UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIS EDUARDO BARROSO MAFRA

AVALIAÇÃO DA TÉCNICA DE SNAPSHOTTING PARA MITIGAR OS EFEITOS DO COLD START DE FUNÇÕES JAVASCRIPT EXECUTADOS EM AMBIENTE FAAS

CAMPINA GRANDE - PB

2023

LUIS EDUARDO BARROSO MAFRA

AVALIAÇÃO DA TÉCNICA DE SNAPSHOTTING PARA MITIGAR OS EFEITOS DO COLD START DE FUNÇÕES JAVASCRIPT EXECUTADOS EM AMBIENTE FAAS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Thiago Emmanuel Pereira da Cunha Silva
Professor Daniel Lacet de Faria Fireman

CAMPINA GRANDE - PB

2023

LUIS EDUARDO BARROSO MAFRA

Avaliação da técnica de snapshotting para mitigar os efeitos do cold start de funções Javascript executados em ambiente FaaS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Thiago Emmanuel Pereira da Cunha Silva
Orientador – UASC/CEEI/UFCG

Professor Claudio Baptista
Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 14 de Fevereiro de 2021.

CAMPINA GRANDE - PB

RESUMO (ABSTRACT)

The serverless computing model allows the creation and execution of applications in the cloud, delegating to the platform the responsibility for managing and scaling the infrastructure. As a result, billing for services considers only the execution time of requests, naturally resulting in efficient use of resources. This strategy aims to reduce the costs of keeping services running, but it comes with a burden: starting applications takes time (cold start), and doing this whenever the application is requested can be a hindrance to their performance, especially in a highly scalable environment. The Prebaking technique emerges as a solution to this problem, using the method of snapshotting the state of a process to deal with the cold start, obtaining good results for applications created in Java. In this direction, this work aims to evaluate the use of this method to reduce the cold start of Javascript applications that use the V8 runtime.

Avaliação da técnica de snapshotting para mitigar os efeitos do cold start de funções Javascript executados em ambiente FaaS

Luis Eduardo Barroso Mafra
luis.mafra@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, PB

Thiago Emmanuel Pereira
(Orientador)
temmanuel@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, PB

Daniel Lacet de Faria Fireman
(Co-Orientador)
daniel.fireman@ifal.edu.br
Instituto Federal de Alagoas
Arapiraca, AL

RESUMO

O modelo de computação serverless permite a criação e execução de aplicações na nuvem, delegando para a plataforma a responsabilidade de gerenciamento e escalonamento da infraestrutura. Com isso, a cobrança pelos serviços considera apenas o tempo de execução de requisições, havendo naturalmente um uso eficiente de recursos. Essa estratégia visa reduzir custos de manter os serviços executando, mas vem com um ônus: iniciar aplicações demandam um tempo (cold start), e fazer isso sempre que a aplicação for requisitada pode ser um empecilho para o desempenho delas, principalmente em um ambiente altamente escalável. A técnica de Prebaking surge como uma solução para esse problema, utilizando o método de *snapshot* do estado de um processo para lidar com o cold start, obtendo bons resultados para aplicações criadas em Java. Nessa direção, este trabalho visa avaliar a utilização desse método para a redução do cold start de aplicações Javascripts que utilizam a runtime V8.

1 INTRODUCÃO

A computação em nuvem foi fortalecida pelo modelo de computação sem servidor (*serverless computing*), que implanta e inicializa servidores HTTP sem intervenção manual dos usuários, além de apenas contabilizar e cobrar os recursos utilizados durante o processamento das requisições. Para economizar recursos, provedores de nuvem que oferecem este modelo fazem *downscaling* para zero de servidores que não estão recebendo requisições, e portanto, normalmente é utilizado para servir aplicações sem estado (*stateless*). Existem 2 abordagens comuns em serverless: BaaS (*Backend as a Service*) e FaaS (*Function as a Service*). O BaaS oferece um backend como um serviço, no qual é gerenciado e mantido pela própria plataforma. Essa abordagem permite que os desenvolvedores possam terceirizar o gerenciamento de recursos e máquinas virtuais, podendo se concentrar apenas do desenvolvimento do frontend. FaaS utiliza funções codificadas pelos usuários para criar instâncias (réplicas) de servidor sob demanda para lidar com as requisições. Dessa forma, os desenvolvedores precisam abstrair os componentes arquiteturais de uma aplicação em funções, que poderão ser chamadas por meio de APIs gerenciadas pelo provedor de FaaS.

O modelo serverless vêm se popularizando nos últimos anos, estudos em 2020[11] mostraram que houve um aumento anual de 75% de seu uso entre 2018 e 2020, e a expectativa do mercado é que a taxa de crescimento anual entre 2023 e 2028 seja de 22,2%[2]. Esse modelo tem se tornado bastante atrativo por permitir que os desenvolvedores possam se concentrar apenas na implementação da lógica do negócio, deixando de lado toda a parte de provisionamento e gerenciamento de recursos, e cobrando apenas pelo tempo de execução das requisições.

