

Serverless-simulator - Simulador de computação sem servidores

Serveless-simulator - Serverless computing simulator

Igor Andrade Brito¹, Renata Spolon Lobato¹, Aleardo Manacero Jr.¹, Roberta Spolon²

¹Departamento. de Ciências de Computação e Estatística - Instituto de Biociências, Letras e Ciências Exatas, São José do Rio Preto - SP

²Departamento de Computação - Faculdade de Ciências, Bauru - SP
Universidade Estadual Paulista “Júlio de Mesquita Filho” – UNESP Brasil
{igor.brito, renata.spolon, aleardo.manacero, roberta.spolon}@unesp.br

Resumo — A computação sem servidores (serverless) é uma arquitetura de computação em nuvem que ganhou crescente interesse devido à sua capacidade de escalabilidade automática, pagamento baseado no uso e execução de aplicativos na nuvem, eliminando a necessidade de gerenciar infraestrutura de servidores. Este artigo apresenta o desenvolvimento de um simulador para esse tipo de computação, o serverless-simulator foi projetado para reproduzir com precisão o comportamento de plataformas serverless em uma variedade de cenários operacionais, permitindo análises detalhadas do desempenho.

Palavras Chave - computação em nuvem, computação sem servidores, simulador, simulador de computação sem servidores.

Abstract — Serverless computing is a cloud computing architecture that has gained increasing interest due to its ability for automatic scalability, pay-per-use pricing model, and execution of applications in the cloud without the need to manage server infrastructure. This article presents the development of a simulator for this type of computing. The serverless-simulator was designed to accurately reproduce the behavior of serverless platforms across a variety of operational scenarios, enabling detailed performance analyses..

Keywords - cloud computing, serverless computing, simulator, serverless simulator.

I. INTRODUÇÃO

A sociedade atualmente está completamente conectada, o uso da Internet e seus diversos serviços diferentes só cresce, e por conta disso existe o interesse constante na melhoria e aperfeiçoamento desses serviços, seja em relação a facilidade de desenvolvimento, gerenciamento e implantação, redução de custos, aprimoramento do desempenho e maior segurança.

Por conta disso, surgiu um novo modelo de computação baseado na computação em nuvem, chamado de *serverless* (sem servidores) [1]. Este método está disponível nas principais provedoras de computação em nuvem e é utilizado por diversas empresas em seus negócios e também para fins de pesquisa ou industriais.

A análise do desempenho e comportamento de sistemas computacionais é muito importante pois pode ajudar na sua melhoria de diversos modos diferentes, como reduzir custos,

melhorar performance e eficiência, além de permitir um exame detalhado sem a necessidade de efetivamente possuir o sistema. Portanto, a motivação para a realização deste estudo é aprofundar sobre sistemas *serverless* e para isso precisamos de ferramentas capazes de simular o comportamento desse tipo de sistema. A intenção é que traga contribuições para os usuários de forma que seja uma ferramenta com uma interface gráfica intuitiva para que seja fácil para qualquer usuário utilizar o sistema sem problemas.

Deste modo, este trabalho tem como objetivo propor um modelo para simulação de computação sem servidores.

As seções subsequentes deste artigo estão organizadas da seguinte forma: na Seção 2 é feito uma breve fundamentação teórica sobre computação sem servidores e simuladores. Na seção 3 são apresentados os trabalhos relacionados ao assunto proposto. Já na seção 4 está descrito o funcionamento do simulador desenvolvido. A seção 5 contém os testes realizados juntamente com a análise de resultados. A seção 6 encerra esse artigo com a conclusão

II. FUNDAMENTAÇÃO TEÓRICA

A. Computação sem servidores

Serverless computing ou computação sem servidores é uma plataforma que fornece desenvolvimento e implantação eficiente de aplicações sem a necessidade de gerenciar as camadas de infraestrutura [2], facilitando e permitindo que os desenvolvedores tenham um foco maior na lógica do negócio sem se preocupar com escalonamento e providenciamento de recursos e infraestrutura pois tecnicamente a aplicação está executando em um servidor externo garantido pela provedora. Nesse modelo, a provedora aloca e gerencia os servidores e os códigos são executados em containers gerados a partir de gatilhos definidos. Esse modelo de computação foi introduzido inicialmente pela Amazon Lambda em 2014 e após isso outras empresas como Google e Microsoft a adotaram em 2016.

