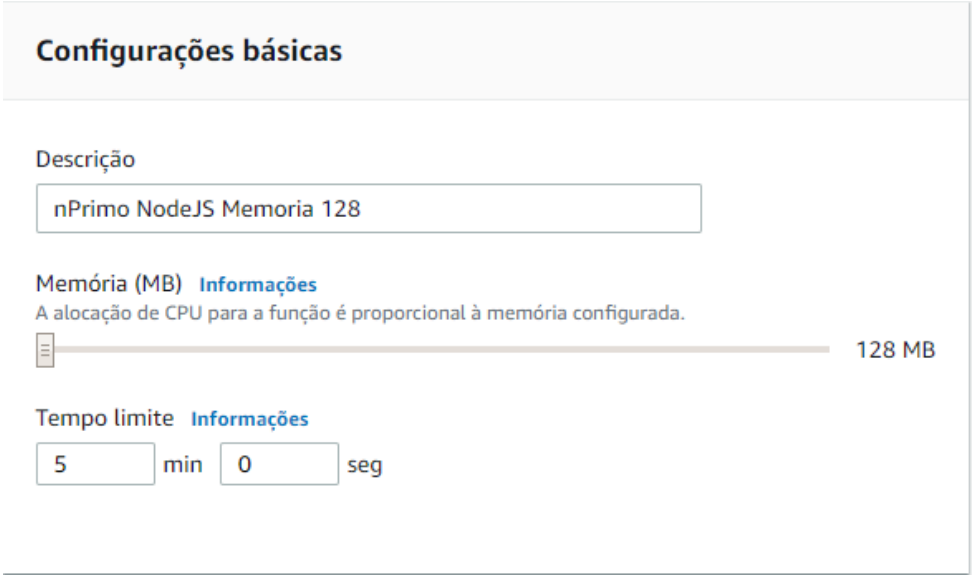


Fonte: AWS (2019b)

Após isso, existem as opções para determinar a quantidade de memória RAM que será alocada e o tempo limite de execução da função. A Figura 16 apresenta tais opções, e o desenvolvedor tem a possibilidade de escolher os valores conforme o intervalo e granularidade definidos pela plataforma. O provedor também fornece outras configurações específicas, como variáveis de ambiente, para gerar pares de chave-valor acessíveis a partir do código de função, de forma a armazenar parâmetros de configuração sem alterar partes específicas do código de função. Existe também a possibilidade de solicitar a alteração do limite de execuções concorrentes da função que a região permite por padrão.

O serviço Amazon CloudWatch é responsável pelo monitoramento, exibição de métricas e execução de eventos agendados, e interage bem com o AWS Lambda. Podem ser definidos alarmes que identificam comportamentos anormais conforme métricas definidas. É possível configurar um evento padrão de outro serviço do provedor que irá invocar a função

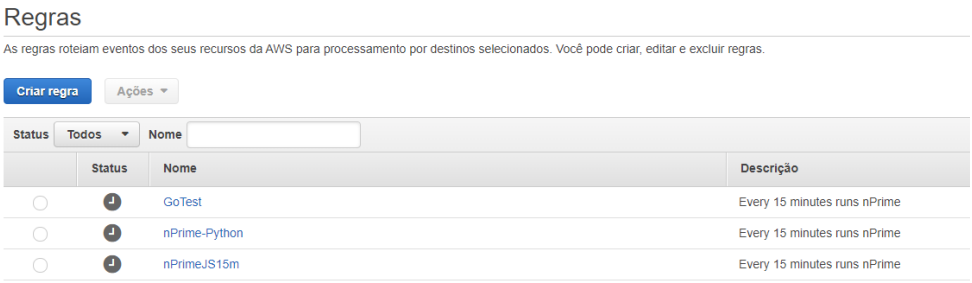
Figura 16 – Configurações básicas da função Lambda.



Fonte: AWS (2019b)

FaaS ou através de eventos programados. A Figura 17 exibe a configuração programada de invocação da função conforme os intervalos de tempo determinados.

Figura 17 – Regras de agendamento de eventos na AWS.