Devido ao seu modelo de cobrança, a plataforma naturalmente não mantém recursos que estão desnecessariamente alocados, nos levando ao problema do *cold start* [7]. O cold start está relacionado ao atraso na execução da função quando não existe uma instância de servidor disponível para lidar com a requisição HTTP, dessa forma, o usuário precisa esperar que o provedor de nuvem inicie uma nova instância de servidor, afetando diretamente o tempo de resposta da requisição. O cold start é um problema conhecido, e já houve trabalhos com o objetivo de mitigar seus efeitos [5, 6, 9, 10]. Dentre as etapas relacionadas ao cold start, temos:

(1) O tempo que leva para provisionar recursos e iniciar o ambiente para execução, podendo ser máquinas virtuais ou contêineres.
(2) Inicio do processo e da runtime que irá executar a função.
(3) Etapas de inicialização específicas do servidor, que inclui carregamento do código fonte da função e suas dependências.

Para redução do cold start, uma técnica chamada Prebaking foi desenvolvida e teve sua eficiência avaliada [10]. Essa técnica tem como objetivo criar um snapshot do processo de uma instância de servidor, de tal forma que esse snapshot possa ser reutilizado posteriormente, evitando a necessidade de execução das etapas 2 e 3, descritas anteriormente. Esse trabalho não busca resolver o problema da etapa 1, visto que o provisionamento de recursos é responsabilidade da plataforma. O Prebaking Warm [10] é uma variação da técnica de Prebaking, que faz uma requisição para aquecer o servidor, ou seja, inicializar estruturas de dados internas e carregar dependências, para só após isso criar o snapshot.

Os resultados apresentados demonstram que a técnica reduz significativamente o cold start, especialmente quando o servidor é submetido à uma requisição de aquecimento antes de gerar o snapshot. Apesar disso, a técnica foi avaliada utilizando apenas funções escritas em Java, ou seja, mesmo que tenha um bom resultado, não temos evidência empírica que ela é eficiente para outras linguagem. As linguagens para FaaS são geralmente linguagens gerenciadas, como Python, Java, Javascript ou Go, e são gerenciadas por suas próprias runtimes de maneiras distintas. Portanto, o nosso objetivo foi avaliar se o Prebaking e o Prebaking Warm possuem o mesmo efeito de reduzir o cold start para funções Javascript, que, diferentemente de Java, é uma linguagem compilada em tempo de execução, e utiliza a runtime V8 para essa compilação.

O restante do artigo é organizado da seguinte forma: A seção 2 busca contextualizar o que o leitor precisa saber para entender o trabalho, com mais detalhes sobre plataformas FaaS, o problema do cold start, e as técnicas de snapshot utilizadas. A seção 3 apresenta outros trabalhos realizados com o mesmo objetivo de mitigar os efeitos do cold start. A seção 4 descreve o design do experimento utilizado para avaliar o impacto de snapshotting em funções Javascript. A seção 5 apresenta os resultados obtidos, e como o cold start se comportou para as diferentes técnicas de snapshotting. E para finalizar, na seção 6 trazemos as considerações finais e possíveis limitações.

2 FUNDAMENTAÇÃO TEÓRICA

Para o entendimento do experimento e do que estamos querendo avaliar, é preciso compreender também alguns conceitos importantes sobre nosso objeto de estudo, que é o **cold start** de funções como serviço. Antes de tudo é preciso entender o que é uma **Função como serviço(FaaS)**. FaaS é um modelo de serviço das plataformas de nuvem que permite com que os usuários criem blocos de código em formato de função em alguma linguagem de alto nível, deixando todo o provisionamento e inicialização de recursos como responsabilidade do provedor. No paradigma FaaS, a função só é acionada por um evento de um usuário (por exemplo, uma chamada à API), que irá disparar a criação de uma réplica de servidor executando a função do usuário, passando pelas etapas de inicialização necessárias, que inclui criação de máquinas virtuais até o carregamento do código fonte e suas dependências. Após a execução da função, a plataforma irá manter a réplica disponível por alguns minutos para o caso de chegar uma nova requisição. Caso contrário, a plataforma irá terminar a réplica do servidor, e a próxima requisição precisará passar pelas mesmas etapas de inicialização. Essa prática leva ao conhecido problema de *cold start*, que é o atraso na execução de uma função, quando não existe uma réplica de servidor disponível para executá-la.

A Figura 1 apresenta uma visualização da execução de requisições segundo o modelo de concorrência na AWS (*Amazon Web Services*), potencializando a necessidade de mais instâncias de servidor. A imagem mostra o fluxo de requisições chegando, e como o tempo pode ser afetado. A requisição 1 é a primeira a chegar, o que desencadeia a criação de uma instância de servidor para processar a requisição, passando por etapas de inicialização antes da execução de fato da função. Na chegada da requisição 2, a instância de servidor anteriormente criada ainda está disponivel na plataforma, e portanto, é reutilizada para servir a requisição, fazendo com que a requisição seja processada mais rápido que a anterior. Após um determinado tempo, essa instância de servidor é terminada pela plataforma para guardar recursos, fazendo com que a próxima requisição a chegar, a terceira nesse caso, precise novamente passar pelas etapas de inicialização do servidor. A requisição 4 chega enquanto a instância de servidor está ocupada processando a terceira requisição, e portanto, a plataforma também precisa instanciar um novo servidor para processá-la. O tempo de execução dessas funções são atrasadas devido à espera pela inicialização do servidor, problema esse que poderia ser mitigado caso seja possível reduzir esse tempo de inicialização.

