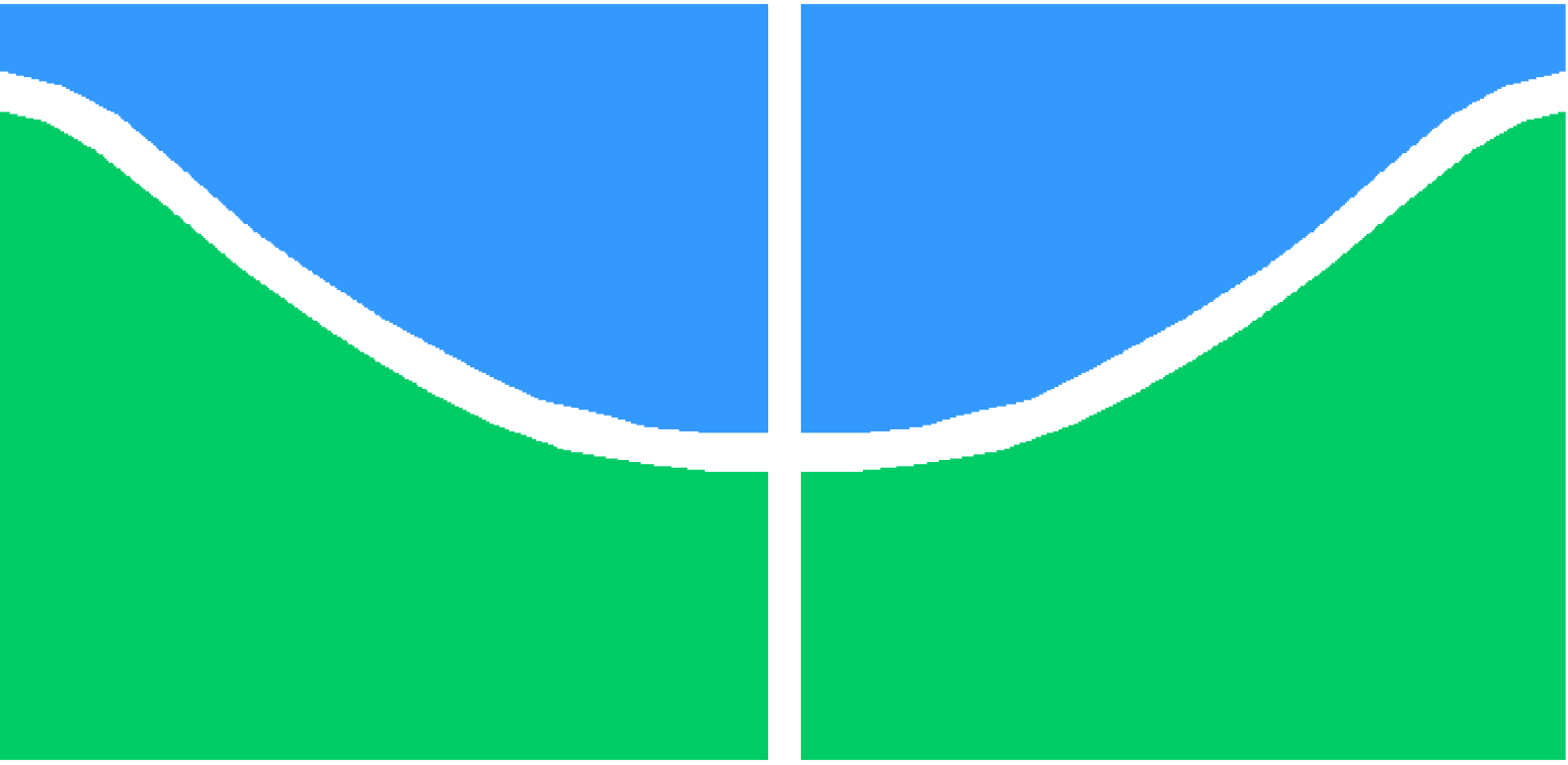

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Implementação de novo testador de desempenho na biblioteca CCF

Autor: William Silva de Almeida
Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF
2021

William Silva de Almeida

Implementação de novo testador de desempenho na biblioteca CCF

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF

2021

William Silva de Almeida
Implementação de novo testador de desempenho na biblioteca CCF/ William Silva de Almeida. – Brasília, DF, 2021-
56 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Tiago Alves da Fonseca

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2021.

1. CCF. 2. Teste de Software. 3. Desempenho. 4. Qualidade de Software. I. Prof. Dr. Tiago Alves da Fonseca. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação de novo testador de desempenho na biblioteca CCF

CDU 02:141:005.6

William Silva de Almeida

Implementação de novo testador de desempenho na biblioteca CCF

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 05/11/2021:

Prof. Dr. Tiago Alves da Fonseca
Orientador

MSc. José Roberto Menezes Monteiro
Convidado 1

MSc. Ricardo Ajax Dias Kosloski
Convidado 2

Brasília, DF
2021

Eu dedico este trabalho à todas pessoas que em algum momento da jornada perderam fé na própria capacidade.

Agradecimentos

Creio que os momentos de sucesso da vida não podem ser resumidos apenas à etapa final, neste caso, o TCC. Considerando essa frase, esses agradecimentos são destinados às pessoas que contribuíram em momentos da minha vida desde que entrei na UNB.

Primeiramente devo agradecer à minha família próxima, que desde o início me proporcionou não apenas segurança financeira mas principalmente apoio emocional e ensinamentos éticos. Seria ingenuidade da minha parte afirmar que eu teria os resultados parecidos sem esse auxílio.

Serei grato eternamente aos meus amigos de vida, que me acompanharam desde antes do ingresso na universidade até os momentos mais difíceis nestes anos. Nos principais momentos em que precisei de distração, conselhos, amenizar as tristezas, comemorar boas fases entre outros, pude contar com o apoio destas grandes pessoas a quem não desejo nada mais do que todo sucesso.

No meu período do estágio, tenho a agradecer à muitas pessoas que tive contato na Scytl. Foram 2 anos de uma fase importante da vida a qual tive muitos bons momentos, e mesmo os negativos me levaram a aprender. Essas pessoas me ensinaram coisas muito mais importante do que aspectos de Engenharia de Software e princípios técnicos: aspectos mais profundos da vida que me levaram a evoluir como ser humano. Devo muito à essas pessoas, mesmo as que não tenho mais tanto contato. Tenho certeza que em sua maioria, conseguirão realizar seus objetivos na vida.

Dentro da universidade, conheci algumas pessoas muito diferentes do que eu tive contato antes. Colegas e amigos que mesmo não conhecendo no início do curso, tiveram extrema importância para que eu me mantivesse motivado para finalizar o curso. Muito obrigado.

Ao meu orientador, ao professor Bruno Ribas que me ajudou com o ambiente e aos profissionais da área que consultei antes de iniciar este projeto, minha gratidão por me darem inspiração, motivação e uma luz inicial para realizar este trabalho.

Finally, at the end of this journey, I want to thank you the two engineers from CCF that helped me in this project: Amaury Chamayou and Eddy Ashton. They were very helpful and taught a lot about Software Engineering.

Muito obrigado a todos que me deram suporte nesses últimos anos da minha vida.

"If you're offered a seat on a rocket ship,
don't ask what seat.
Just get on."
(Sheryl Sandberg)

Resumo

O crescente uso das criptomoedas é um dos fatores que popularizaram o *blockchain*. Esse modelo de transações descentralizadas atraiu organizações, principalmente com *blockchains* permissionados. O *Confidential Consortium Framework* é uma ferramenta de código aberto que oferece uma rede de *blockchains* com alto-desempenho e confidencialidade. Ao decorrer da evolução do CCF, o testador de desempenho tornou-se obsoleto porque foi implementado para suportar um protocolo de internet modificado que não é mais necessário. O objetivo deste trabalho é exercitar técnicas de Engenharia de Software aplicadas ao desenvolvimento de uma nova ferramenta de teste de desempenho para a biblioteca CCF, adicionado novas funcionalidades e requisitos.

Palavras-chaves: CCF. Teste de Software. Desempenho. Qualidade de Software

Abstract

The growing usage of cryptocurrencies reflects the popularization of blockchain. This model of decentralized transactions attracted organizations, specifically those interested in permissioned blockchains. Confidential Consortium Framework (CCF) is a open-source tool that offers a blockchain network with high-performance and confidentiality. In the course of CCF evolution, the performance testing client became outdated because it was built to support a custom network protocol which is no longer needed. The goal of the present work is to exercise software engineering techniques applied to development of a new performance testing client for CCF, adding new features and missing requirements.

Key-words: CCF. Software Testing. Performance. Software Quality

Lista de ilustrações

Lista de abreviaturas e siglas

CCF	*Confidential Consortium Framework*
ID	*Identificador Único*
XP	*Extreme Programming*
HTTP	*Hypertext Transfer Protocol*
MVP	*Minimum Viable Product*
TCC	Trabalho de Conclusão de Curso
TEE	*Trusted Execution Environments*

Sumário

1 Introdução

Blockchain tornou-se uma palavra frequentemente usada em ambientes tecnológicos. Essa tecnologia permite a realização de transações descentralizadas e transparentes, porém o fato de todos poderem ter acesso às informações contidos na rede é um fator que inibe o uso dessa tecnologia por parte de organizações.

Como uma alternativa ao *blockchain* público, onde todos podem ter acesso aos dados, surgiu o conceito de *Consortium Blockchain*. Esse modelo alternativo tem como principal diferencial o permissionamento das operações relacionadas à transações, permitindo que seja possível restringir a escrita e leitura de dados apenas por um grupo selecionado. Porém, bibliotecas que oferecem confidencialidade apresentam uma desvantagem: o processo para verificação de uma transação é mais lento comparado à uma rede pública (DIB et al., 2018).

O *Confidential Consortium Framework* (CCF) foi lançado como uma alternativa de *blockchain* que oferece redes unindo alto desempenho e confidencialidade. Apesar desses pontos positivos, um fator que aparece como ponto negativo é a falta de ferramentas para testes de *software*. A aplicação até conta com uma ferramenta para testes de desempenho, porém ela tornou-se obsoleta com a evolução do CCF.

Com os fatores apontados como motivação, observou-se uma oportunidade de contribuição para esse repositório, criando um testador de desempenho com tecnologias atuais e de acordo com requisitos atuais do *framework*. Com o apoio para a validação por parte dos desenvolvedores e o uso de conceitos aprendidos durante o curso de Engenharia de *Software*, este trabalho propõe a implementação de uma aplicação para testes de desempenho.

1.1 Justificativa

Testes de Software são uma importante parte da área de Engenharia de Software. Por meio deles é possível gerar métricas, como as de desempenho, para avaliação e obtenção de *feedbacks* sobre a aplicação. A biblioteca de código-aberto *Confidential Consortium Framework* é um componente de *software* que oferece uma rede de *blockchains* com alto desempenho, mas atualmente não é possível obter parâmetros de desempenho sobre ela por conta de seu testador obsoleto. Dado o contexto, foi possível iniciar o desenvolvimento de um novo testador para a biblioteca CCF, com o diferencial de se contar com a validação das contribuições por parte dos desenvolvedores da biblioteca.