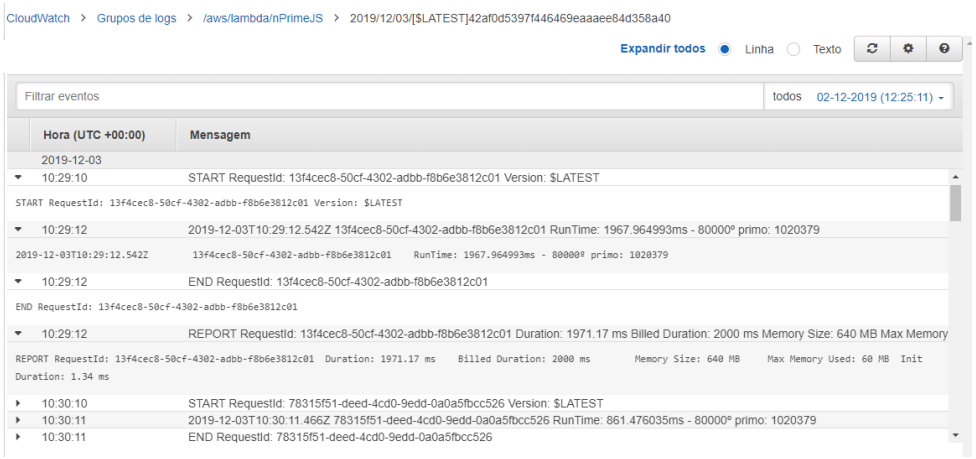


Fonte: AWS (2019b)

Na Figura 18 é possível verificar a forma como os *logs* são exibidos e disponibilizados aos usuários. Os *logs* são agrupados conforme a função, e entre as informações estão a data e hora que a função foi executada, o identificador (ID) da execução, a versão da função executada, o resultado da execução, o tempo de execução efetivo e o tempo de execução cobrado (arrendondado para cima em múltiplos de 100 ms), a memória alocada e a memória máxima usada. O usuário pode aplicar filtros para busca de *logs* por intervalos ou caracteres específicos. Os registros foram extraídos através de um *script* que executava um módulo de interface de linha de comando da AWS que buscava tais registros e os armazenava na máquina local, após isso

eram tratados e os seus dados extraídos para compor os valores deste estudo de caso.

Figura 18 – Logs das invocações das funções Lambda.