A técnica de Prebaking [10] surge como uma alternativa para reduzir o cold start, e garantir que a função seja executada o mais rápido possível. Essa técnica utiliza um software para Linux chamado de CRIU (Checkpoint/Restore In Userspace), que pausa um processo em execução e salvar um checkpoint em disco do seu estado atual [1]. Os dados salvos podem ser utilizados para recuperar o processo em execução no mesmo estado anterior. Uma variação dessa técnica na qual chamaremos de **Prebaking Warm** [10], salva o estado do processo após uma execução da função já ter sido realizada, com o intuito de carregar dependências em memória para que futuras execuções referentes a essa função sejam atendidas mais rapidamente. A figura 2 mostra visualmente como o snapshot é criado para as diferentes técnicas, e como o processo é restaurado utilizando sempre o CRIU.

3 REVISÃO DA LITERATURA

O cold start é um problema recorrente para quem atua utilizando modelo serverless em plataformas de nuvem, e portanto, já vem sendo objeto de estudo. Trabalhos similares foram realizados que visam também mitigar os efeitos do cold start de FaaS com diferentes estratégias, sendo com uso de snapshot a mais comum entre elas.

O SEUSS(*Serverless Execution via Unikernel SnapShot*)[5, 6] é um método que utiliza unikernels para isolar a execução de aplicações. Unikerneles são kernels especiais onde todos os processos compartilham o mesmo espaço de memória, e é construído utilizando os sistemas operacionais como biblioteca[3]. O SEUSS melhora o cold start através de snapshots do estado do unikernel em diferentes momentos do ciclo de vida da função. Os resultados para o SEUSS mostraram que a latência do cold start para uma função NOOP foi de 7,5 ms no pior caso. Apesar de suas vantagens, a decisão de utilizar unikernels pode tornar mais complexa a integração do método com plataformas serverless.

Como forma de complementar a estratégia de snapshots, houve um trabalho com foco em avaliar o uso de snapshots para mitigar o efeito do cold start em FaaS para diferentes dispositivos de armazenamento[4]. A ideia utilizada é semelhante à abordagem do Prebaking, utilizando de snapshots para guardar o estado de instâncias de servidores já invocadas. A performance do cold start é avaliada armazenando snapshots em diferentes tipos de disco, bem como os trade-offs de se utilizar cada um, para determinar como aproveitá-los para um alocamento eficiente de snapshots. Apesar disso, a análise não se estendeu para qual a runtime das funções avaliadas, mas focou apenas nos tipos de discos.

Algumas estratégias mais robustas também surgiram, como otimizar a própria plataforma serverless [9], dentre elas, fazer pings periodicamente para reduzir a frequência de ocorrências do cold start.

4 METODOLOGIA

Para avaliar a técnica de snapshotting iremos direcionar o foco em responder 2 questões principais:

(1) É possível reduzir o cold start de funções Javascript usando a técnica de Prebaking?
(2) O tempo de execução da função é afetada após ser restaurada a partir de um snapshot?

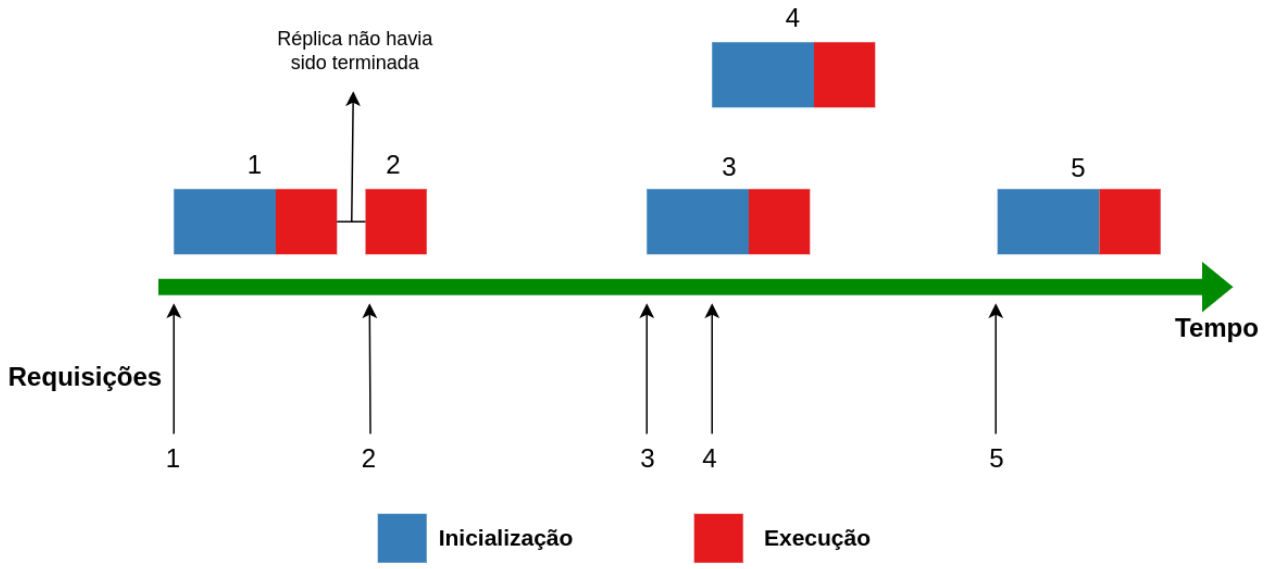

Figura 1: Tempos que compõem o cold start para instanciar réplicas de servidores, em um modelo FaaS concorrente na AWS