A oportunidade de contribuição para uma biblioteca de código-aberto é bom espaço

para aplicar e aprimorar conhecimentos de Engenharia de Software adquiridos durante o tempo de curso, não só tecnicamente, mas práticas também relacionadas à metodologia e boas rotinas. Junto com o contato direto com desenvolvedores mais experientes, notou-se uma chance para gerar conhecimento de maneira pública e aprimorar habilidades relacionadas à este trabalho.

1.2 Objetivos

1.2.1 Objetivo Geral

O principal objetivo desse artigo é o desenvolvimento de uma aplicação que sirva como testador de desempenho para a biblioteca *Confidential Consortium Framework*.

1.2.2 Objetivos Específicos

1. Desenvolver um testador de desempenho com tecnologias acessíveis;
2. Introduzir entendimento sobre tecnologias de *consortium blockchain* de alto desempenho;
3. Atualizar a ferramenta de testes da biblioteca, adicionando funcionalidades ausentes;

1.3 Estrutura do Documento

O documento em questão é composto por 5 principais capítulos:

1. **Introdução:** contém a contextualização do trabalho junto à definição dos objetivos planejados;
2. **Fundamentação Teórica:** introdução de conceitos teóricos entendidos como importantes para compreensão da ideia apresentada;
3. **Proposta de Trabalho:** descrição e aprofundamento das decisões relacionadas à implementação e planejamento do trabalho;
4. **Metodologia:** contém o esclarecimento da metodologia aplicada na elaboração deste trabalho.
5. **Procedimentos:** este capítulo contém detalhes da execução de testes feitos usando o testador de desempenho.
6. **Resultados e Discussões:** apresentação de dados relacionados aos resultados após testes junto às explicações e descrições dos retornos obtidos após os testes.

7. **Conclusão:** resumo dos procedimentos e descobertas executados durante este trabalho.

2 Fundamentação Teórica

2.1 Testes de Software

Um conceito importante a ser abordado nesse trabalho será o de testes no contexto de *software*. Como definido no livro *Introduction to Software Testing* (AMMANN; OFFUTT, 2008), o ato de testar se refere à avaliação de um sistema através da observação de sua execução. Segundo Sommerville, existem diferentes modelos de testes para diversas finalidades (SOMMERVILLE, 2011). A categoria relevante para esse trabalho é a de Testes de Desempenho, onde se verifica a carga máxima que um sistema pode processar até desviar de suas características esperadas.

Uma forma diferente de se categorizar testes é entre as nomenclaturas de automatizados e manuais. Focando nos automatizados, podem ser divididos em 4 categorias de testes (MAHAJAN; SHEDGE; PATKAR, 2016):

- Confiabilidade;
- Segurança;
- Corretude;
- Desempenho.

2.1.1 Testes automatizados

Os testes automatizados serão necessários por conta do trabalho se tratar de testes de desempenho. Os motivos dessa combinação são: confiança nos resultados, naturalmente repetíveis, reuso de *scripts* e a detalhamento dos *logs*(MAHAJAN; SHEDGE; PATKAR, 2016). Também é relevante citar dois pontos apresentandos no livro *Introduction to Software Testing*: a possibilidade de diminuir erros por conta de omissão de etapas da rotina de testes e a eliminação de algumas variáveis que podem impactar nos resultados esperados. (AMMANN; OFFUTT, 2008)

O artigo *Automation Testing in Software Organization*(MAHAJAN; SHEDGE; PATKAR, 2016) cita 6 estágios da automatização de testes, sendo eles:

- Análise de viabilidade;
- Projeto da solução;
- Desenvolvimento dos *scripts*;

- Implantação;
- Execução dos *scripts*;
- Manutenção.

Devido ao foco na implementação de um testador, o artigo irá focar principalmente nas etapas de projeto e desenvolvimento da solução.

2.1.2 Testes de desempenho

Muitos sistemas possuem metas relacionadas a desempenho, como tempo de transferência e tempo de resposta dadas certas condições (MYERS; BADGETT; SANDLER, 2012). O *Confidential Consortium Framework CCF*, o *framework* foco de nosso estudo, não possui números relacionados às metas citadas. Porém segundo sua documentação, são esperadas altas taxas de transferências, baixo tempo de resposta e alta disponibilidade. Dados relacionados à essas propriedades podem ser obtidos a partir de testes de desempenho.

Observando o objetivo de alto desempenho do *framework* foco de estudo, os testes de desempenho se tornam importantes para descobrir os limites do sistema. Além disso, é possível obter dados sobre o funcionamento utilizando tanto dados próximos às rotinas de uso, quanto volume de dados além dos valores usuais para elaborar requisitos de desempenho compatíveis com a realidade do produto (SOMMERVILLE, 2011).

2.2 *Confidential Consortium Framework* (CCF)

O *framework* objeto de estudos neste trabalho é o *Confidential Consortium Framework*, uma infraestrutura de *blockchain* gratuita e de código aberto desenvolvida por equipes da *Microsoft* voltada para o uso de organizações. De acordo com o artigo publicado junto ao seu lançamento (SHAMIS et al., 2019), os projetos de *blockchains* atuais não oferecem confidencialidade e desempenho de acordo com que muitas aplicações necessitam.

Nesse cenário de necessidade de privacidade dos dados junto com outros aspectos técnicos surgiram outros tipos de *blockchain*, como o privado e o *consortium* (DIB et al., 2018), este último que será abordado em um próximo capítulo.

Como exemplo, os autores citam que o *Bitcoin* pode levar até uma hora para confirmar uma transação além de ter seus contratos acessíveis para todos. Tendo outra tecnologia como amostra, há a análise do *Solidus*, um protocolo para transações confidenciais em *blockchains* públicos. Em seu texto de publicação, é possível encontrar a informação de que, apesar de oferecer uma forte confidencialidade, o processo de geração

de provas pode levar até 1 minuto em uma máquina consumidora (CECCHETTI et al., 2017).

O *CCF* foi concebido para ser uma alternativa que resolvesse essas questões: um *framework* de *blockchain* de consórcios com alto desempenho e confidencialidade com foco em computação de várias partes. Sua principal característica é a execução das operações de *blockchain* em uma rede de *hardwares* que possuam ambientes de execução confiável. Dessa forma, é possível prover alta taxa de transferência, alta disponibilidade, latência baixa sem comprometer a integridade e a confidenciabilidade do código e dados em execução no registro (SHAMIS et al., 2019). Outro ponto é a cobertura de testes unitários no CCF: em sua esteira de publicação há uma rotina dedicada a eles a fim de garantir maior qualidade de software.

2.2.1 *Multi-party computation* (MPC)

O livro *A Pragmatic Introduction to Secure Multi-Party Computation*(Evans; Kolesnikov; Rosulek, 2018) afirma que o objetivo da *multi-party computation* (MPC) é permitir um grupo portador de dados, que não possuem relação de confiança entre si, executem uma função em conjunto que depende de todas suas entradas individuais.

Como exemplo, podemos usar o *Yao's Millionaires' problem* (YAO, 1982). Ele consiste na seguinte questão: existem dois milionários que buscam saber quem é o mais rico, e para tal não revelando informações adicionais principalmente o valor de suas fortunas. A função executada em conjunto seria a comparação entre os valores, o número de pessoas se refere ao número de partes e as entradas seriam suas fortunas.

Esse conceito é útil no entendimento do CCF para o entendimento de confiança entre os nós de uma rede. Cada nó executa a mesma aplicação, e mudanças no armazenamento de chaves-valor precisam ser aceitas por um certo números de nós antes de serem replicadas para todos (SHAMIS et al., 2019).

2.2.2 *Blockchain*

Nos últimos anos, as criptomoedas popularizaram um modelo de transações descentralizadas e transparentes. Seu entendimento é importante para a compreensão do *framework CCF*, que se baseia em inovações ligadas ao seu uso. Pela definição do livro Desmistificando *Blockchain*: Conceitos e Aplicações(ALVES et al., 2018), *blockchain* é:

> "*Blockchain* é uma tecnologia que faz uso de uma arquitetura distribuída e descentralizada para registrar transações de maneira que um registro não possa ser alterado retroativamente, tornando este registro imutável."'

Dois conceitos presentes nessa tecnologia também são os de livro-razão e contratos inteligentes (VADAPALLI, 2020). O livro-razão se trata de um registro compartilhado entre os participantes da rede que armazena as transações concluídas de forma íntegra utilizando o modelo de Árvores de Merkle. Além disso, existem os contratos inteligentes, que são um conjunto de regras que validam as condições e termos para conclusão das transações.

No contexto de uso de criptomoedas, o acesso público às transações pode ser válido e atrativo. Porém para outros propósitos, como o uso dentro de empresas, se torna interessante ter controle dos usuários, das permissões de quem consulta e grava transações.

2.2.2.1 *Consortium Blockchain*

Consortium blockchain trata-se de um conjunto de organizações que formam um *consortium* com permissão para mandar e ler dados relacionados às transações (SHRIVAS; YEBOAH, 2018). Por conta dessa característica, esse modelo viabiliza o uso de *blockchains* por múltiplas organizações de forma conjunta.

O *consortium blockchain* possui algumas características comparadas à rede pública, a exemplo das permissões dos nós presentes na rede, taxas de transferência de transações maiores, escalabilidade menor, melhor proteção contra problemas externos e regras da rede com controle facilitado (DIB et al., 2018).

A infraestrutura CCF é fundamentada nesse modelo, e apresenta algumas melhorias quanto a alguns aspectos. Os nós precisam ser autorizados pelos participantes da rede e são executados nos *Trusted Execution Environments*, garantindo confidencialidade e desempenho da rede (SHAMIS et al., 2019).

2.2.3 *Trusted Execution Environments* (TEE)

Trusted Execution Environments consiste em ambiente de processamento resistente à adulterações que são executados em um *kernel* separado (SABT; ACHEMLAL; BOUABDALLAH, 2015). O que garante autenticidade do código processado, integridade dos tempos de execução e pode servir como prova de confiança para terceiros.

Os TEEs, ou *enclaves* como podem ser referidos, são os ambientes responsáveis por executar os nós da rede criada no CCF. Nas implementações iniciais foram utilizadas as *Intel Software Guard Extensions* (SGX), aumentando confidencialidade e integridade para os dados e códigos carregados(SHAMIS et al., 2019). Pesquisas já mostraram algumas vulnerabilidades que comprometem o funcionamento correto do SGX (NILSSON; BIDEH; BRORSSON, 2020).

3 Proposta de Trabalho

A proposta de trabalho abarca a criação de um novo testador de desempenho para a biblioteca *Confidential Consortium Framework*, que resolva algumas inconsistências que o antigo testador apresentava adicionando novas funcionalidades.

Segundo os desenvolvedores do CCF, o antigo testador é frágil, acoplado e feito para usar um protocolo, desenvolvido por eles e que foi abandonado durante a evolução da biblioteca, que não é mais suportado pelo *framework*. Tendo esses problemas em vista, há a necessidade da criação de um testador mais simples, que tenha suporte para os protocolos *HTTP* e *WebSocket* e outros requisitos que estão abordados na Seção 3.1.1.