Na visão de [3], para um serviço ser considerado *serverless* ele deve escalar automaticamente sem a necessidade de provisionamento explícito e ter o pagamento baseado em seu uso. Em qualquer plataforma de computação sem servidores o usuário deve somente escrever uma função em alguma lin-

guagem de alto nível, selecionar o evento que a inicializa, como carregar uma imagem no storage e deixar o sistema *serverless* lidar com todo o resto: seleção da instância, escalonamento, implantação, tolerância a falhas, monitoramento, configurações de segurança, etc [3]. O modelo permite aos desenvolvedores dissolver grandes aplicações em pequenas funções possibilitando os componentes escalar individualmente [4], e pode ser dividido em 2 categorias: *Backend-as-Service* (BaaS) e *Function-as-Service* (FaaS) [2]. Neste artigo só vamos tratar sobre a *Function-as-Service*, que é a implementação mais conhecida e importante da computação sem servidores, também conhecida por *serverless functions*. Com esse modelo os desenvolvedores precisam somente implementar o código de uma função de pequena duração e definir os gatilhos (*trigger*) para executá-la [2]. Essas funções tipicamente descrevem pequenas partes de uma aplicação completa e seu tempo de execução normalmente é limitado. Elas não ficam constantemente ativas e as plataformas de FaaS permitem que seja definido um evento para que as funções sejam iniciadas, esse evento pode ser uma requisição do cliente, serviços externos, algum fluxo de dados ou outros. Portanto a provedora é responsável por precificar, executar as funções de acordo com a demanda em instâncias isoladas e escalar suas execuções horizontalmente de acordo com o número de eventos recebidos.

Este trabalho tem como foco a categoria *Function-as-Service*. Ela é uma nova abordagem para a computação em nuvem que permite que os desenvolvedores executem funções orientadas a eventos na nuvem sem a necessidade de gerenciar e alocar os recursos ou configurar o ambiente de execução. Desse modo os programadores podem focar inteiramente no código da aplicação, e a provedora é responsável por gerenciar as dependências, compilar o código, configurar os sistemas operacionais e gerenciar a alocação dos recursos, além de carregar novas instâncias se a demanda aumentar muito e finalizar as instâncias terminadas [5].

As *Function-as-Service* também são caracterizadas pelos seus estados das instâncias das funções, são 3 estados definidos para as instâncias [6]: inicializando, executando, em espera (*initializing*, *running*, e *idle*). O estado inicializando acontece quando a infraestrutura está começando novas instâncias, o que pode incluir a configuração de novas máquinas virtuais ou containers para lidar com a carga de trabalho excessiva e também a inicialização da aplicação que são as tarefas iniciais realizadas pela instância como importar bibliotecas ou criar uma conexão com o banco de dados. A instância vai permanecer em estado de inicialização até ser capaz de responder as chamadas que chegam, e enquanto esse estado não é finalizado a instância não é capaz de aceitar nenhuma requisição. É importante ressaltar que a parte de inicialização da aplicação é tarifada pela maioria das provedoras deste tipo de serviço, enquanto o resto da parte de inicialização não é. Quando uma requisição é recebida pela instância ela passa para o estado executando, é nesse estado que ela é analisada e processada. O tempo nesse estado também é tarifado pelas provedoras. Depois do processamento terminar, a plataforma mantém as instâncias ativas (*warms*) e ociosas por um determinado período de tempo para que ela consiga lidar com a carga de trabalho que possam chegar. Esse é o chamado estado de espera, e ele não é tarifado.

Além disso, as instâncias podem ter dois tipos de início: o frio e o quente. em [6], quando uma requisição tem um início frio (*cold start*) significa que ela passou por todo o processo de lançamento e inicialização de uma nova instância. No caso de a plataforma possuir alguma instância em espera quando a requisição chegar, ela vai ser direcionada para que a instância seja reutilizada. Esse direcionamento para que uma nova instância não seja iniciada se chama início quente (*warm start*). Os inícios frios podem ser muito mais longos que os inícios quentes para algumas aplicações, portanto um número alto de inícios frios podem impactar na capacidade de resposta da aplicação e a experiência do usuário [7]. Muitas das pesquisas feitas na área de computação *serverless* focam em aperfeiçoar o início frio.

