UNIVERSIDADE FEEVALE

DIOVANE BARBIERI GABRIEL

TRILHAS DE AUDITORIA COM BLOCKCHAIN: UMA ANÁLISE DE DESEMPENHO EM ALGORITMOS TRANSACIONAIS

Novo Hamburgo

2021

DIOVANE BARBIERI GABRIEL

TRILHAS DE AUDITORIA COM BLOCKCHAIN: UMA ANÁLISE DE DESEMPENHO EM ALGORITMOS TRANSACIONAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Feevale

Orientador: Prof. Dra. Adriana Neves dos Reis

Novo Hamburgo

2021

DIOVANE BARBIERI GABRIEL

TRILHAS DE AUDITORIA COM BLOCKCHAIN: UMA ANÁLISE DE DESEMPENHO EM ALGORITMOS TRANSACIONAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Feevale

APROVADO EM: ___ / ___ / _____

PROF. DRA. ADRIANA NEVES DOS REIS
Orientador – Feevale

PROF. DRA. MARTA ROSECLER BEZ
Examinador interno – Feevale

PROF. DR. RICARDO FERREIRA DE OLIVEIRA
Examinador interno – Feevale

Novo Hamburgo
2021

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, minha esposa, minha família, amigos e às pessoas que convivem comigo diariamente, minha gratidão, pelo apoio, e em especial a minha avó materna, Jeni, que desde 2019 não está mais entre nós, mas que esteve ao meu lado em todos os momentos importantes da minha vida.

RESUMO

A informação tem se tornado cada vez mais um recurso valioso dentro das organizações, e para proteger e garantir a sua integridade, conceitos antigos de trilhas de auditoria têm sido aprimorados, e novas técnicas de segurança estão sendo criadas à medida que novas tecnologias vão surgindo. Pensando nisso, autores da atualidade criaram novas abordagens aplicando a crescente tecnologia do *blockchain* que permite que informações sejam armazenadas em seu interior de forma que não possam mais ser alteradas. Desta forma, é possível utilizar o conceito convencional de trilha de auditoria a partir da geração de *logs* e gravá-los no interior do *blockchain*, ganhando a propriedade da imutabilidade e, concomitantemente, separando as informações do dispositivo titular do sistema, pois um registro de *log* armazenado no mesmo equipamento é igualmente vulnerável. Com isso, este trabalho traz uma análise de desempenho em diferentes algoritmos alternativos de transação de informações de trilhas de auditoria com o *blockchain*, com experimentos comparativos relacionados à forma de envio dos *logs* à rede *blockchain*, juntamente com a análise do desempenho, através da medição do tempo necessário para cada transação, em diferentes cenários propostos, incluindo o método colocado por Kalis e Belloum (2018). Por fim, como parte dos resultados, este trabalho faz recomendações de uso do algoritmo assíncrono com fila, que de acordo com os experimentos, demonstrou os melhores resultados de acordo com as análises de média de desempenho geral e padronização na duração do tempo das transações.

Palavras-chave: *Blockchain*. Desempenho. *Log*. Trilha de auditoria

ABSTRACT

Information has become an increasingly valuable resource within organizations, and to protect and ensure its integrity, old concepts of audit trails have been improved, and new security techniques are being created as new technologies emerge. With that in mind, today's authors have come up with new approaches by applying the growing blockchain technology that allows information to be stored inside so that it can no longer be altered. In this way, it is possible to use the conventional concept of audit trail from the generation of logs and record them inside the blockchain, gaining the property of immutability and, at the same time, separating the information from the system's owner device, as a record of the log stored on the same equipment is equally vulnerable. Thus, this work presents a performance analysis in different alternative algorithms for the transaction of audit trail information with the blockchain, with comparative experiments related to how logs are sent to the blockchain network, along with performance analysis, through measurement of the time required for each transaction, in different proposed scenarios, including the method placed by Kalis e Belloum (2018). Finally, as part of the results, this work makes recommendations for the use of the asynchronous algorithm with a queue, which, according to the experiments, showed the best results according to the analysis of general performance average and standardization in transaction time duration.

Keywords: Audit Trail. Blockchain. Log. Performance

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

APEEER	Aplicação Para Execução de Experimentos e Extração de Resultados
EVM	*Ethereum Virtual Machine*
EOA	*External Owned Account*
PoS	*Proof-to-Stake*
PoW	*Proof-to-Work*
SGBD	Sistema de Gerenciamento de Banco de Dados

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais a informação torna-se um recurso crítico dentro das organizações. O crescente número de dispositivos e computadores conectando todos os tipos de usuários por todo mundo, faz com que progressivamente mais dados sejam processados em menores períodos de tempo. Para ser capaz de entender o que implica a segurança de um computador, é necessário compreender os termos vulnerabilidade e risco, segundo Bosworth e Kabay (2002).