A aplicação de teste consistirá na implementação em 3 etapas separadas e bem definidas, que permitirão a execução de testes mais flexíveis e a realização de algumas tarefas de forma *offline*. O resumo da proposta das etapas são:

- **Gerador de requisições:** *script* que realiza a geração de requisições pré-serializados e as salva em um arquivo.
- **Distribuidor de requisições:** será responsável por ler o arquivo com as requisições e mandá-los aos *endpoints* de rede do CCF, calculando salvando os dados obtidos em um arquivo.
- **Analisador:** encarregado de realizar a leitura do arquivo gerado na etapa anterior e disponibilizar os resultados das métricas de análise, junto com gráficos com informações obtidas no decorrer da execução do testador.

O antigo testador tornou-se defasado de acordo com a evolução do *CCF*, e encontra-se em um estado de desuso pelos desenvolvedores. Com um novo testador espera-se que a biblioteca se torne mais útil para os usuários que terão acesso a dados como taxa de resposta, latência e *commits* de transações.

3.1 Aplicação

Será criada uma aplicação de console dividida em 3 partes de acordo com o apresentado na Seção 3. A configuração e execução das rotinas que compõe o testador se darão através do console. A comunicação entre as partes será feita através de arquivos no formato *Apache Parquet*, que oferece armazenamento eficiente e com boa performance de dados formatados em colunas.

Para desenvolvimento do Gerador e do Distribuidor de requisições será usada a linguagem C# com auxílio do *framework* .NET 5, por conta da combinação de bom desempenho, ter suporte multiplataforma e possuir uma grande quantidade de bibliotecas de auxílio, a exemplo da Parquet.Net, que permite a criação e leitura com alta performance de arquivos no formato em questão (SHARMA; MARJIT; BISWAS, 2018).

A última etapa, o Analisador, será implementado com Python com auxílio da ferramenta *pandas*, biblioteca de código aberto que fornece meios flexíveis e fáceis para manipulação e análise de dados (MCKINNEY, 2011). O resultado das análises será mostrado no console, podendo contar com gráficos adicionais que serão salvos como imagens.

Como parte da flexibilidade proposta, a aplicação não será acoplada ao CCF, podendo ser executada independente de pacotes provenientes da biblioteca. Será necessário apenas um *endpoint* da rede disponível para receber e responder as requisições enviadas pelo testador.

3.1.1 Requisitos

Para o desenvolvimento do testador, foram definidos alguns requisitos funcionais e não-funcionais contemplando atualizações, como a adição de funcionalidades em relação ao atual *framework* de testes. Com auxílio dos mantenedores, foi possível obter a lista de requisitos com descrição:

- **Arquitetura flexível:** o novo sistema deverá ser mais simples que o atual, que claramente separa as etapas do teste.
- **Melhor desempenho:** o testador deverá ser capaz de realizar requisições na mesma frequência que a versão anterior, porém de forma a consumir menos recursos do computador.
- **Taxa de envio de requisições alta:** a aplicação deverá ser capaz de enviar milhares de requisições, mais rápido do que a capacidade que o CCF pode processar.
- **Protocolos de requisição:** deverá ser possível enviar requisições para a rede do CCF tanto pelo protocolo *WebSocket* quanto por HTTP.
- **Customização de parâmetros:** o sistema deverá aceitar parâmetros de customização para as requisições, tanto no corpo quanto no cabeçalho e na URL que serão enviados.
- **Calculo de métricas adicionais:** ao final da execução das etapas, a aplicação apresentará as seguintes métricas adicionais:
  - Latência de respostas e *commits*
  - Taxa de transferência intermediária

– Métricas divididas por *endpoints*

– Taxa de transações por segundo

A partir desses requisitos, será possível validar se o testador atende às necessidades dos interessados no projeto. A definição de requisitos também auxiliará na escolha de tecnologias, que serão feitas de forma a se encaixarem nas necessidades de cada parte do sistema.

3.1.2 Arquitetura Proposta

A Figura 1 apresenta a visão geral do sistema. Contando com 3 *scripts* distintos que não se comunicam diretamente, as interfaces entre as partes serão os arquivos *Parquet* que contém informações das requisições durante todo o processo.

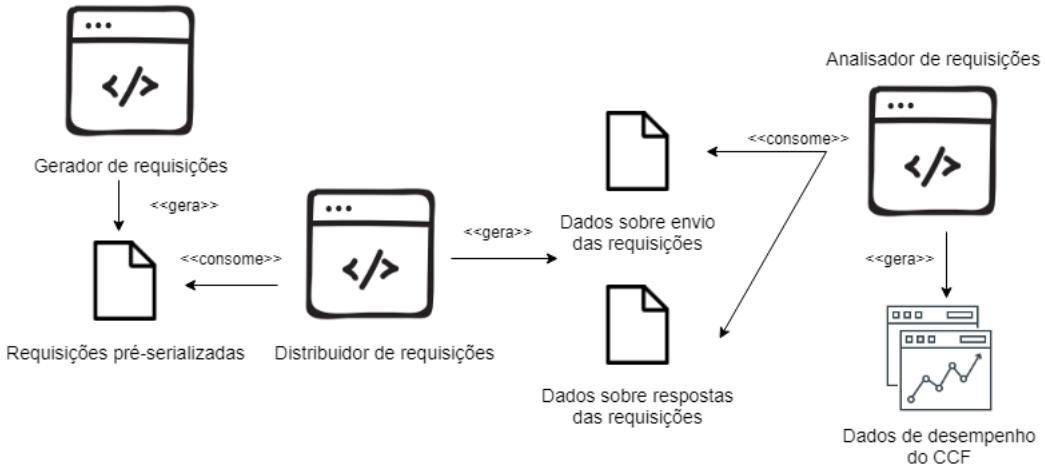

Figura 1 – Diagrama de arquitetura da aplicação

Separando as etapas de teste dessa maneira é possível em caso de necessidade, que outras ferramentas sejam integradas no processo, como uma outra ferramenta que envie requisições ou outro *script* que tenha o papel de analisar os saídas geradas. Por meio do consumo ou geração de arquivos com as mesmas características, flexibiliza-se o uso de novas bibliotecas auxiliares ou outros *scripts* em caso de futura necessidade.

O Gerador de Requisições gera um arquivo que contém as requisições pré-serializadas junto com o identificadores, destinos e conteúdos das mensagens. Esse arquivo é consumido pelo Distribuidor de Requisições e, junto à informação de um *host* especificado, é enviado em forma de requisições *HTTP* via protocolo TCP para o *endpoint* anteriormente indicado. Ao final da execução do Distribuidor de Requisições, são produzidos 2 arquivos:

- O primeiro com dados sobre o envio das requisições, como o identificador e o carimbo de tempo do envio.

- O segundo com dados da resposta recebida pelo servidor, com dados de identificação da mensagem, carimbo de tempo da resposta e a resposta bruta enviada pelo servidor.

Por último, o Analisador lerá esses dois arquivos e assim produzirá indicadores de desempenho com base em cálculos feitos utilizando o número de mensagens enviadas e recebidas, o tempo de envio e resposta junto com o status das mensagens de resposta. Introduzindo o identificador das mensagens em todos os arquivos, será possível rastrear o percusso de uma mensagem desde a sua pré-serialização até o recebimento de sua resposta.

3.1.2.1 Gerador de requisições

O gerador de requisições será o ponto de partida do testador, fornecendo um arquivo contendo uma coluna com os identificadores únicos das mensagens e outra contendo o binário correspondente às requisições pré-serializadas. A Figura 2 mostra o exemplo de arquivo a ser gerado no final dessa etapa.

Message ID	Serialised Request
0	POST /app/log/private HTTP/1.1 Content-type: application/json Content-Length: 52 {"id": 0, "msg": "Some arbitrary message payload 0"}
1	POST /app/log/private HTTP/1.1 Content-type: application/json Content-Length: 52 {"id": 1, "msg": "Some arbitrary message payload 1"}
2	POST /app/log/private HTTP/1.1 Content-type: application/json Content-Length: 52 {"id": 2, "msg": "Some arbitrary message payload 2"}
...	...

Figura 2 – Esquema de colunas das mensagens pré-serializadas - Imagem retirada de <https://github.com/microsoft/CCF/discussions/2324>

Esse *script* será capaz de serializar tanto requisições que utilizam o protocolo *WebSocket* quanto *HTTP*. A personalização dos recursos e métodos das requisições serão importantes para testar diferentes nós do CCF.

Os requisitos para o gerador consistem na escolha de uma linguagem que apresente suporte à bibliotecas que suportem o formato *Apache Parquet* e com bom desempenho. A escolha do *Parquet* se deve à eficiência de armazenamento de dados representados em colunas, que serão de baixa complexidade porém em grande quantidade. O último item é a preparação de requisições *HTTP* utilizando o cabeçalho HTTP/1.1, por conta da falta de suporte do CCF à versões mais novas do protocolo.

Com base nos requisitos e prévia validação junto aos principais mantenedores do CCF, foram escolhidas as seguintes tecnologias:

- C#
- .NET 5 *framework*
- Parquet.NET
- System.CommandLine

Na Seção 3.1, foram apresentados os principais motivos pela escolha da linguagem, *framework* e formato de arquivos. Em adição à aplicação, foi adicionado o pacote *System.CommandLine* que facilitará o tratamento de argumentos no comando de execução do *script*, e ainda fornecerá uma interface de ajuda com os comandos existentes.

3.1.2.2 Distribuidor de requisições

Nesta etapa ocorrerá a principal atividade do teste de desempenho, o envio das requisições para um nó da rede do CCF. O distribuidor de requisições deverá abrir uma única conexão com um nó, carregar o conteúdo obtido no gerador de requisições e então enviar as mensagens pré-serializadas o mais rápido possível para um *endpoint* da rede, respeitando os requisitos.

Serão gerados 2 arquivos também no formato *Parquet*, contendo informações tanto do envio das mensagens quanto do recebimento das respostas. O arquivo referente ao envio deverá conter duas colunas: a primeira com o identificador único das requisições e a segunda com o carimbo de tempo do envio. Na Figura 3 é possível observar o exemplo do esquema dessa tabela.

Message ID	Send Time
0	1615895076.6057985
1	1615895076.6058214
2	1615895076.6058254
...	...

Figura 3 – Esquema de colunas das requisições enviadas - Imagem retirada de <https://github.com/microsoft/CCF/discussions/2324>

As requisições deverão ser enviadas de forma assíncrona, ou seja, o envio deverá prosseguir mesmo que as respostas das mensagens não sejam recebidas. Mantendo apenas uma conexão aberta, serão usadas 2 *threads* para realizar a tarefa: a primeira para mandar as requisições e outra para receber as respostas assim que estiverem disponíveis no *endpoint*.

Dessa forma, será possível potencializar o estresse de um *endpoint* dado um curto espaço de tempo. De acordo com o recebimento das mensagens, suas informações deverão ser salvas em outro arquivo *Parquet*. As colunas desse arquivo conterão identificador único da requisição, carimbo de hora do recebimento e a mensagem de resposta. Na Figura 4, é mostrada uma tabela de exemplo.

Para adicionar opções de uso para o usuário, algumas opções de execução deverão ser implementadas. Em uma execução simples do *script*, esta parte enviará todas as requisições, aguardará pelas respostas e após isso finalizará a execução. As customizações de execução serão:

1. Repetição da lista de requisições N vezes.
2. Repetição da lista de requisições infinitamente até receber um comando de saída.
3. Repetição das requisições até o recebimento de uma resposta igual N vezes.
4. Tempo de espera entre cada envio.
5. *Pipeline* máximo de N requisições por vez.