Para responder as questões de pesquisa, executamos experimentos e medimos o **tempo de inicialização** de um servidor HTTP para lidar com as requisições, e o **tempo de resposta** de 200 execuções de função. O modelo de concorrência usado pela AWS é de que cada réplica de função só lida com uma requisição por vez, ou seja, se uma réplica estiver ocupada quando uma requisição chegar, a plataforma irá criar uma nova réplica para processar a nova requisição. E caso a réplica esteja inativa por determinado tempo, a plataforma irá terminá-la para guardar recursos[8]. Para reproduzir esse comportamento, nosso experimento iniciou um servidor para lidar com a função, e apenas a primeira requisição precisou esperar que o servidor estivesse devidamente pronto. Após isso, as próximas requisições foram enviadas sequencialmente e de forma constante. Portanto, o que medimos e chamamos de **cold start** afeta apenas a primeira requisição da função.

Para inicializar o servidor HTTP, variamos 3 técnicas possíveis:

- **Vanilla**: Não utiliza nenhuma técnica, apenas as configurações padrões da V8
- **Prebaking**: Pausa um processo em execução e salva um checkpoint em disco do seu estado atual. Os dados salvos são reutilizados posteriormente para recuperar o processo em execução no mesmo estado anterior.
- **Prebaking Warm**: Salva o estado do processo após uma execução da função já ter sido realizada.

Como o tempo de resposta ou de inicialização pode variar dependendo da característica das funções, avaliamos inicialmente para 3 funções diferentes: **NOOP**, **Markdown Render** e **Image Resizer**.

A função **NOOP** é bem simples, não faz nada e retorna sucesso para toda requisição que chegar nela. Esta função será a base para detectarmos o impacto do snapshot no cold start, visto que o tempo de execução da função é praticamente nulo. Por sua vez, a função **Markdown Render** converte um arquivo markdown fixo para uma página HTML, e retorna um pedaço dessa página como resposta da requisição. E por último, a função **Image Resizer** carrega uma imagem e reduz a escala dela em 10% para cada requisição que chega.

Além disso, para medir a eficiência do aquecimento dos snapshots, tal qual é feito por trabalhos anteriores[10], executamos a mesma bateria de experimentos para 3 funções sintéticas que carregam diferentes quantidades de dependências de código (módulos) para a execução da função: **Small**, que carrega 50 módulos, **Medium**, que carrega 250 módulos, e **Large**, que carrega 1250 módulos.

Para termos uma melhor confiança estatística, repetimos o experimento 200 vezes. Todos as execuções dos experimentos foram realizados utilizando a versão 17.0.1 do Node, e executados em uma máquina virtual de uma cloud openstack, com 8vCPU, 32GB RAM, HDD e o Ubuntu 18.04. O código das funções e o script usado para executar os experimentos podem ser encontrados nesse repositório: https://github.com/luismafra/tcc.

5 RESULTADOS

As técnicas de Prebaking e Prebaking Warm reduzem o cold start utilizando os snapshots de servidores pré inicializados, seja com snapshots não aquecidos, como é feito no Prebaking, seja com snapshots aquecidos por uma execução, como é feito no Prebaking Warm. A seguir, será apresentada a análise dos experimentos executados, bem como a diferença existente entre o Prebaking e o Prebaking Warm.

5.1 Impacto dos snapshots

Podemos visualizar na Figura 3 o cold start apenas da função NOOP. Como essa função não possui lógica ou dependências, o tempo de execução da mesma é similar para as 2 variações de Prebaking utilizadas para instanciar o servidor, ou seja, aquecer o snapshot