Apesar do nome, a computação sem servidores pode ser considerada um oxímoro - porque servidores ainda são usados para fazer a computação. O nome sugere que o usuário do serviço simplesmente escreve o código e deixa o servidor responsável por alocar e administrar as tarefas em nuvem [3].

B. Simuladores

Os simuladores podem ser usados para avaliar o desempenho de sistemas sem a necessidade de realizar experimentações com o sistema real.

O uso de simuladores se faz necessário pois o benchmarking em sistemas de computação em nuvem é inviável por conta do seu alto custo para ser realizado. Além disso, seu uso fornece insumos para tomadas de decisão nas empresas e ainda resulta em menores custos de implementação.

III. TRABALHOS RELACIONADOS

Existe um número relativamente pequeno de simuladores de computação sem servidores disponíveis na literatura. Em [4] foi feita uma plataforma para computação sem servidores com foco em performance. Foi implementado usando .NET e implantado com Microsoft Azure usando o Azure Storage como ferramenta para persistir os dados. O protótipo desenvolvido consiste de dois componentes: um serviço web que disponibiliza uma REST API e um serviço de trabalho que gerencia e executa os containers das funções. A camada web descobre os trabalhadores através de uma camada de mensagem que consiste de várias filas de Azure Storage.

Já no trabalho de [8] foi feito um simulador com os conceitos de *Simulation-as-Service* (SimaaS) que tem como premissa a utilização de filas de mensagens para uma organização assíncrona das execuções das funções e que consiga lidar com altas cargas de mensagens. Ele é totalmente compatível com múltiplas plataformas de computação em nuvem e pode ser executado em uma combinação de provedoras e com suporte para microserviços e componentes *serverless*. Para avaliação, os autores utilizaram dados reais de uma empresa de investimentos e fizeram uma comparação de custos com 3 diferentes máquinas e configurações: uma utilizando a plataforma *serverless* desenvolvida, e as outras duas usando o modelo de computação em nuvem tradicional (VMs), uma utilizando um servidor single core e outro utilizando um servidor mais potente com 128 cores. A plataforma *serverless* utilizada foi o AWS.

No trabalho de [6] foi desenvolvido um simulador de performance público para computação *serverless* com alto grau

O primeiro passo para começar o desenvolvimento foi determinar os requisitos que nosso sistema deveria ter. Ficou definido que o nome do sistema será *serverless-simulator* e que é necessário possuir:

IV. SERVERLESS-SIMULATOR

O primeiro passo para começar o desenvolvimento foi determinar os requisitos que nosso sistema deveria ter. Ficou definido que o nome do sistema será *serverless-simulator* e que é necessário possuir:

- interface gráfica para facilitar a interação do usuário com o sistema e permitir que os usuários leigos, que não conhecem linguagens de programação ou possuem dificuldades ao trabalhar com essas, por não serem especialistas da área de computação, possam desenvolver simuladores para avaliar o desempenho de seus trabalhos;
- utilizar o escalonamento por requisição como técnica de escalonamento e simular um processo que não tenha nenhum tipo de fila;
- possuir uma análise de resultados com informações sobre a execução do simulador como a quantidade total de máquinas que rodaram, quantas eram quente e quantas eram fria e seus respectivos tempos de execução, quantidade de vezes que as máquinas ficaram paradas e sua duração nesse estado;
- ter o código aberto e disponível para que qualquer pessoa interessada possa examinar e entender melhor a aplicação.
- Foi decidido fazer uma aplicação web pela facilidade de desenvolvimento e de acesso, sendo seu uso possível em qualquer plataforma que permita usar um navegador com conexão à internet. O modelo do sistema é o cliente-servidor no qual o cliente (tela) foi desenvolvido usando o framework Angular, que é baseado em Typescript/Javascript. Já para o servidor, que é a nossa aplicação principal, foi escolhido o framework Spring baseado em Java.
- Uma simulação no sistema é caracterizada pelos seguintes campos:
- Taxa de chegada: é a quantidade de requisições que o sistema vai receber. Medido em requisições por segundo;
- Tempo da resposta quente: representa o tempo médio

que o sistema leva para responder uma chegada quente (*warm start*). Medido em segundos;

- Tempo da resposta fria: representa o tempo médio que o sistema leva para responder uma chegada fria (*cold start*), considerando o tempo de inicialização que uma instância leva para ficar pronta para executar a requisição. Tem que ser maior que o tempo da resposta quente. Medido em segundos;
- Tempo limite de expiração: é o tempo que uma instância fica em espera (*idle*) aguardando a chegada de uma nova requisição. Se esse tempo se esgota sem que uma requisição chegue a instância é finalizada. Medido em segundos;
- Tempo da simulação: é o tempo total da simulação. Medido em segundos;

Na figura 1 é apresentado o fluxograma da execução de uma simulação, e mostra o passo a passo do sistema quando é iniciada uma simulação.