3.1.2.3 Analisador

O analisador foi implementado utilizando a linguagem *Python* junto à biblioteca de análise de dados *pandas*. Por meio da leitura dos arquivos gerados nas etapas anteriores, foi possível criar um *data frame* contendo as informações agregadas relevantes para o

Message ID	Receive Time	Raw Response
0	1615895077.6057985	HTTP/1.1 200 OK content-length: 4 content-type: application/json x-ms-ccf-transaction-id: 2.17 true
1	1615895077.6058214	HTTP/1.1 200 OK content-length: 4 content-type: application/json x-ms-ccf-transaction-id: 2.18 true
2	1615895077.6058254	HTTP/1.1 200 OK content-length: 4 content-type: application/json x-ms-ccf-transaction-id: 2.19 true
...	...	...

Figura 4 – Esquema de colunas das respostas das requisições recebidas - Imagem retirada de <https://github.com/microsoft/CCF/discussions/2324>

cálculo de métricas que foram definidas com base nos requisitos dos desenvolvedores e em algumas métricas que eram fornecidas pela plataforma de testes anterior.

Por meio de funções fornecidas pelo *pandas*, foram calculadas os seguintes dados após a execução:

- Tempo total de envio de requisições.
- Tempo total de recebimento de requisições.
- Duração total do teste.
- Quantidade de *bytes* enviados/recebidos.
- Tempos de latências (mínimo, máximo e média).
- Quantidade total de requisições e taxa de transferência.
- Taxa de sucesso das requisições.

4 Metodologia

4.1 Metodologia de Desenvolvimento

Para o desenvolvimento da aplicação, foram consideradas 2 metodologias ágeis que se complementam nas fases de planejamento e execução: *Kanban* e *Extreme Programming* (XP). A adaptação dessas metodologias se deram por conta da oportunidade de otimizar as etapas de planejamento e desenvolvimento da aplicação.

O *Kanban* foi escolhido para etapa de planejamento, por conta de algumas possibilidades que ele oferece, como: visualização do fluxo de tarefas, limite de trabalho em progresso e gerenciamento do fluxo de tarefas (AHMAD; MARKKULA; OIVO, 2013). Com essa metodologia será possível transformar as discussões feitas com os mantenedores do CCF em cartões. Além disso, o fluxo para este projeto contará com 4 colunas separando os estágios de desenvolvimento de uma tarefa:

1. *To Do*
2. *Doing*
3. *Validating*
4. *Done*

Na primeira etapa, *To Do*, estarão presentes os cartões extraídos de discussões e conversas realizadas diretamente com os principais desenvolvedores do CCF. De acordo com a ordem de execução das etapas previstas, as tarefas serão movidas para a coluna *Doing*, onde será feito o estudo de tecnologias e desenvolvimento do código. Na raia *Validating* serão colocados os cartões com atividades em processo de avaliação pelos desenvolvedores, que observarão o código, a execução dos *scripts* e o conteúdo saída referentes à tarefa. Por último, os cartões que forem aprovados pelos mantenedores e estiverem de acordo com os requisitos serão movidos para a coluna *Done*.

De acordo com (SOMMERVILLE, 2011), o XP possui diversos princípios ou práticas relacionados com desenvolvimento interativo. Entre as práticas dessa metodologia, serão adotadas algumas para as etapas *Doing* e *Validating* do *Kanban*. Segue as 4 principais práticas a serem usadas no desenvolvimento:

- **Pequenos *Releases*:** serão desenvolvidos produtos viáveis mínimos (MVP) para cada uma das 3 partes do testador.

- **Projeto simples:** em cada tarefa será produzido apenas o necessário para atender as necessidades daquela parte.
- **Refatoração:** se melhorias forem encontradas durante o desenvolvimento, deverão ser realizadas refatorações.
- **Cliente no local:** os principais desenvolvedores auxiliarão na validação dos entregáveis do projeto, de acordo com as necessidades previstas.

5 Procedimentos

Este trabalho terá como referência um cenário de testes que consiste em executar instâncias do testador construído conectado a um nó do *CCF*, a fim de analisar o comportamento do desempenho apresentados por um servidor da biblioteca quando submetido a determinado volume de requisições na fila. Para isolamento de variáveis como a latência de conexão da *internet*, tanto o servidor quanto o testador serão executados em um *cluster* situado na universidade.

Será usada a versão 0.18.5 do *CCF* como base e as instâncias do testador serão executadas em máquinas diferentes dentro do *cluster*. Em caso de o servidor apresentar diminuição na capacidade de processamento das requisições, serão mostrados resultados como o tempo de resposta ao longo do tempo, o uso de recursos por parte do servidor e um gráfico com a latência em relação ao tempo.

A configuração básica de cada máquina são processadores *Intel Core i7-8700* com 16GB de memória RAM. O sistema operacional utilizado é o Ubuntu 20.04.3. A instalação das ferramentas tanto do servidor quanto do testador serão através de *releases* disponíveis nos repositórios correspondentes.

Os cenários de teste serão executados com intervalo de pelo menos 1 minuto para o retorno do servidor CCF para o estado inicial. A monitoração de recursos deverá começar ao menos 10 segundos antes da execução do testador e se encerrar ao menos 5 segundos após. Os dados relacionados a medidas do servidor serão coletados pelo próprio testador, e transformados em métricas e gráficos utilizando-se as bibliotecas *pandas*[1] e *matplotlib*[2]. O monitoramento relativo ao uso de recursos do servidor será possível com a biblioteca de *python psrecord*[3], que será atrelada ao processo correspondente ao nó do CCF em execução e retornará gráficos relacionados ao uso de CPU e memória RAM daquela instância.

[1] Disponível em: https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html
[2] Disponível em: https://matplotlib.org/
[3] Disponível em: https://github.com/astrofrog/psrecord

6 Resultados e Discussões

6.1 Cenário de Controle

O primeiro cenário que será usado para comparações posteriores foi definido como o envio de requisições com as seguintes características:

- Envio de 50 mil requisições por bateria
- Método *POST*
- Mensagens padrões sequenciais
- Rota `/app/log/private`

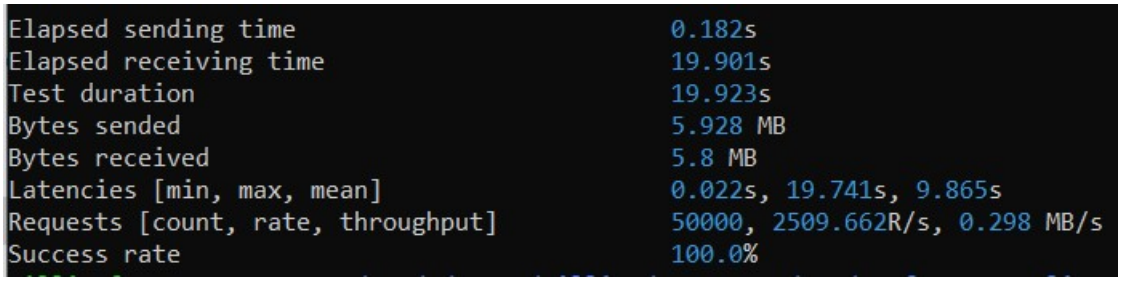

Figura 5 – Resultados (Cenário Controle)

Após a execução, notaram-se algumas informações relevantes que serão importantes para efeito de análise, entre elas estão o tempo de envio em 0,182 segundos, a taxa de transferência de 0,298 *Megabytes* por segundo e a taxa de sucesso em 100%. Uma questão a ser observada detalhadamente será o aumento da latência de resposta do servidor ao longo do tempo, onde se inicia em 0,02 segundos e tem sua máxima em 19,741 segundos.

Como é observado na Figura 6, a distribuição das latências é praticamente uniformemente ao longo dos tempos entre 0s e 20s que é o intervalo de duração dos testes, e também, é possível notar que há um crescimento constante no tempo de processamento do servidor. Mesmo com todas requisições recebidas pelo servidor em centésimos de segundos, o tempo de resposta cresce ao decorrer do teste. Isso é devido ao fato de o servidor operar de modo monoprocessado, enfileirando todas as requisições e atendendo-as de maneira sequencial. A taxa média atendimento é de 2510 requisições por segundo.

Na Figura 7, que contém a monitoração de recursos da instância do nó a ser utilizado, observa-se um aumento considerável de uso da CPU no momento em que as

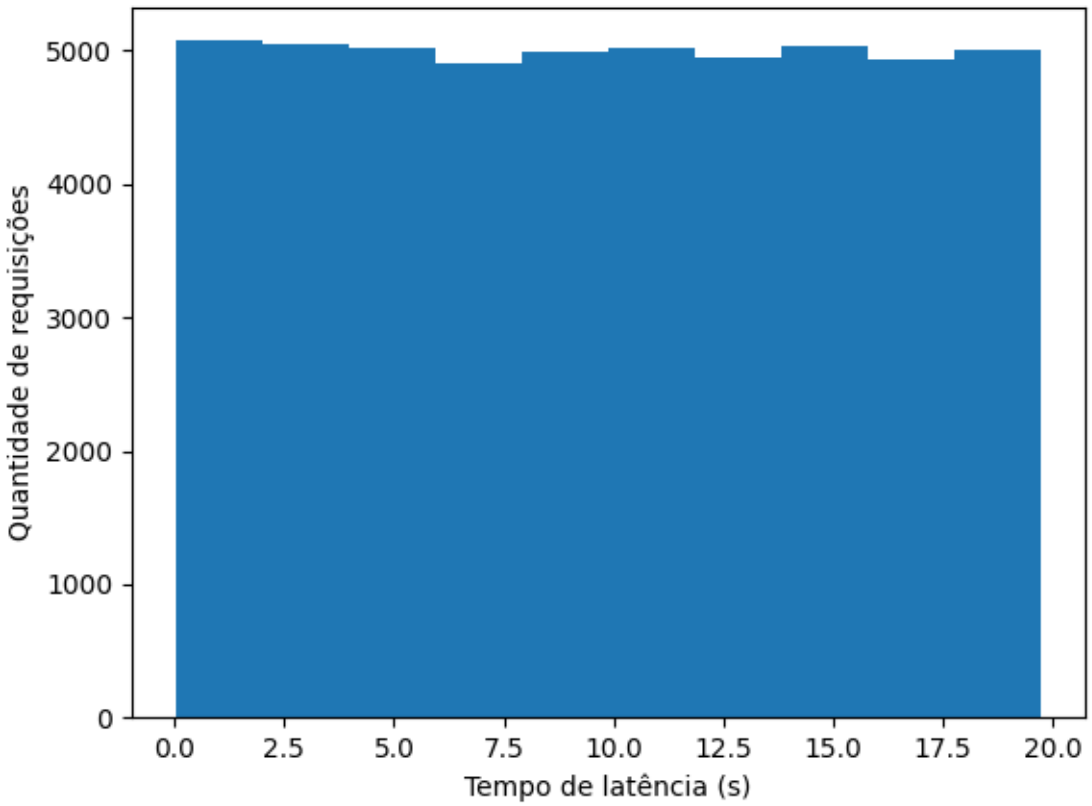

Figura 6 – Histograma de latências(Cenário Controle)

requisições começam a ser respondidas, um pouco antes dos 20 segundos de monitoração. Durante o período de teste, a porcentagem de utilização da CPU sobe de 20% para 100% e apenas retorna ao estado inicial quando os arquivos de resultados são salvos em disco. O uso de memória *RAM* apresenta um discreto aumento nos segundos finais de teste e permanece dessa maneira até o final da monitoração.