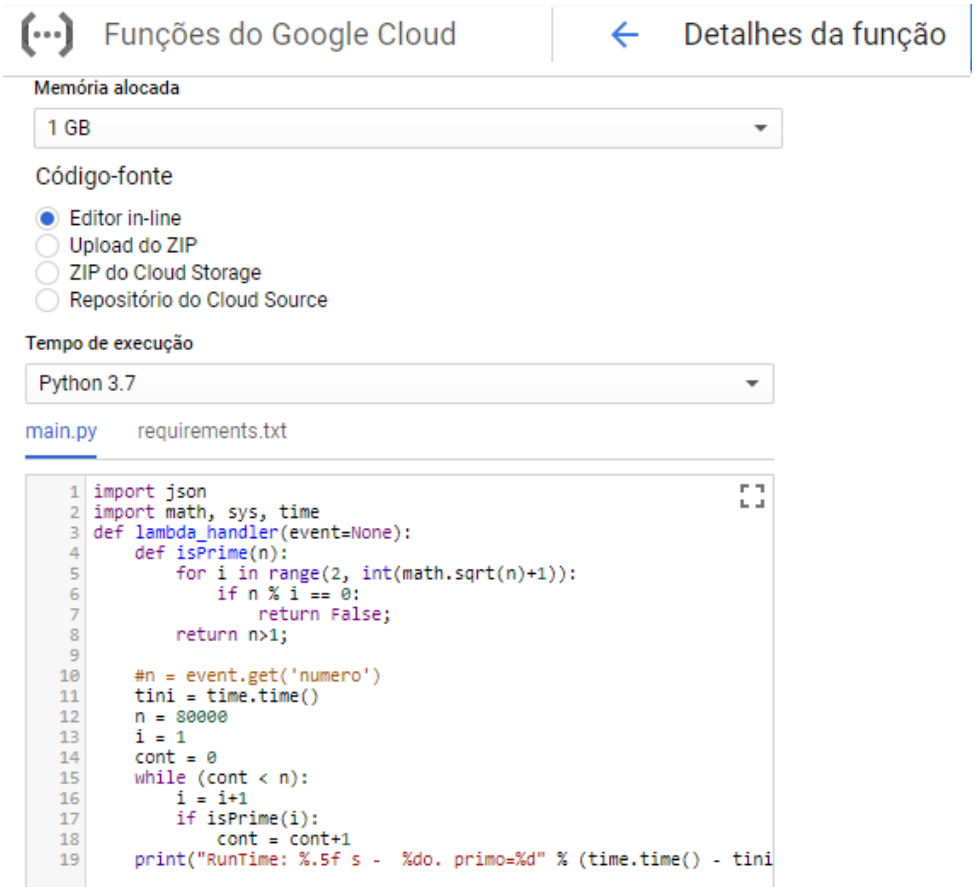


Fonte: AWS (2019b)

APÊNDICE B – AMBIENTE FAAS GOOGLE CLOUD FUNCTIONS

Este apêndice retrata as funcionalidades, características e interface do provedor Google Cloud Functions (GCF), bem como detalha os procedimentos para criação do ambiente de experimento deste trabalho. A Figura 19 exibe o ambiente de configuração de uma função no GCF, sendo possível escolher o tamanho de memória RAM que será alocada, forma de inclusão do código fonte, linguagem de programação e demais configurações.

Figura 19 – Ambiente de inserção do código FaaS e configurações da função no GCF.

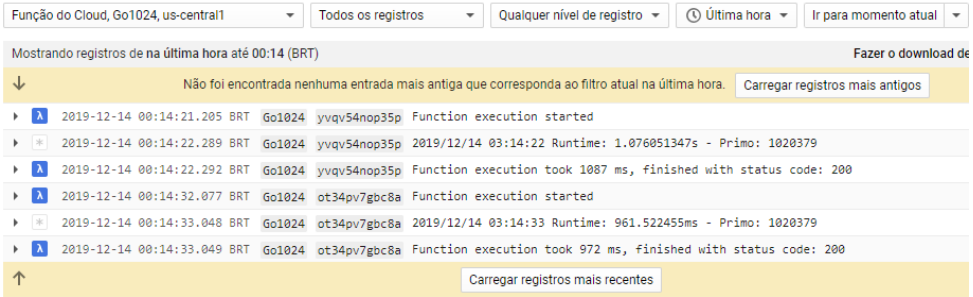


Fonte: GOOGLE CLOUD (2019b)

Na Figura 20 é possível verificar os registros que são oferecidos pelo visualizador de *logs* do GCF. As informações contidas são a data e a hora de execução da função, a função que originou tal registro, o identificador (ID) da execução e informações de execução da função, como por exemplo, quando ela foi invocada, tempo de execução e quando foi finalizada. É possível buscar *logs* usando filtros e efetuar o *download* desses *logs* diretamente pelo navegador. Os registros foram extraídos manualmente, e um *script* foi utilizado para tratar os dados.

Por fim, a Figura 21 apresenta o Cloud Scheduler do Google para invocar periodica-

Figura 20 – Logs das invocações das funções em GCF.



Fonte: GOOGLE CLOUD (2019b)

mente a função utilizada nos experimentos deste trabalho. É possível configurar a frequência de invocação, a função a ser executada, e acessar os *logs* de execuções anteriores.

Figura 21 – Agendamento de eventos no GCF.

Fonte: GOOGLE CLOUD (2019b)

APÊNDICE C – AMBIENTE FAAS IBM CLOUD FUNCTIONS

Este apêndice apresenta o ambiente da IBM Cloud Functions, terceiro e último provedor FaaS estudado por este trabalho experimental. Todas as configurações ficam praticamente em um mesmo painel, na Figura 22 é possível verificar o editor de código oferecido pela plataforma.

Figura 22 – Ambiente de inserção do código FaaS em IBM Cloud Functions.