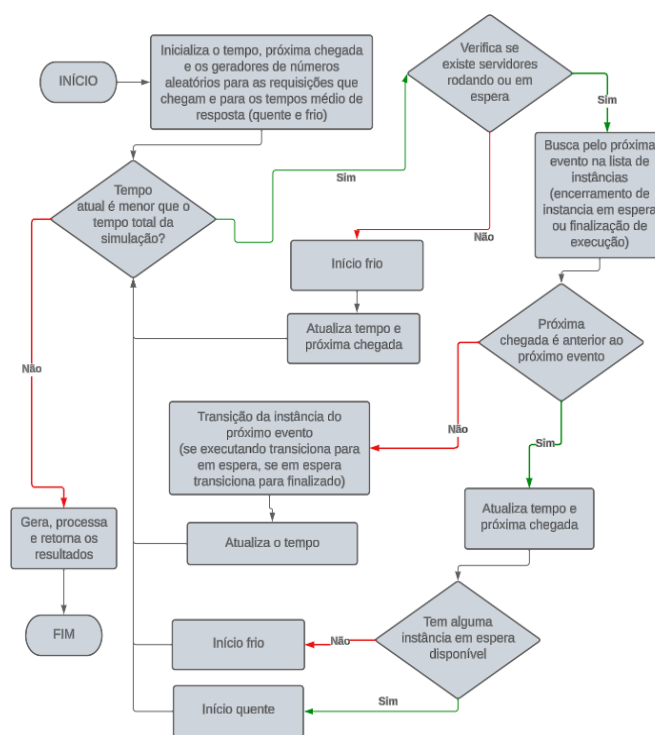


Figure 1. Fluxograma da execução da simulação

O fluxo começa com a inicialização dos geradores de números aleatórios que representam o tempo de chegada das requisições e os tempos de execução para os inícios frios e quentes, esses valores são calculados pela taxa de chegada, o tempo médio de resposta fria e o tempo médio de resposta quente, todos informados pelo usuário. Essa geração utiliza a distribuição de probabilidade exponencial para simular a chegada de clientes no sistema. As distribuições de probabilidade contém várias informações sobre um certo evento e podem

ser utilizadas para analisar desempenho de simulações. A distribuição exponencial é utilizada para modelar diversos eventos que acontecem na natureza, ela não tem uma memória sobre todos os eventos, portanto a ocorrência de um evento não está relacionado com o eventos passados. Além da inicialização dos geradores, no início da simulação também são inicializadas as variáveis que controlam o tempo e a próxima chegada.

Após essa etapa, é iniciada a repetição principal da simulação que só vai acabar quando o tempo de simulação chegar no tempo total (informado pelo usuário), o primeiro passo dentro dessa repetição é verificar se existe algum servidor executando ou em espera. Se não existir nenhum, um novo início frio é executado e o tempo e próxima chegada são atualizados. Se algum servidor estiver executando ou em espera é feito uma busca pelo próximo evento na lista de instâncias, esse próximo evento pode ser um encerramento de um instância em espera ou a finalização da execução. Em seguida é verificado o menor valor entre o próximo evento e a próxima chegada de requisição, se o menor valor for o do próximo evento é realizado a transição da instância para em espera se a instância estiver executando ou para finalizado se ela estiver em espera, em seguida o tempo é atualizado e a execução volta para o início da repetição. Agora se o menor valor for a chegada de requisição a primeira coisa a ser feita é atualizar o tempo e a próxima chegada e depois é verificado se existe alguma instância em espera disponível, se sim um início quente é executado e se não o executado é o início frio. Após isso a execução volta para o início da repetição.

Quando o tempo da execução atingir o tempo total da simulação informado pelo cliente a repetição e simulação são encerradas e então é realizado uma análise e processamento da simulação e por fim é gerado o relatório com os resultados finais.