6.2 Cenário com Mensagens Customizadas

Este cenário é semelhante ao cenário de controle na maioria dos aspectos, nota-se a diferença na mensagem que é enviada para o servidor: em vez de mensagens sequenciais com o *ID* da requisição, são enviadas mensagens com *strings* contendo Identificador Globais Únicos(*GUID*):

- Envio de 50 mil requisições por bateria
- Método *POST*
- Mensagens únicas contendo *GUIDs*

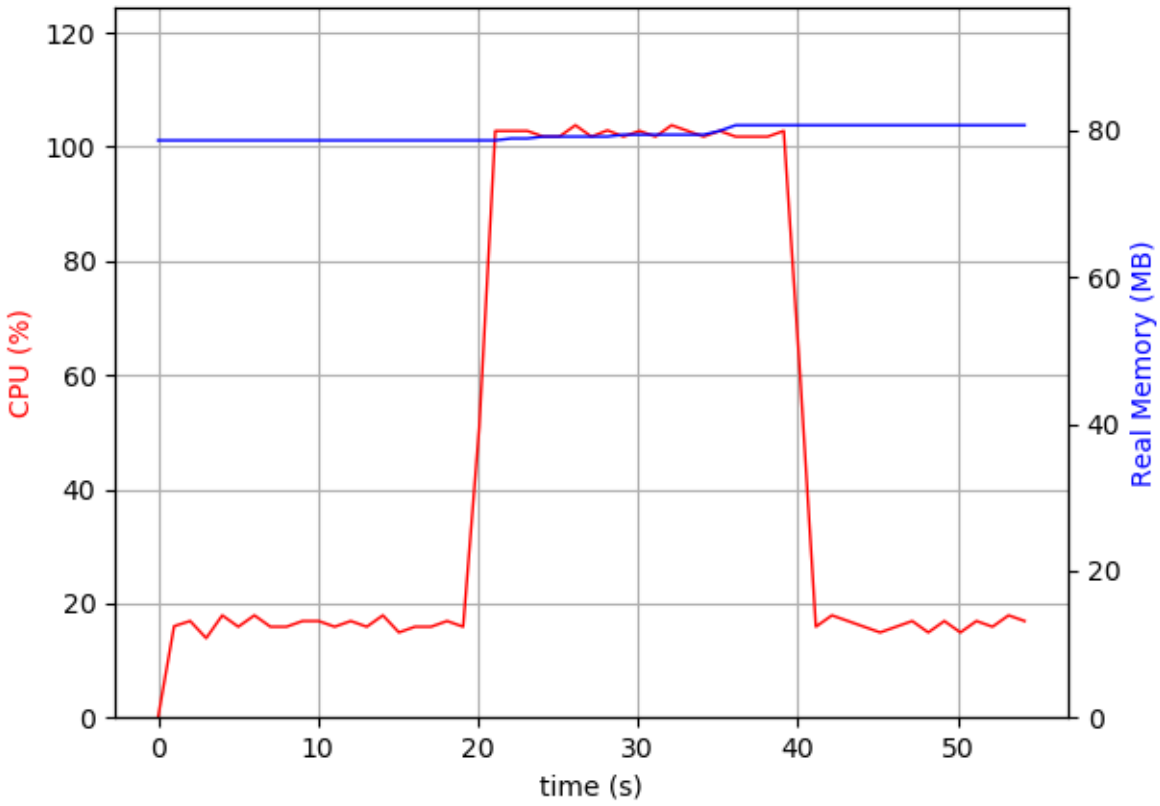

Figura 7 – Uso de CPU e RAM(Cenário Controle)

- Rota `/app/log/private`

```
Elapsed sending time                    0.198s
Elapsed receiving time                  20.618s
Test duration                           20.658s
Bytes sended                            7.089 MB
Bytes received                          5.8 MB
Latencies [min, max, mean]              0.04s, 20.46s, 10.257s
Requests [count, rate, throughput]      50000, 2420.37R/s, 0.343 MB/s
Success rate                            100.0%
```

Figura 8 – Resultados (Cenário com Mensagens Customizadas)

Este cenário se assimila bastante ao primeiro cenário apresentado, com pequenas diferenças notadas nos tempos de recebimento e por consequência nas latências. Ao se trocar o modelo de mensagem enviada, notou-se um aumento dos *bytes* totais enviados em cerca de 1 *Megabyte*, o que pode ter influenciado nos tempos ligeiramente maiores deste cenário.

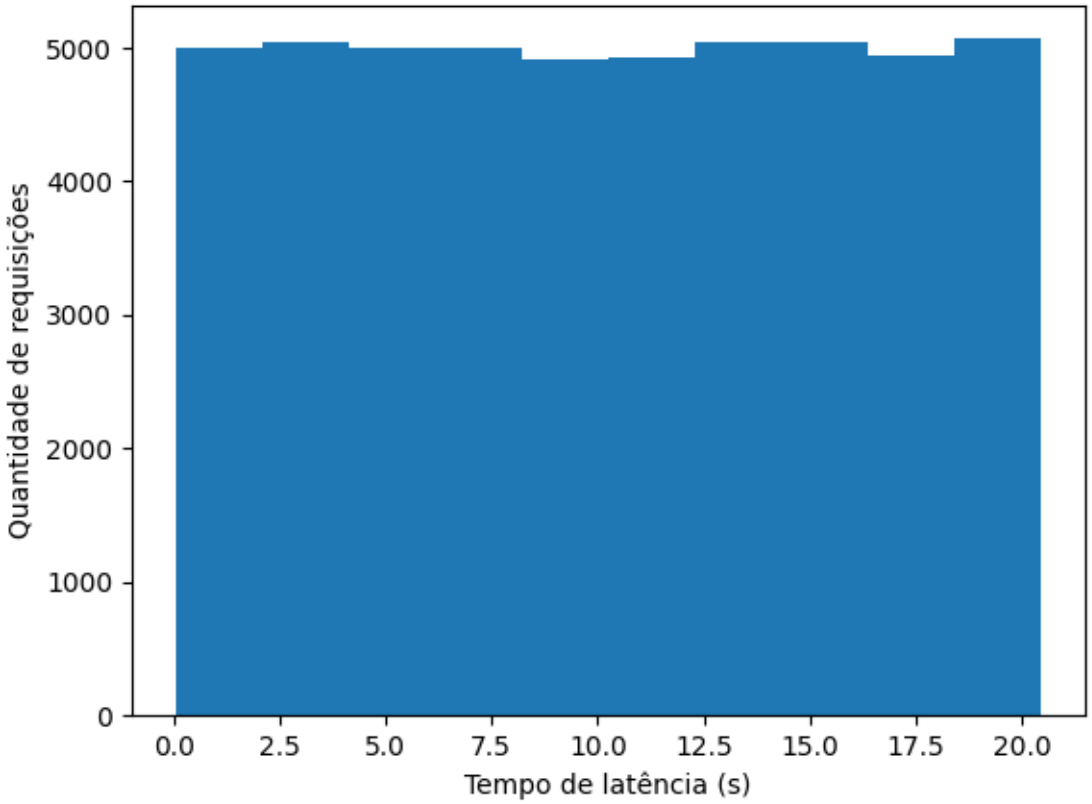

Figura 9 – Histograma de latências(Cenário Mensagem Customizada)

A distribuição vista na Figura 9 indica para um tempo de processamento e resposta dos servidores semelhante, seguindo um crescimento ao longo do teste.

Quanto à monitoração de recursos, notou-se uma diferença, porém não representativa, quanto ao cenário anterior: o uso de memória RAM. O maior uso de memória começou em um momento anterior, cerca de 10 segundos após o início de envio de respostas, e o total de memória utilizada também apresentou um valor maior. O uso de CPU foi similar ao caso de controle, como pode ser visto na Figura 10. Seu início se deu em 20% em momentos antes do início do teste e subiu para 100% no período de duração.

6.3 Cenário com Método *GET*

Este cenário tem como principal característica o método de requisição diferente com outra rota, onde é necessário menor processamento comparado ao cenário de controle. Quanto ao número de *bytes* enviados, notou-se uma quantidade menor, cerca de 0.3 *Megabytes* a menos e em relação aos *bytes* recebidos houve um crescimento de 1 *Megabyte*

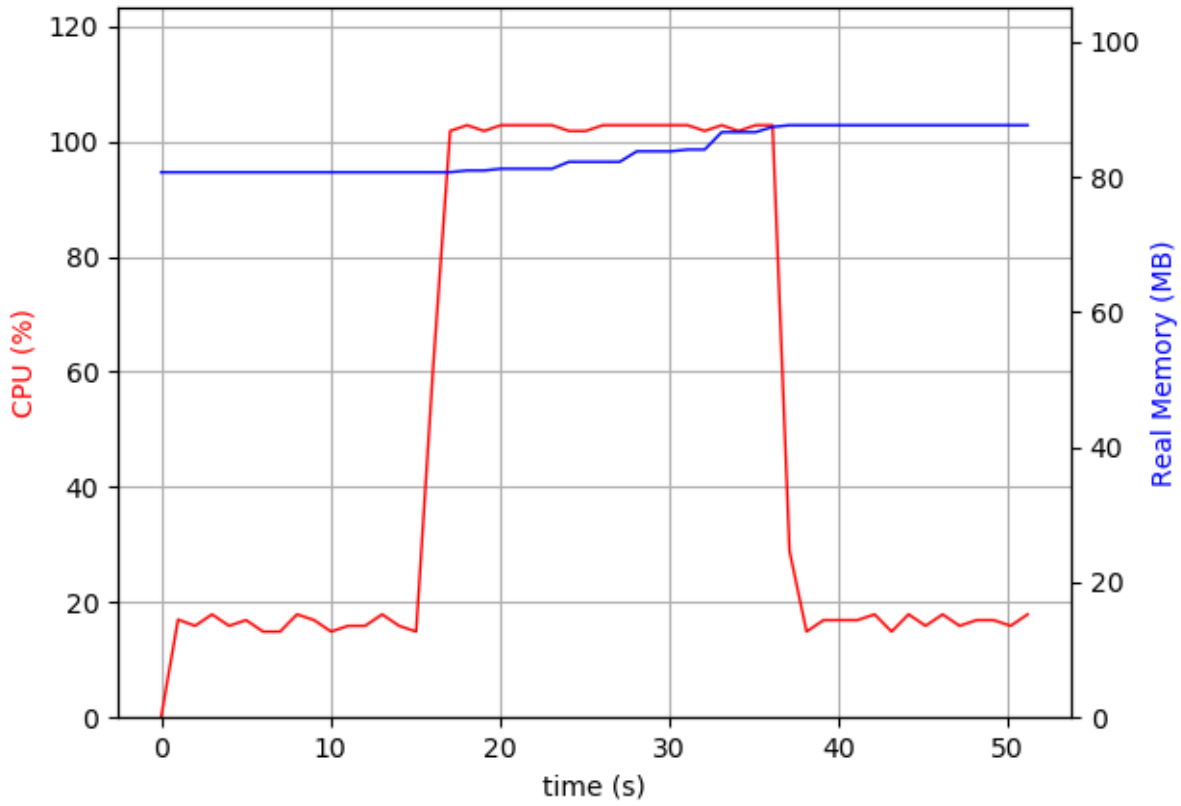

Figura 10 – Uso de CPU e RAM(Cenário Mensagem Customizada)

- Envio de 50 mil requisições por bateria
- Método *GET*
- Rota `/app/commit`

```
Elapsed sending time                    0.169s
Elapsed receiving time                  1.489s
Test duration                           1.493s
Bytes sended                            5.628 MB
Bytes received                          6.9 MB
Latencies [min, max, mean]              0.004s, 1.324s, 0.672s
Requests [count, rate, throughput]      50000, 33489.618R/s, 3.769 MB/s
Success rate                            100.0%
```

Figura 11 – Resultados (Cenário Método *GET*)

Notou-se neste cenário uma diferença maior quanto nos resultados obtidos. Os tempos de envio foram similares, porém o tempo de recebimento foi consideravelmente menor, representando cerca de 7,5% comparado ao tempo de recebimento do cenário

controle. Devido ao menor tempo de recebimento, outras métricas apresentaram mudanças significantes, como: taxa de requisições de 33.490 requisições por segundo e a taxa de transferência de 3.769 *Megabytes* por segundo.