Fonte: IBM (2019a)

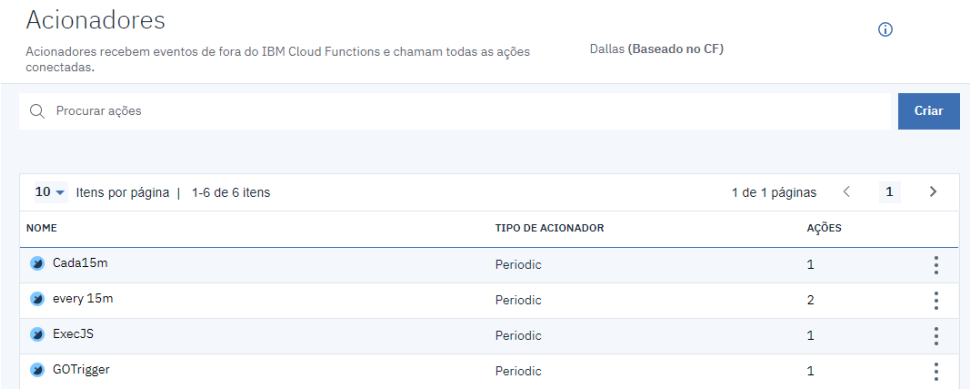
Os parâmetros básicos de configuração podem ser visualizados na Figura 23. É possível selecionar a linguagem de programação utilizada, o tempo limite para execução da função, e a memória alocada.

Figura 23 – Configurações básicas da função em IBM Cloud Functions.

Fonte: IBM (2019a)

A invocação da função, assim como nas demais plataformas, também pode ser agendado por meio de um acionador, conforme apresentado na Figura 24.

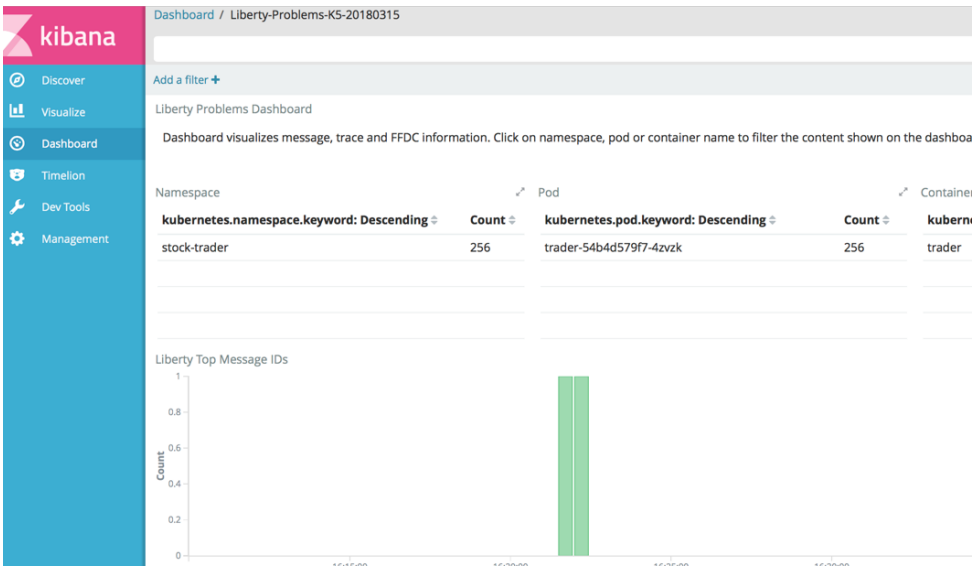
Figura 24 – Agendamento de eventos na IBM.



Fonte: IBM (2019a)

Durante o processo de coleta de resultados, a IBM iniciou o processo de descontinuação do uso do software de visualização de dados Kibana[1], porém os registros permaneceram sendo atualizados, e os dados ficaram disponíveis ao longo do trabalho. A Figura 25 exibe uma imagem da tela do Kibana na IBM Cloud Functions.

Figura 25 – *Logs* das invocações das funções em IBM Cloud Functions.



Fonte: IBM (2019a)

[1] <https://www.elastic.co/pt/products/kibana>