Esse relatório gerado inclui a quantidade total de requisições feitas, quantidade de requisições frias e sua porcentagem, quantidade de requisições quentes e sua porcentagem, quantidade de requisições rejeitadas e sua porcentagem, o tempo médio de um vida das instâncias, a quantidade média de instâncias totais e a média de instância executando e em espera. Com essas informações o usuário do sistema consegue prever e analisar custos baseado em diferentes configurações do sistema.

V. TESTES E RESULTADOS

Uma série de testes foi conduzida para avaliar o desempenho do simulador, abrangendo diversos cenários e cargas de trabalho. Os resultados obtidos foram então comparados com as expectativas estabelecidas.

A. Cenário 1

O primeiro cenário de teste consiste em realizar uma mesma simulação 10 vezes e analisar as diferenças entre elas. Os valores utilizados são fixos e foram os mesmo utilizados no Sim-Fass e estão descritos na tabela 1:

TABLE I. VALORES UTILIZADOS PARA TESTES DO CENÁRIO 1

Parâmetro	Valor
Taxa de chegada	0.9 req/s
Tempo do serviço quente	1.991 seg
Tempo do serviço frio	2.244 seg

Parâmetro	Valor
Limite de expiração	10 min
Tempo de simulação	10 ⁶ seg

Ao rodar a simulação com os mesmos valores várias vezes, a expectativa era de que os valores obtidos não seriam excessivamente discrepantes, embora não necessariamente idênticos. Essa expectativa se fundamenta no fato de que o algoritmo subjacente ao nosso teste emprega uma distribuição exponencial para gerar valores aleatórios. A distribuição exponencial é caracterizada por sua tendência a produzir valores que se concentram em torno de uma média, mas com uma probabilidade razoável de desvio. Portanto, esperávamos observar uma certa variação nos resultados, refletindo a natureza estocástica do processo, enquanto ainda mantínhamos uma coerência geral. Na tabela 2 a seguir temos alguns atributos do resultado, sua média, desvio padrão e intervalo de segurança. (Valores usados para cálculo foram arredondados na quarta casa decimal).

TABLE II. MÉDIA, DESVIO PADRÃO E INTERVALO DE CONFIANÇA DE ATRIBUTOS DA RESPOSTA DA SIMULAÇÃO

Atributo	Média	Desvio padrão	Intervalo de confiança
Probabilidade início frio (%)	0.07705	0.00163049072	[0.076, 0.0781]
Probabilidade rejeitar (%)	0	0	[0, 0]
Média de vida da instância (segundos)	2084.53714	40.3617304	[2060, 2110]
Média de instâncias (quantidade)	4.44768	0.0100116732	[4.44, 4.45]
Média de instâncias executando (quantidade)	0.45136	0.000711617875	[0.451, 0.452]
Média de instâncias em espera (quantidade)	3.99631	0.00954530775	[3.99, 4]

Além disso, é importante ressaltar que o desvio padrão associado aos valores gerados pelo algoritmo é relativamente pequeno. O desvio padrão é uma medida estatística que indica a dispersão dos valores em torno da média. Neste contexto, um desvio padrão reduzido sugere que a variação entre os resultados esperados é limitada, o que aumenta nossa confiança na consistência do modelo. Assim, embora possamos esperar certa variabilidade nos resultados devido à natureza da distribuição exponencial, a magnitude dessa variação é mitigada pela estreita dispersão dos valores em relação à média, fornecendo uma base sólida para nossa análise e interpretação dos dados resultantes.

B. Cenário 2

Projetamos para o segundo cenário de teste observar a relação entre a taxa de chegada de requisições e a probabilidade de ocorrência de inícios frios. Nesse teste foram realizadas simulações com o valor da taxa de chegada variando entre 0.001 e 10 requisições por segundo e com o tempo da simulação variando de 1000 até 1000000 segundos.

TABLE III. VALORES UTILIZADOS PARA TESTES DO CENÁRIO 2

Parâmetro	Valor
Taxa de chegada	0,9 req/s
Tempo do serviço quente	1,991 seg
Tempo do serviço frio	2,244 seg
Limite de expiração	10 min
Tempo de simulação	10 ⁶ seg

Esperávamos que, conforme a taxa de chegada aumentasse, a probabilidade de início frio diminuísse. Isso se deve ao fato de que uma maior taxa de chegada implica em uma maior atividade do sistema, reduzindo assim o tempo de inatividade entre as requisições e diminuindo as chances de um início frio. Além disso, consideramos também o tempo de simulação como um fator influente. Era esperado que, à medida que o tempo de simulação diminuísse, a probabilidade de início frio aumentasse. Isso se deve à natureza do início frio, onde o sistema tem menos tempo para acumular recursos e preparar-se para o processamento de requisições. Na figura 2 podemos ver um gráfico no qual as linhas representam o tempo de simulação, o eixo x é a taxa de chegada (em req/s) e o eixo y é a probabilidade de início frio (em porcentagem).