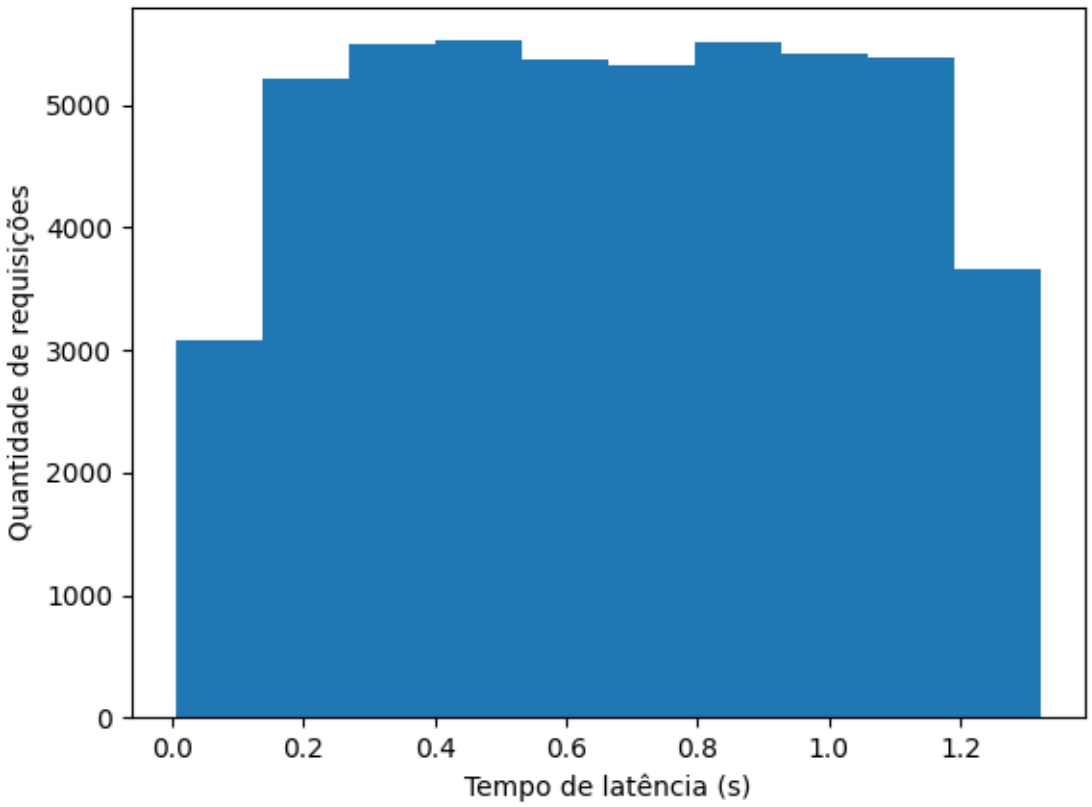

Figura 12 – Histograma de latências(Cenário Método *GET*)

A Figura 12 apresenta um padrão diferente aos anteriores: menos requisições estão distribuídas no primeiro e no último setor do histograma. Uma fator que afetou este histograma foram os tempos iniciais de latência mais baixos, que chegam a ser em milésimos de segundo.

Em relação ao uso de recursos do computador, este cenário apresentou também uma mudança em relação aos cenários anteriores. O uso de memória *RAM* foi praticamente constante, e não apresentou grandes mudanças mesmo com a execução do testador. Quanto ao uso de CPU, o gráfico apresentou um pico, devido à curta duração do teste, e que atingiu um valor maior do que os cenários anteriores: cerca de dobro do valor excedente ao chegar em 100%.

A Figura 14 mostra o crescimento praticamente contínuo com pequenos ruídos em pontos próximos aos de 0,3 e 1,1 segundos, o que pode ser confirmado com as lacunas

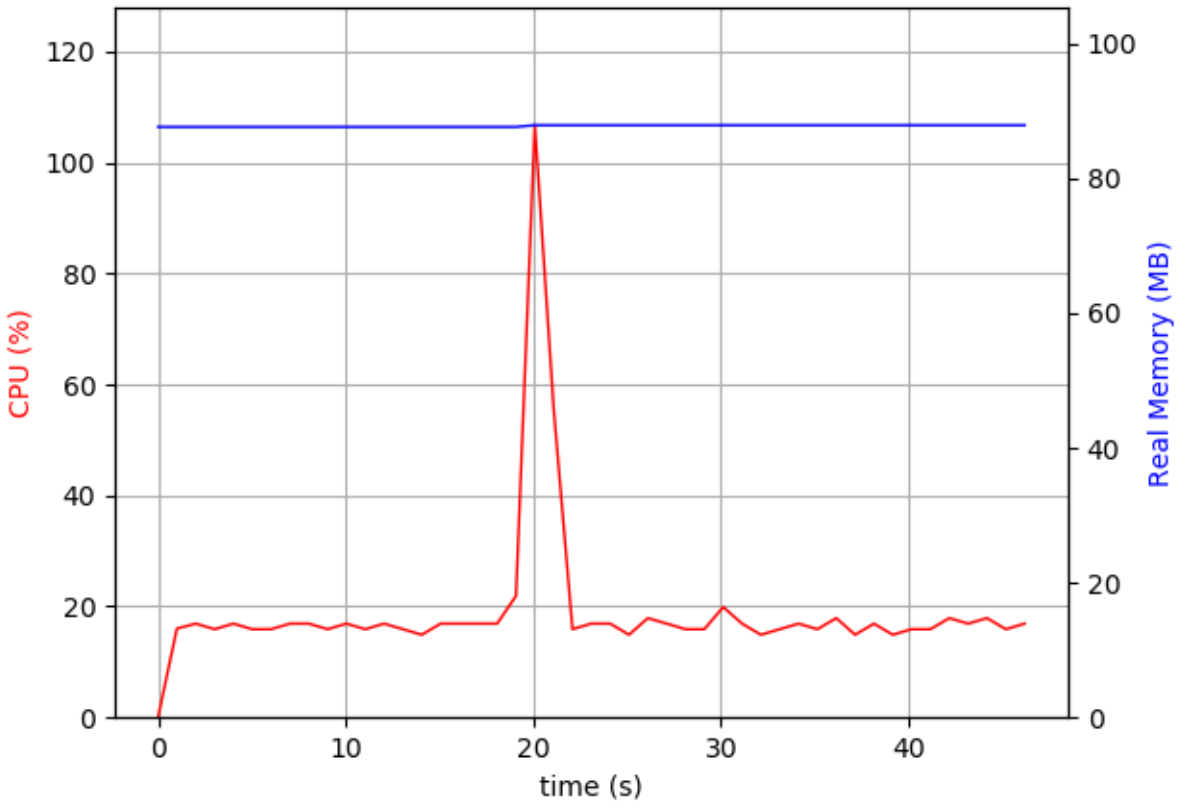

Figura 13 – Uso de CPU e RAM(Cenário Método *GET*)

observadas no histograma anterior. Devido ao grande número de requisições exibidos neste gráfico, os ruídos são praticamente imperceptíveis ao longo da progressão do teste.

6.4 Cenário com mensagens maiores

A grande diferença desse cenário para o de controle se dá no corpo das mensagens enviadas, onde enquanto no primeiro caso as mensagens continham até 15 caracteres, este cenário tem o tamanho de suas mensagens de 300 caracteres alfanuméricos randômicos.

- Envio de 50 mil requisições por bateria
- Método *POST*
- Mensagens com caracteres randômicos com mais caracteres
- Rota `/app/log/private`

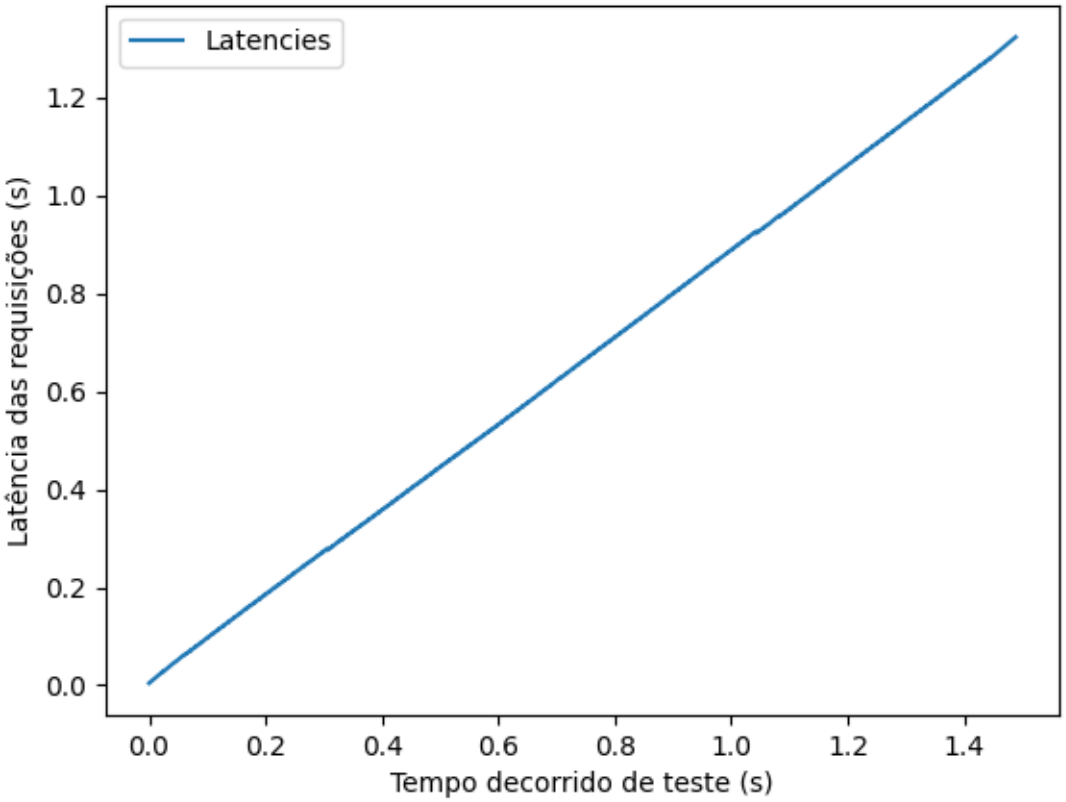

Figura 14 – Latências ao decorrer do teste(Cenário Método *GET*)

```
Elapsed sending time                    1.812s
Elapsed receiving time                  21.083s
Test duration                           21.127s
Bytes sended                            20.339 MB
Bytes received                          5.739 MB
Latencies [min, max, mean]              0.044s, 19.315s, 9.598s
Requests [count, rate, throughput]      50000, 2366.64R/s, 0.963 MB/s
Success rate                            100.0%
```

Figura 15 – Resultados (Cenário com mensagens maiores)

Neste cenário, por conta do maior tamanho de bytes enviados houve um acréscimo no tempo de envio das requisições comparando com os outros cenários, porém a latência não sofreu grandes alterações comparada ao cenário de controle. A taxa de transferência foi a maior observada entre todos os casos vistos neste trabalho.