Bosworth e Kabay (2002) explicam que a vulnerabilidade é um ponto fraco em sistema computacional ou nos meios que o cercam que pode comprometer a segurança. Um risco, dentro da computação, é a probabilidade de um evento ocasionar uma perda de qualquer natureza, como por exemplo, perdas financeiras e de pessoal, perda de reputação e clientes, incapacidade de operar normalmente ou completamente durante um período de tempo, incapacidade de crescimento, e violação de leis e regras governamentais.

Em casos onde a propriedade do dispositivo é diferente da propriedade dos segredos contidos dentro do mesmo, é essencial que existam formas e mecanismos de auditoria que possam identificar e determinar qualquer tentativa de fraude (SCHNEIER; KELSEY, 1999).

Bosworth e Kabay (2002) seguem dizendo, que precauções especiais devem ser tomadas a fim de validar a integridade e assegurar que dados críticos sejam devidamente protegidos. A principal medida a ser tomada para evitar esse tipo de perda é a auditoria, cujo papel inclui:

- Recomendar iniciativas que minimizem, ou preferencialmente, eliminem vulnerabilidade e riscos;
- Garantir que os devidos controles de segurança estejam em vigor;
- Determinar se os dispositivos de auditoria e segurança são válidos;
- Verificar se os parâmetros, trilhas de auditorias e mecanismos de segurança estão funcionando da forma devida.

Xu, Chadwick e Otenko (2005) abordam em seu trabalho, que em sistemas computacionais, ações relacionadas às operações do usuário e controle de acesso devem ser mantidas em arquivos de *log* seguros para detecção de violações e intrusões ou para objetivos relacionados à auditoria do sistema.

Seguindo o que dizem Xu, Chadwick e Otenko (2005), nos arquivos de *log* é comum que seja armazenada uma grande quantidade de informações sigilosas. Sendo assim, é de

extrema importância garantir que, em caso de violações do sistema, tais *logs* não sejam comprometidos e que a violação possa ser detectada.

No caso de um atacante experiente, o primeiro objetivo será obter acesso ao sistema de *log* de auditoria para que seja possível apagar qualquer vestígio que sinalize o ataque, e manter o *modus operandi* em segredo perante aos administradores do sistema, assegurando, assim, que esse método fique disponível para ataques futuros (BELLARE; YEE, 1997).

Kalis e Belloum (2018) afirmam que é importante destacar que, em casos onde os arquivos de *logs* são armazenados no mesmo dispositivo da aplicação, as duas informações podem estar igualmente vulneráveis e poderão ser comprometidas em cenários de ataques. Pensando nisso Kalis e Belloum (2018) propõem uma nova forma de assegurar a integridade dos dados e dos *logs*, que utiliza da geração de *hash* dos *logs* gerados pela trilha de auditoria para enviar ao *blockchain*, portanto estes dados ficarão completamente isolados e independentes do sistema principal.

Com a tradução literal de *blockchain* tem-se o que pode-se chamar de "cadeia de blocos" ou "corrente de blocos", o que justamente sugere seu funcionamento. De acordo com Galen et al. (2018), *blockchain* trata-se de um livro de registros de transações públicas, seguro e digital.

Crosby et al. (2016) afirmam que o *blockchain* é fundamentalmente um banco de dados distribuído em um registro público, com todos os eventos e transações digitais realizados e compartilhados entre todos os seus participantes. Caso algum desses participantes tente adulterar algum bloco, este último será descartado, pois será comparado ao restante dos participantes. Desta maneira, para que seja possível a adulteração de informações do *blockchain*, seria preciso um ataque que atingisse ao mesmo tempo mais de 50% dos participantes, "o que é operacionalmente impossível" (SANT'ANA, 2018, p. 4).

Kalis e Belloum (2018) afirmam que, além do conceito básico do *blockchain*, é importante também conhecer e entender o conceito de *Smart Contracts*, ou no português, *Contratos Inteligentes*. Contratos Inteligentes são uma forma de fazer cumprir digitalmente um contrato ou acordo entre partes através de um código computacional.

Kalis e Belloum (2018) seguem dizendo que este conceito é anterior ao *blockchain* há mais de uma década, mas apenas quando o *blockchain* foi implementado é que finalmente foi possível implementar esses contratos sem a necessidade de um terceiro confiável para execução do contrato.

Uma tecnologia descentralizada que implementa *blockchain* e oferece a capacidade de publicar contratos inteligentes em seu conteúdo é o Ethereum (KALIS; BELLOUM, 2018). Ao publicar esses contratos para o Ethereum *blockchain*, todas as partes envolvidas podem inspecionar facilmente o contrato tendo a garantia de que o contrato execute

exatamente como especificado (KALIS; BELLOUM, 2018).