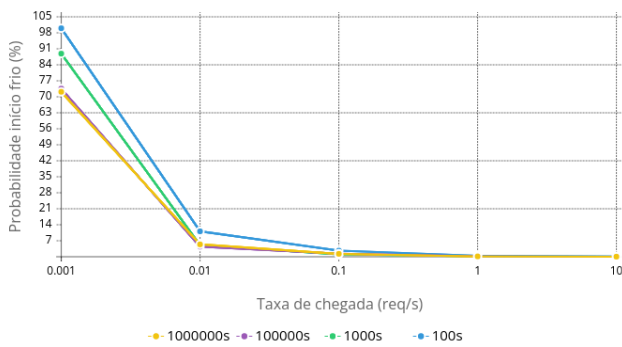


Figure 2. Gráfico com a taxa de chegada, sua probabilidade de início frio em diferentes tempos de simulação

C. Cenário 3

O próximo cenário de teste proposto envolve alterar o tempo limite de expiração entre 10 segundos e 30 minutos e verificar como o sistema se comporta. A tabela 4 contém os valores utilizados nos parâmetros para o caso de teste.

TABLE IV. VALORES UTILIZADOS PARA TESTES DO CENÁRIO 3

Parâmetro	Valor
Taxa de chegada	0.9 req/s
Tempo do serviço quente	1.991 seg
Tempo do serviço frio	2.244 seg
Limite de expiração	10 min
Tempo de simulação	10 ⁶ seg

Nossa expectativa é que, à medida que o tempo limite de expiração diminua, aumente a probabilidade de ocorrência de

inícios frios. Isso se deve ao fato de que um tempo limite mais curto resulta em uma janela de oportunidade mais estreita para reutilizar instâncias previamente inicializadas, levando potencialmente a um aumento na inicialização de novas instâncias. Além disso, também é esperado que a redução do tempo limite também esteja correlacionada com uma diminuição na quantidade média de instâncias em espera, já que a alta rotatividade de instâncias resultante de tempos de expiração mais curtos pode reduzir o congestionamento do sistema e melhorar a eficiência global de utilização de recursos. A figura 3 mostra a probabilidade de início frio de acordo com o tempo de simulação, e a figura 4 apresenta a quantidade média de instâncias em espera de acordo com o tempo da simulação.

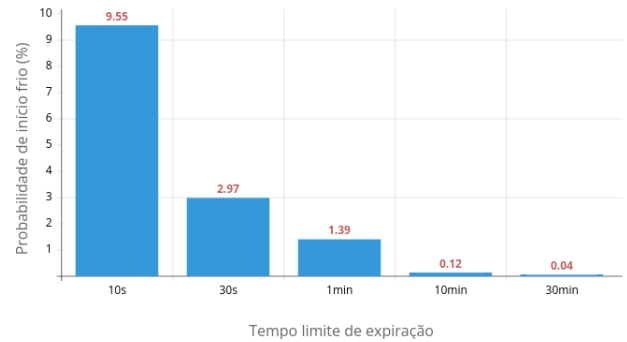


Figure 3. Gráfico com a taxa de chegada, sua probabilidade de início frio em diferentes tempos de simulação

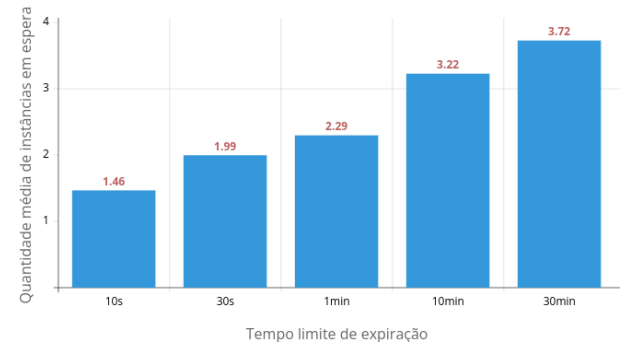


Figure 4. Gráfico com a taxa de chegada, sua probabilidade de início frio em diferentes tempos de simulação

D. Análise de resultados

Nosso objetivo principal foi avaliar o desempenho do simulador e validar se ele está produzindo resultados que se enquadram com o esperado; e os resultados dos três cenários de teste foram consistentemente satisfatórios e estiveram dentro das expectativas estabelecidas.