O histograma seguiu o padrão da maioria dos casos, uma distribuição quase uniforme com um leve decréscimo ao longo do teste. O crescimento do tempo de resposta por parte do servidor pode estar relacionado ao enfileiramento visto anteriormente.

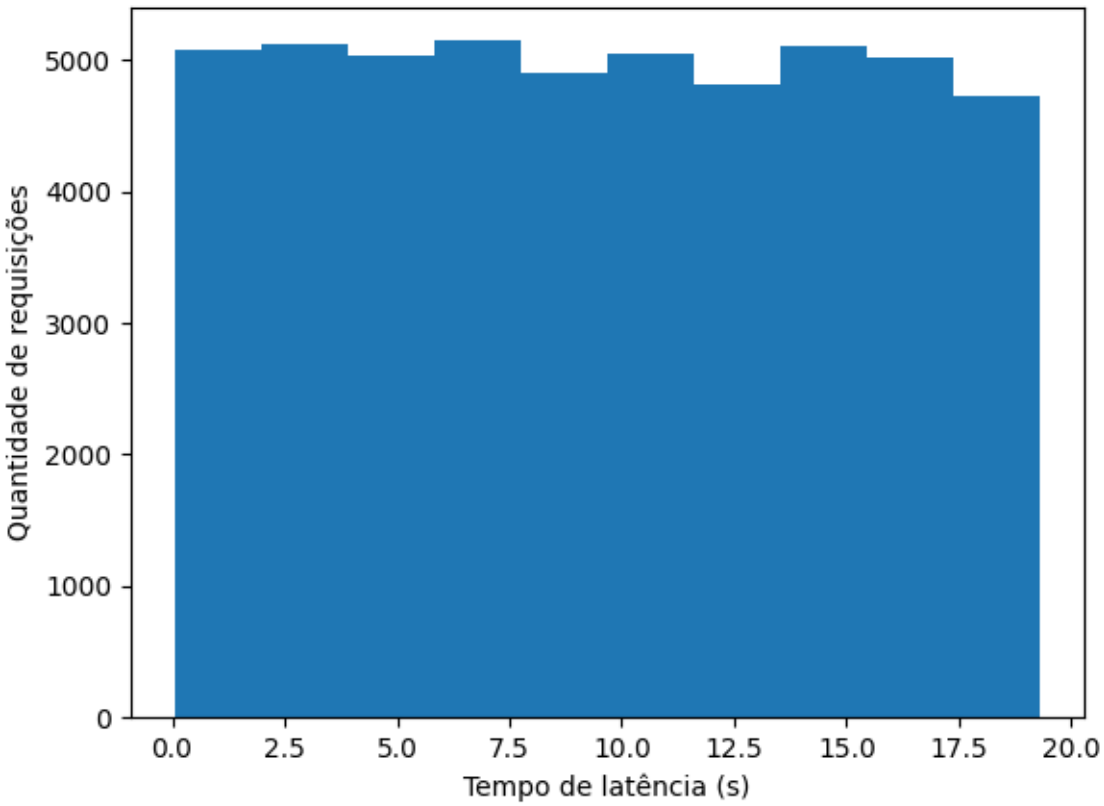

Figura 16 – Histograma de latências (Cenário com mensagens maiores)

Quanto ao uso de recursos, observamos algumas mudanças em relação ao cenário de controle. No caso do uso de CPU, o comportamento é similar ao cenário onde foram enviadas 200 mil requisições: um pico para até 120% nos primeiros segundos e a estabilização em 100% até o término da execução. Quanto ao uso da memória RAM, houve um crescimento considerável onde no momento inicial estava no valor de 20 *Megabytes* e ao final da execução aferiu-se o uso de 150 *Megabytes*, um crescimento absoluto comparável ao caso onde se enviaram mais requisições.

6.5 Cenário com mais requisições

Este cenário possui características muito próximas ao cenário de controle com exceção do número total de requisições. Deverão ser analisadas o quádruplo de requisições totais em comparação ao primeiro teste.

- Envio de 200 mil requisições por bateria
- Método *POST*

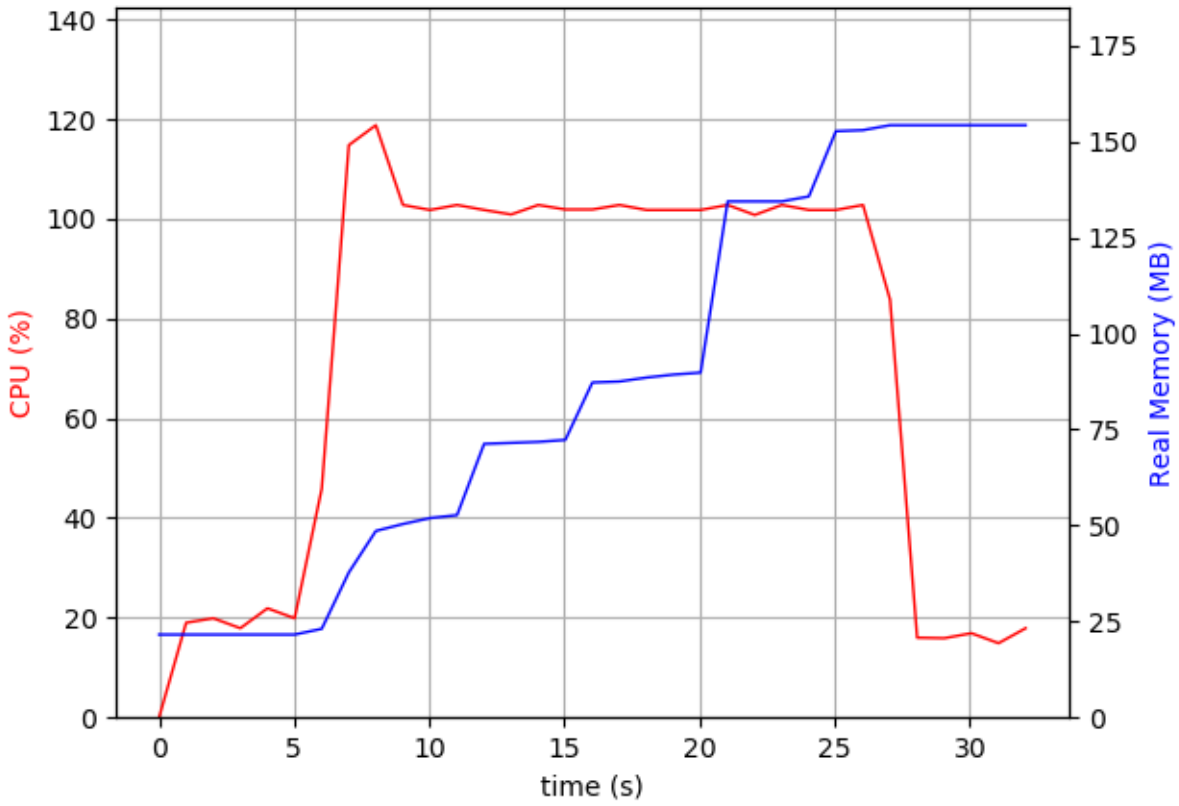

Figura 17 – Uso de CPU e RAM(Cenário com mensagens maiores)

- Mensagens sequenciais
- Rota `/app/log/private`

```
Elapsed sending time                    0.592s
Elapsed receiving time                  81.001s
Test duration                           81.042s
Bytes sended                            23.978 MB
Bytes received                          23.2 MB
Latencies [min, max, mean]              0.041s, 80.45s, 40.076s
Requests [count, rate, throughput]      200000, 2467.856R/s, 0.296 MB/s
Success rate                            100.0%
```

Figura 18 – Resultados (Cenário com mais requisições)

As métricas deste cenário se mostraram muito próximas à proporção esperada. Com exceção do tempo de envio de 0,592 segundos, dados como: tempo de recebimento igual a 81 segundos, valor de latência máximo em 80 segundos e total de *bytes* enviados e recebidos de 23 *Megabytes* apresentaram a proporção próxima de 4 vezes em relação

ao valor encontrado no cenário de controle. Os valores das taxas de transferências e de requisições foram ligeiramente diferentes: 0,296 *Megabytes* por segundo contra 0,298 e 2467 requisições por segundo contra 2509 respectivamente.

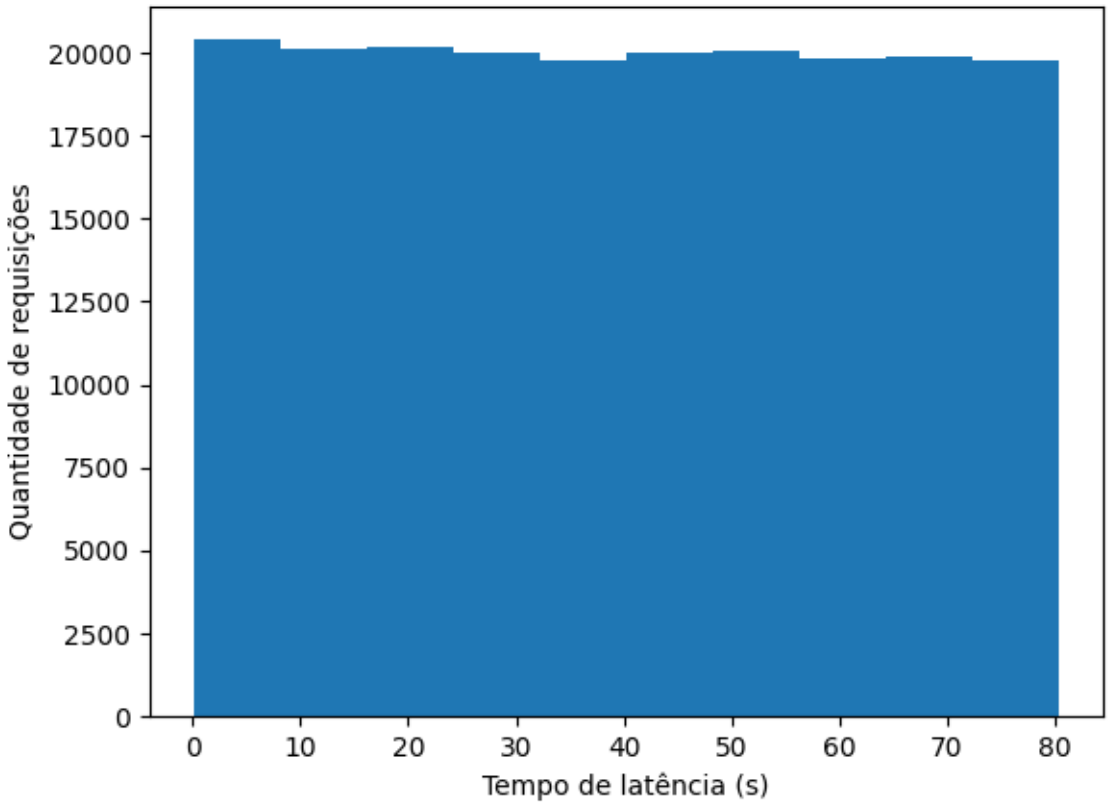

Figura 19 – Histograma de latências (Cenário com mais requisições)

O histograma da Figura 19 apresenta a distribuição similar aos das Seções 6.1 e 6.2, tendo suas requisições bem distribuídas em intervalos proporcionais. Com base neste histograma e nos dados obtidos, é possível perceber a similaridade no crescimento do tempo de latência, em um valor aproximadamente constante.