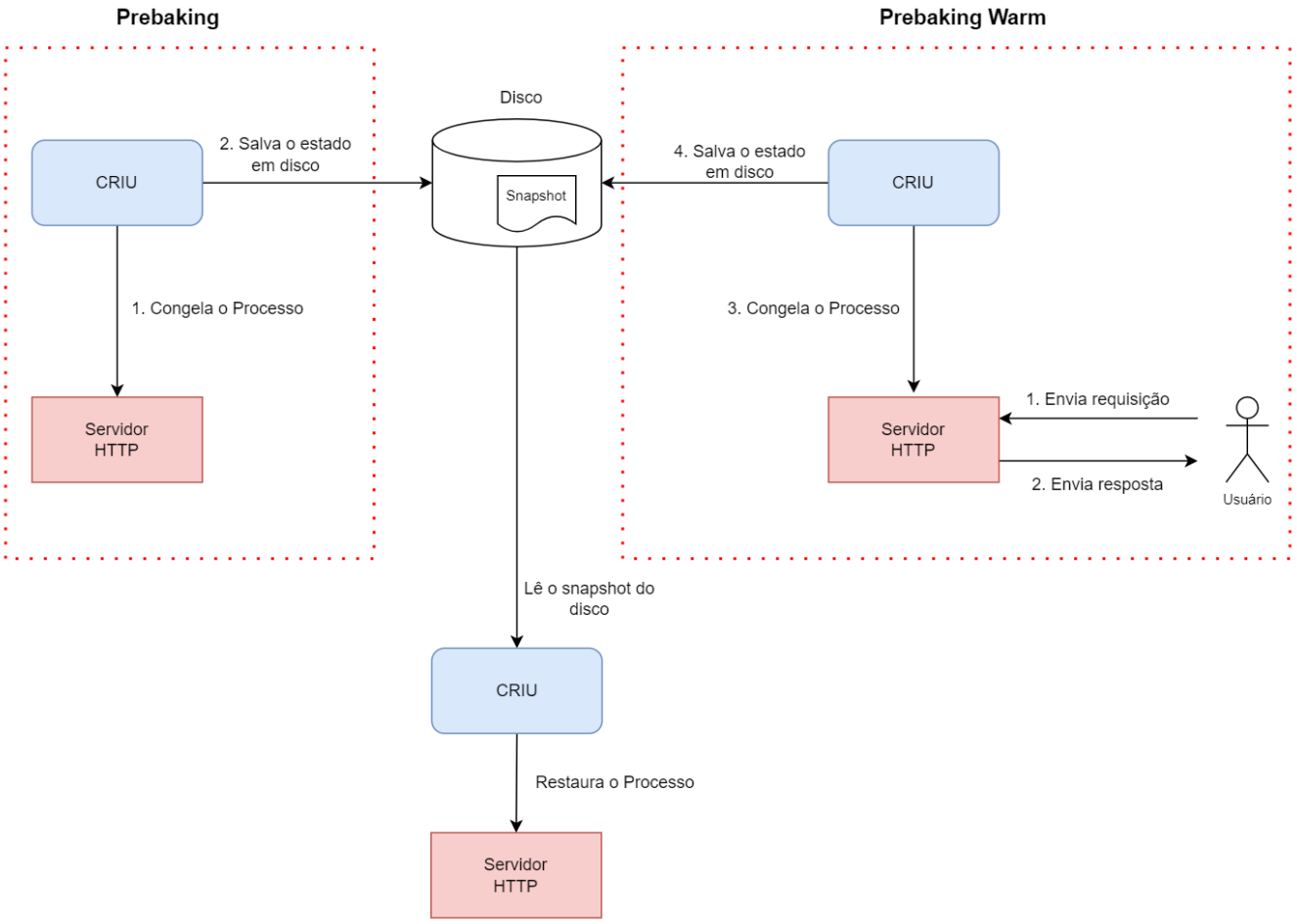

Figura 2: Como os snapshots são criados nas técnicas de Prebaking e Prebaking Warm

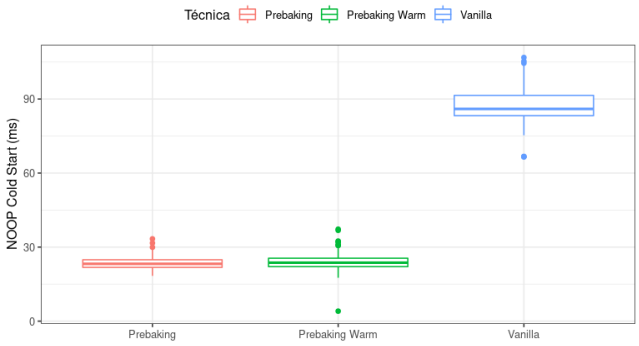

Figura 3: Cold Start (ms) apenas da função NOOP para as diferentes técnicas analisadas

não irá ter um impacto no tempo de resolução da requisição pelo servidor. Portanto, a diferença do cold start entre a técnica Vanilla e as 2 variações de Prebaking é exatamente o ganho obtido de usarmos snapshots para iniciar o servidor, ao invés das configurações padrões da runtime.

5.2 Funções Padrões

A figura 4 mostra o resultado da execução do experimento para as 3 funções escolhidas: NOOP, Markdown e Image Resizer. O gráfico evidencia a redução significativa que aconteceu com as técnicas de Prebaking e Prebaking Warm, em relação a Vanilla, tal comportamento se repetiu para as 3 funções. Outro ponto importante de notar é a melhora do Prebaking Warm em relação ao Prebaking, ocasionado pela execução da função, que foi significativamente reduzida graças ao aquecimento do snapshot. Como podemos ver, a diferença entre as 2 variações do Prebaking só são notadas nas funções Markdown e Image Resizer. Tal fato pode ser explicado devido à suas implementações serem mais robustas, com carregamento de dependências e estrutura de dados internas, que foram armazenados em memória ao utilizar o Prebaking Warm.