As ideias iniciais foram cumpridas e as métricas de desempenho, como tempo de resposta, probabilidades dos inícios frio e quantidade média de instâncias, mostraram-se dentro dos limites aceitáveis em todos os cenários testados, portanto, os resultados obtidos nesta fase da pesquisa fornecem uma validação sólida do sistema proposto.

Os resultados dos cenários de teste demonstraram que o simulador de plataforma *serverless* desenvolvido é uma ferramenta eficaz e precisa para análise e simulação de sistemas *serverless*. Os resultados obtidos fornecem uma base sólida para futuras investigações e desenvolvimentos nesta área, e confirmam a utilidade e a confiabilidade do simulador como uma ferramenta de pesquisa e desenvolvimento, podendo contribuir com avanços significativos no campo da computação em nuvem e da computação sem servidores.

VI. CONCLUSÕES

Neste trabalho foi apresentado um pouco sobre computação sem servidores, além do processo de simulação desse tipo de serviço. Então foi introduzido o *serverless-simulator*, um simulador para computação *serverless* que tem como objetivo prever o comportamento de um sistema de acordo com os valores informados.

Com o *serverless-simulator* é possível prever a infraestrutura necessária para aplicação, a qualidade do serviço e seu comportamento, permite também analisar o desempenho do seu sistema com diferentes parâmetros e possibilita que o usuário otimize melhor seus recursos. Tudo isso com uma interface gráfica simples e fácil de ser utilizada.

Para trabalhos futuros pensamos na possibilidade de implementar uma previsão de custos de plataformas *serverless*, como por exemplo a AWS Lambda, Google Cloud Functions e Azure Cloud Functions. Também pensamos em realizar um teste de usabilidade com especialistas da área para medir o quão útil o sistema é. Além disso, temos como objetivo disponibilizar o sistema no internet para acesso de todos, manter o *serverless-simulator* atualizado e adicionar novos recursos oferecidos pelas plataformas.

AGRADECIMENTOS

Os autores expressam seus agradecimentos para a UNESP (Universidade Estadual Paulista “Júlio de Mesquita Filho”), FAPESP (“Fundação de Amparo à Pesquisa do Estado de São Paulo”), CAPES (“Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior”) e aos demais envolvidos que contribuíram cientificamente para o sucesso desta pesquisa. O presente trabalho foi parcialmente financiado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES).

REFERÊNCIAS BIBLIOGRÁFICA

- [1] A. P. Rajan, “A review on serverless architectures - function as a service (FaaS) in cloud computing”. TELKOMNIKA (Telecommunication Computing Electronics and Control), 2020.
- [2] J. Nupponen, D. Taibi, “Serverless: What it Is, What to Do and What Not to Do”. 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), 2020.
- [3] E. Jonas, et al. “Cloud programming simplified: A berkeley view on serverless computing”. arXiv preprint arXiv:1902.03383, 2019.
- [4] G. Mcgrath, P. R. Brenner, “Serverless Computing: Design, Implementation, and Performance”. 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2017.
- [5] A. Jangada, D. Pinckney, Y. Brun, A. Guha, “Formal foundations of serverless computing”. Proceedings of the ACM on Programming Languages, v. 3, n. OOPSLA, p. 1-26, 2019.
- [6] N. Mahmoudi, H. Khazaei, “SimFaaS: A Performance Simulator for Serverless Computing Platforms”. Proceedings of the 11th International Conference on Cloud Computing and Services Science, 2021.
- [7] L. Wang, M. Li, Y. Zhang, T. Ristenpart, M. Swift, “Peeking behind the curtains of serverless platforms”. USENIX Annual Technical Conference (USENIX ATC 18), pages 133–146, 2018.
- [8] K. Kritikos, P. Skrzypek, “Simulation-as-a-Service with Serverless Computing”. 2019 IEEE World Congress on Services (SERVICES), 2019.