A Figura 20 com os dados de recursos utilizados durante esse teste apresenta algumas peculiaridades em relação aos cenários anteriores. Em relação ao uso de CPU, o formato do gráfico é similar aos cenários que usaram o método *POST*, com exceção do início que apresenta um pico de solicitação da CPU similar ao da Seção 6.3 apresentou. Porém, o uso máximo de CPU durante o teste chegou a um valor maior de 120%, mais de 13% em relação ao máximos observados anteriormente. O uso de memória *RAM* apresentou aumentos mais significativos em determinados pontos: após o servidor receber as requisições, próximo à metade do teste e cerca de 5 segundos antes do término. Esses

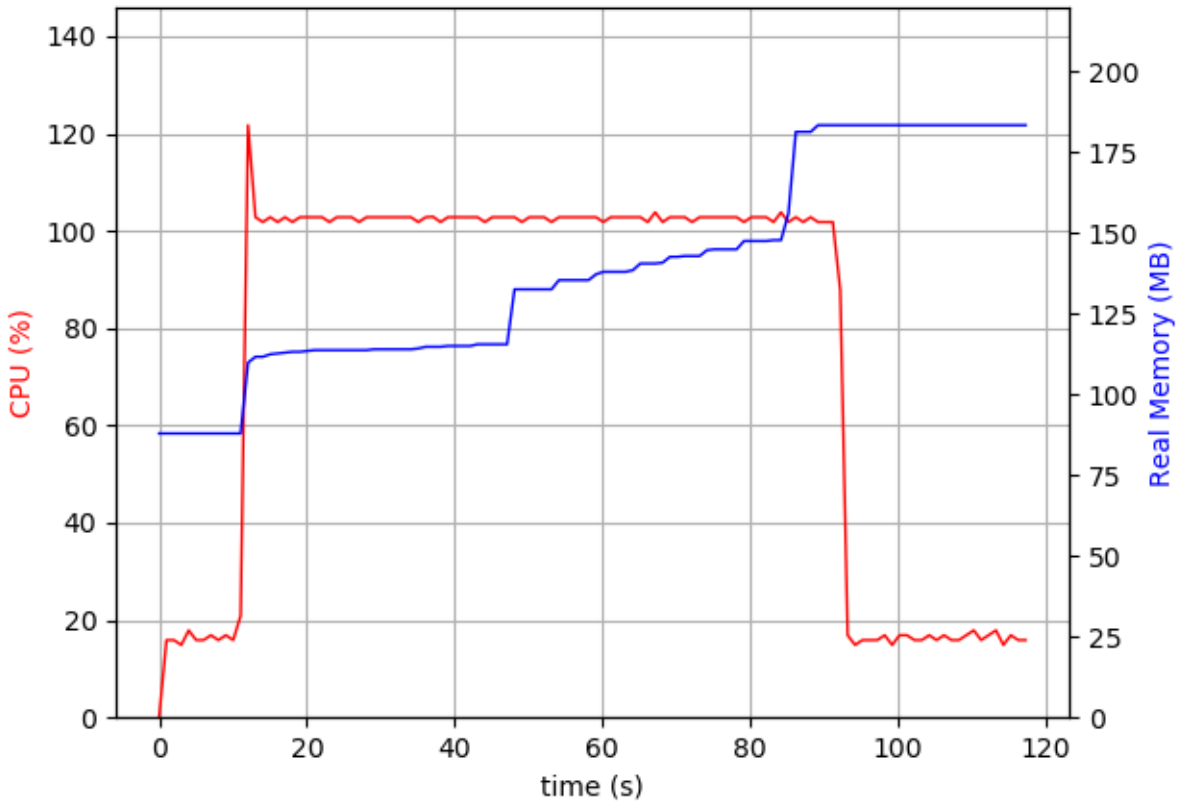

Figura 20 – Uso de CPU e RAM(Cenário com mais requisições)

aumentos foram em maiores proporções, e ao término do teste o uso de memória *RAM* era de mais de 2 vezes o valor inicial, cerca de 180 *Megabytes*.

Um possível motivo para esse problema é proveniente da arquitetura monoprocessada de requisições, que enfileiram as requisições recebidas e não manda respostas paralelamente. O envio de grandes quantidades de requisições deste cenário seguida do enfileiramento das mesmas, é um motivo pertinente para o uso de memória *RAM* exagerado comparado aos outros casos, já que o servidor precisa guardar em memória a grande quantidade de requisições que serão respondidas em seguida.

Como explicação, os desenvolvedores da biblioteca sinalizaram que o motivo de um nó ser monoprocessado é a necessidade de concentrar as gravações e leituras de uma única conexão em uma única *thread*.

7 Conclusão

Por meio deste trabalho, foi possível fazer a análise de alguns aspectos de desempenhos relacionados à biblioteca *Confidential Consortium Framework*. A partir da construção de um *software* que supriu requisitos levantados pelos desenvolvedores e uso de tecnologias acessíveis, tornou-se viável a execução de baterias de testes com parâmetros customizáveis e que poderá ser integrada na *pipeline* de testes do *CCF*.

Após a definição da arquitetura a partir de um tópico criado pelos desenvolvedores em seu repositório, foram escolhidas as tecnologias de acordo com as necessidades das 3 partes pré-definidas. Nas duas primeiras partes usou-se o *framework* .NET 5 com C# 9 por conta da necessidade de velocidade de criação das requisições pré-serializadas, envio e recebimento. Para o Analisador, optou-se pelo *Python* junto à alguns módulos que facilitaram a geração de gráficos e cálculo de métricas na questão de análise de dados.

Para obtenção de avaliações de desempenho de um servidor do *CCF*, instalou-se a aplicação de testes disponibilizada pelos desenvolvedores da biblioteca e a última versão do testador desenvolvido em um *cluster* disponibilizado pela Universidade de Brasília, a fim de evitar que possíveis instabilidades de rede e uso de recursos por aplicações não analisadas afetassem os dados obtidos após testes. O servidor e o testador foram executados em máquinas distintas para que o processamento das requisições não fosse afetado pela execução do testador.

Como observado no Capítulo 6, foram definidos casos de testes para avaliar como mudanças nas requisições poderiam afetar o desempenho de um nó da aplicação do *CCF*. O padrão notado foi o tempo de resposta praticamente em crescimento constante, que nos histogramas feitos para todos os casos apresentou uma pequena variação por conta do curto de espaço de testes. Com auxílio dos gráficos de recursos obtidos junto com os tempos de latência, nota-se o comportamento de enfileiramento de requisições por parte do nó, o que se deve ao uso de apenas uma *thread* para o processamento das respostas.

Como trabalhos futuros, sugere-se a transformação de um nó do CCF para uma arquitetura de multiprocessos a fim de evitar o enfileiramento como visto neste trabalho, e tornar mais rápido o retorno de respostas. Quanto ao testador, adicionar uma funcionalidade para efetivar as mudanças feitas e melhorar a verificação dos certificados de entrada seriam úteis para um futuro uso na esteira de testes oficial da biblioteca. Além desses trabalhos relacionados a desenvolvimento, *benchmarkings* comparando a outras bibliotecas similares ou com o desempenho do CCF com um banco de dados relacionais pode ser elaborado para a apresentação do projeto.

Referências

AHMAD, M. O.; MARKKULA, J.; OIVO, M. Kanban in software development: A systematic literature review. In: . [S.l.: s.n.], 2013. p. 9–16. Citado na página 37.

ALVES, P. H. et al. Desmistificando blockchain: Conceitos e aplicações. In: _____. [S.l.: s.n.], 2018. p. 1–24. Citado na página 27.

AMMANN, P.; OFFUTT, J. In: *Introduction to Software Testing*. [S.l.: s.n.], 2008. v. 1. Citado na página 25.

CECCHETTI, E. et al. Solidus: Confidential distributed ledger transactions via pvorm. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2017. (CCS '17), p. 701–717. ISBN 9781450349468. Disponível em: <https://doi.org/10.1145/3133956.3134010>. Citado na página 27.

DIB, O. et al. Consortium blockchains: Overview, applications and challenges. 09 2018. Citado 3 vezes nas páginas 21, 26 e 28.

Evans, D.; Kolesnikov, V.; Rosulek, M. [S.l.: s.n.], 2018. Citado na página 27.

MAHAJAN, P.; SHEDGE, H.; PATKAR, U. Automation testing in software organization. *International Journal of Computer Applications Technology and Research*, v. 5, p. 198–201, 04 2016. Citado na página 25.

MCKINNEY, W. pandas: a foundational python library for data analysis and statistics. *Python High Performance Science Computer*, 2011. Citado na página 30.

MYERS, G.; BADGETT, T.; SANDLER, C. Module (unit) testing. In: _____. *The Art of Software Testing*. John Wiley Sons, Ltd, 2012. cap. 5, p. 85–111. ISBN 9781119202486. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119202486.ch5>. Citado na página 26.

NILSSON, A.; BIDEH, P. N.; BRORSSON, J. A survey of published attacks on intel sgx. 2020. Citado na página 28.

SABT, M.; ACHEMLAL, M.; BOUABDALLAH, A. Trusted execution environment: What it is, and what it is not. *2015 IEEE Trustcom/BigDataSE/ISPA*, v. 1, p. 57–64, 2015. Citado na página 28.

SHAMIS, A. et al. Ccf: A framework for building confidential verifiable replicated services. In: . [S.l.: s.n.], 2019. Citado 3 vezes nas páginas 26, 27 e 28.

SHARMA, K.; MARJIT, U.; BISWAS, U. Efficiently processing and storing library linked data using apache spark and parquet. *Information Technology and Libraries*, v. 37, 2018. Citado na página 30.

SHRIVAS, M.; YEBOAH, D. The disruptive blockchain: Types, platforms and applications. In: . [S.l.: s.n.], 2018. Citado na página 28.

SOMMERVILLE, I. In: *Engenharia de Software*. [S.l.: s.n.], 2011. v. 9. Citado 3 vezes nas páginas 25, 26 e 37.

VADAPALLI, R. *BLOCKCHAIN FUNDAMENTALS TEXT BOOK Fundamentals of Blockchain*. [S.l.: s.n.], 2020. ISBN 301.345.908. Citado na página 28.

YAO, A. Protocols for secure computations. *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, p. 160–164, 1982. Citado na página 27.