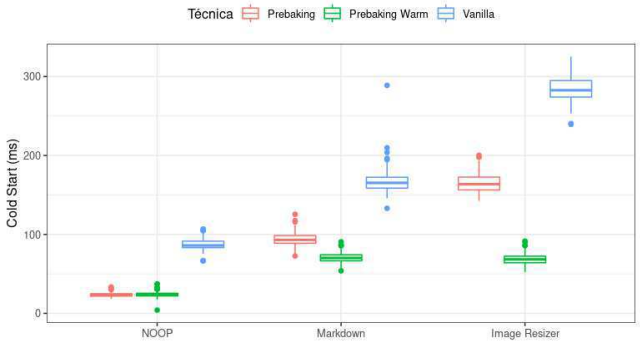

Figura 4: Cold Start (ms) das funções padrões para diferentes técnicas

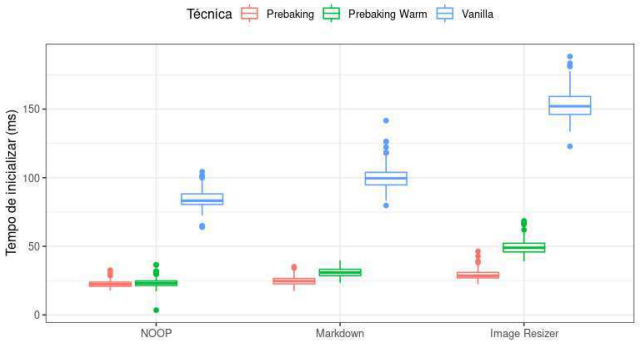

Figura 5: Tempo de inicialização do servidor

Tabela 1: Intervalo de confiança de 95% para a média (ms) do cold start, e o ganho percentual de se usar Prebaking

Função	Vanilla	Prebaking	Redução (%)
NOOP	[86.3, 88.2]	[23.1, 23.8]	[73.0, 73.2]
Markdown	[165.0, 169.0]	[92.6, 94.7]	[43.9, 44.0]
Image Resizer	[282.0, 286.0]	[163.0, 167.0]	[41.8, 42.0]

Tabela 2: Intervalo de confiança de 95% para a média (ms) do cold start, e o ganho percentual de se usar Prebaking Warm

Função	Vanilla	Prebaking Warm	Redução (%)
NOOP	[86.3, 88.2]	[23.6, 24.6]	[72.1, 72.7]
Markdown	[165.0, 169.0]	[69.9, 71.5]	[57.7, 57.7]
Image Resizer	[282.0, 286.0]	[67.9, 69.6]	[75.7, 75.9]

Como é possível ver nas Tabelas 1 e 2, tanto o Prebaking como o Prebaking Warm possuem uma redução percentual do cold start em relação à técnica padrão Vanilla, porém, a redução é de até 75% para o Image Resizer na técnica de Prebaking Warm, enquanto para o Prebaking possuem redução de apenas 42%. A diferença entre as reduções percentuais acontecem pois as funções são diferentes entre si, tanto em nível de dependências de código, como de implementação.

As figuras 5 e 6 mostram o cold start separado em 2 valores: O tempo que leva para o servidor estar pronto, e o tempo de execução da função. Como é possível ver na Figura 5, o resultado mostra que o tempo de inicialização é maior para o Prebaking Warm em relação ao Prebaking, isso se dá muito provavelmente pelo aquecimento do servidor, que deixa o snapshot bem maior com o carregamento de dependências. Apesar disso, a Figura 6 mostra que o tempo de execução da função possui uma grande melhora para o Prebaking Warm, compensando esse valor de aumento, deixando o Prebaking Warm melhor quando somado esses 2 valores, como foi possível visualizar na Figura 4.

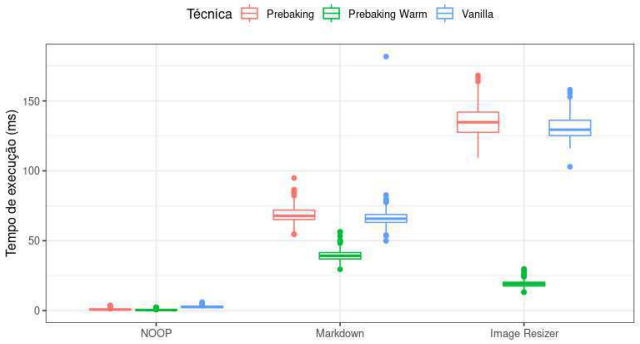

Figura 6: Tempo de execução da função após a inicialização do servidor

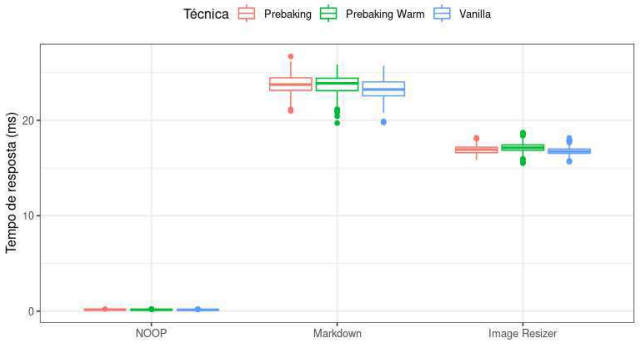

Figura 7: Boxplot da média do tempo de resposta

A figura 7 mostra como o tempo de resposta médio das requisições são bem similares, independente da técnica utilizada. Isso acontece pois a técnica de Prebaking tem como objetivo reduzir o tempo que leva para executar a primeira requisição, dado que ela

Tabela 3: Intervalo de confiança de 95% para a média (ms) do cold start, e o ganho percentual de se usar Prebaking

Função	Vanilla	Prebaking	Redução (%)
Small	[181.0, 187.0]	[115.0, 117.0]	[36.3, 37.4]
Medium	[474.0, 483.0]	[426.0, 432.0]	[10.2, 10.5]
Large	[1696.0, 1723.0]	[1641.0, 1667.0]	[3.2, 3.2]

Tabela 4: Intervalo de confiança de 95% para a média (ms) do cold start, e o ganho percentual de se usar Prebaking Warm

Função	Vanilla	Prebaking Warm	Redução (%)
Small	[181.0, 187.0]	[33.4, 34.4]	[81.5, 81.6]
Medium	[474.0, 483.0]	[65.6, 67.1]	[86.1, 86.2]
Large	[1696.0, 1723.0]	[199.0, 203.0]	[88.2, 88.3]

precisa esperar pelo servidor estar rodando, e portanto, não tem impacto nas requisições subsequentes a ela.

5.3 Impacto do aquecimento

Como o Prebaking Warm tem como objetivo aquecer o servidor para carregar dependências, iremos utilizar as funções sintéticas com o intuito de compreender o impacto do aquecimento dos servidores para diferentes quantidades de dependências.

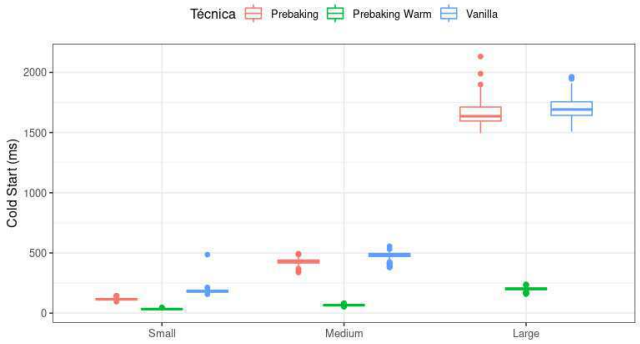

Figura 8: Cold Start (ms) das funções sintéticas para diferentes técnicas

A figura 8 mostra o impacto positivo do Prebaking Warm, que consegue reduzir o cold start da casa dos segundos, para poucas centenas de milissegundos. Apesar disso, o Prebaking não apresentou uma redução tão significativa como acontece no Prebaking Warm. Isso acontece pois na técnica de Prebaking, as dependências só são carregadas durante a execução do código, enquanto no Prebaking Warm essas dependências já estão carregadas no snapshot, agilizando o tempo de execução.

Como é possível ver nas tabelas 3 e 4, para o pior caso, que são 1250 módulos carregados, o Prebaking só reduz o cold start em 3%, enquanto o Prebaking Warm reduz em 88%, graças ao aquecimento do servidor que é feito antes de criar o snapshot, garantindo que a execução da função pela primeira vez seja muito mais rápida devido aos módulos já estarem carregados em memória.

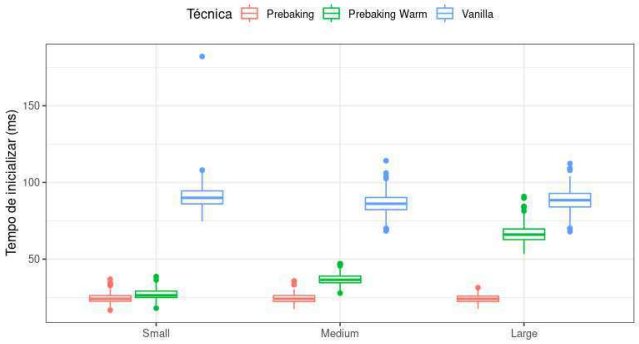

Figura 9: Tempo de inicialização do servidor

A diferença entre o Prebaking e Prebaking Warm no cold start é notável, a figura 9 mostra como o tempo de inicialização das funções com Prebaking Warm foi superior em relação ao Prebaking, e isso se deve pelo fato do Prebaking Warm criar um snapshot a partir de um servidor com vários módulos carregados, o que leva a uma imagem com um tamanho maior, e consequentemente, mais demorada de se carregar. Apesar disso, esse tempo de inicialização ainda é menor que o Vanilla, indicando que ainda assim esse método é mais eficiente nesse quesito. Com a figura 10 conseguimos perceber que o tempo de execução da função para o Prebaking Warm é bem inferior em relação ao Prebaking, o que compensa o tempo de inicialização maior, como foi possível ver na figura 8.

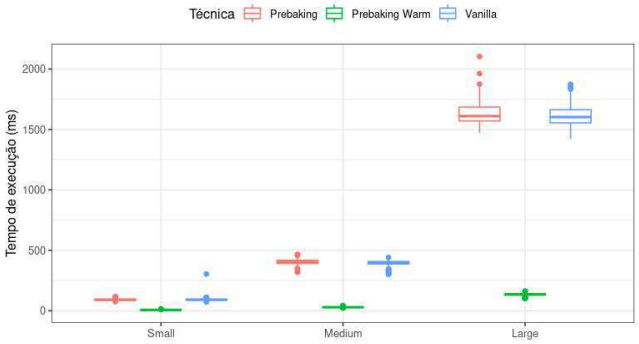

Figura 10: Tempo de execução da função após a inicialização do servidor

Para complementar, a Figura 11 mostra que a técnica também não impacta para as funções sintéticas no tempo de resposta das execuções subsequentes à primeira. Isso acontece independente se houver ou não módulos carregados em memória, pois a partir da segunda execução já irá manter um tempo de resposta consistente devido ao uso de memória.

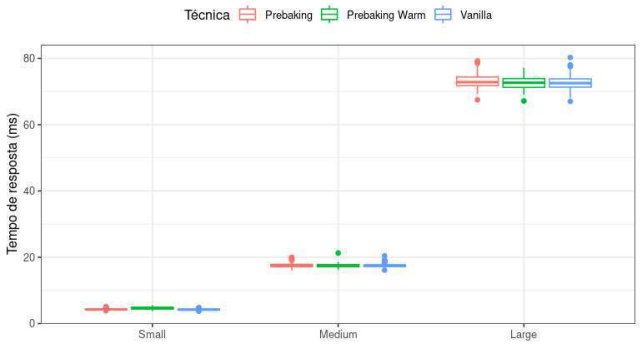

Figura 11: Boxplot da média do tempo de resposta

6 CONCLUSÕES

Plataformas serverless sofrem de um problema conhecido como cold start, que atrasa o processamento de uma requisição até que possua uma réplica de função pronta para processá-la. Vários trabalhos já foram realizados com o intuito de resolver esse problema, e dentro desses trabalhos estava o de Prebaking, que visa utilizar snapshots para pular algumas etapas de inicialização e mitigar os efeitos do cold start na execução da função. Como a técnica de Prebaking só havia sido avaliada para Java, seria preciso garantir que ela funciona para outras linguagens, visto que as plataformas serverless podem conter funções implementadas em diversas linguagens.

Para isso, o foco desse trabalho foi validar a técnica de Prebaking para funções em Javascript, outra linguagem que também é utilizada em plataformas serverless. Os resultados mostraram que tanto o Prebaking como o Prebaking Warm reduziram significativamente o cold start das funções, tirando vantagem do uso de snapshots. Com a análise dos resultados, conseguimos ver que tanto o Prebaking quanto o Prebaking Warm reduziram de forma similar o cold start da função NOOP, em torno de 73% e 72%, respectivamente, mostrando que quando a função não possui lógica, o aquecimento ou não dela é indiferente para o cold start. Por outro lado, isso não se repete para as funções de Markdown e Image Resizer, tendo o Prebaking Warm se saído melhor na redução para as 2 funções. Já para as funções sintéticas, o Prebaking Warm demonstrou que o impacto do aquecimento é proporcional à quantidade de dependências de código, e etapas realizadas na inicialização da função. Os resultados mostraram que ele consegue reduzir o cold start em mais de 80%, enquanto o Prebaking mostrou uma redução de até 36%, tendo sido 3% no pior caso.

REFERÊNCIAS

[1] 2012. CRIU. https://criu.org/Main_Page Last visited: 2022-06-22.
[2] 2022. Serverless Computing Market. https://www.expertmarketresearch.com/reports/serverless-computing-market
[3] James Cadden Jonathan Appavoo Ulrich Drepper Richard Jones Orran Krieger Renato Mancuso Ali Raza, Parul Sohal and Larry Woodman. 2019. Unikernels: The next stage of linux's dominance. *In Proceedings of the Workshop on Hot Topics in Operating Systems* (2019).
[4] Vasileios Karakostas Konstantinos Nikas Georgios Goumas Nectarios Koziris Christos Katsakioris, Chloe Alverti. 2022. FaaS in the age of (sub-)s I/O: a performance analysis of snapshotting. *SYSTOR '22: Proceedings of the 15th ACM International Conference on Systems and Storage* (2022).
[5] Yara Awad Han Dong Orran Krieger Jonathan Appavoo James Cadden, Thomas Unger. 2019. SEUSS: rapid serverless deployment using environment snapshots. (2019).
[6] Yara Awad Han Dong Orran Krieger Jonathan Appavoo James Cadden, Thomas Unger. 2020. SEUSS: skip redundant paths to make serverless fast. *EuroSys '20: Proceedings of the Fifteenth European Conference on Computer Systems* (2020).
[7] Tobias Heckel Guido Wirtz Johannes Manner, Martin EndreB. 2018. Cold Start Influencing Factors in Function as a Service. *Service. In 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018, Zurich, Switzerland* (2018).
[8] Yinqian Zhang Thomas Ristenpart Liang Wang, Mengyuan Li and Michael M. Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. (2018).
[9] Fereidoon Shams Alie Parichehr Vahidinia, Bahar Farahani. 2020. Cold Start in Serverless Computing: Current Trends and Mitigation Strategies. (2020).
[10] Thiago Emmanuel Pereira Paulo Silva, Daniel Fireman. 2020. Prebaking Functions to Warm the Serverless Cold Start. *Middleware '20: Proceedings of the 21st International Middleware Conference* (2020).
[11] Emrah Samdan. 2020. Serverless is Taking Off: Here's Why It's Worth Hopping On. https://blog.thundra.io/serverless-is-taking-off-heres-why-its-worth-hopping-on

AGRADECIMENTOS

Esse trabalho foi financiado pela MCTIC/CNPq-FAPESQ/PB (EDITAL Nº 010/2021) e pela VTEX BRASIL (EMBRAPII PCEE1911.0140).