A proposta de Kalis e Belloum (2018) consiste em armazenar parâmetros durante a interação do usuário, e ao final, o método de publicação é chamado, para enviar o identificador *hash* de auditoria para o contrato inteligente Ethereum. Este método é executado de forma assíncrona, para que o resto do aplicativo possa continuar em execução, enquanto a transação é executada no *blockchain*.

Kalis e Belloum (2018) explicam que o método utilizado para envio das transações ao *blockchain* funciona de forma assíncrona, pois este procedimento pode levar algum tempo até que seja executado e aceito na rede, e ainda destaca um potencial ponto fraco em que, em situação de falha ou interrupção da aplicação durante o processo de auditoria, uma entrada de auditoria incompleta pode ser gravada no banco de dados, enquanto nada é enviado para o *blockchain*, o que invalidaria a trilha de auditoria. Mais detalhes sobre o funcionamento da aplicação podem ser vistos no capítulo 3.

Outro ponto negativo observado pelos autores no método utilizado é em casos onde cinco ou mais transações estão sendo processadas simultaneamente ocorre falha. O que, segundo os autores "geralmente não é um problema para aplicativos menores, mas pode definitivamente impactar aplicativos maiores" (KALIS; BELLOUM, 2018).

A partir deste contexto, o presente trabalho implementa dois métodos alternativos de envio dos *hash's* ao *blockchain*. À vista disso, aplicou-se experimentos de comparação e análise relativos à proposta de Kalis e Belloum (2018) para avaliar na prática como tais implementações se comportam em diferentes cenários de demanda computacional. De tal forma que com base nos resultados foi possível determinar a viabilidade e melhor emprego destas aplicações, conforme descrito na seção 1.1.

1.1 OBJETIVOS

Baseando-se no trabalho de Kalis e Belloum (2018) para envio de trilhas de auditoria ao *blockchain*, este trabalho tem como objetivo a análise de desempenho em algoritmos transacionais, seguindo a metodologia descrita na seção 1.2.

1.1.1 Objetivo geral

Utilizar a tecnologia do *blockchain* para armazenamento de informações que sirvam de lastro para autenticar a integridade de dados através de trilhas de auditoria, e, analisar o desempenho de diferentes alternativas implementadas, através da medição do tempo de duração das transações em diferentes cenários.

Além disso, espera-se contribuir com resultados extraídos de experimentos comparativos entre estas diferentes implementações que utilizam desta tecnologia, para determinar melhores e piores opções em diferentes cenários, de acordo com o contexto aplicado.

1.1.2 Objetivos específicos

Outrossim espera-se contribuir com os resultados extraídos de experimentos comparativos entre diferentes implementações que utilizam desta tecnologia, para determinar melhores e piores opções em diferentes cenários, de acordo com o contexto aplicado, conforme listados nos itens a seguir:

- Construção de métodos alternativos para envio e armazenamento de informações de *log* no *blockchain*;
- Criação de experimentos comparativos medindo desempenho no tempo das transações entre as diferentes propostas implementadas;
- Evidenciação das vantagens e desvantagens de cada implementação nos diferentes cenários;
- Formulação de recomendações sobre as diferentes propostas de implementação de acordo com diferentes contextos.

1.2 METODOLOGIA

Este trabalho do ponto de vista de sua natureza, enquadra-se como experimental, pois consiste, especialmente, em aplicar os objetos do experimento sob influência de determinadas variáveis, em situações conhecidas e controladas, para então visualizar os resultados produzidos pelo experimento sobre o objeto (GIL, 2008).

Partindo desta metodologia, este trabalho consiste na aplicação de experimentos práticos sobre diferentes métodos construídos para envio de informações de trilha de auditoria para o *blockchain*. Da mesma forma, utiliza outra implementação sugerida por Kalis e Belloum (2018), aplicando os mesmos experimentos e variáveis, a fim de analisar o seu desempenho, de forma comparativa. Com os resultados obtidos é possível identificar qual proposta demonstra o melhor resultado, e formula recomendações de qual melhor se aplica em sistemas com maior ou menor demanda de transações. Para executar as práticas de experimento, teve-se como base o procedimento abaixo descrito, como sugerido por Turrioni e Mello (2012):

- Planejamento do experimento;
- Operacionalização das variáveis;
- Definição das técnicas de análise dos dados do experimento;
- Especificação da unidade de análise ou montagem do banco de ensaio;

- Especificação do tempo para condução do experimento;
- Projeto do experimento;
- Realização do experimento e coleta de dados;
- Análise estatística;
- Análise dos resultados;
- Conclusão;
- Redação e publicação dos resultados.

Estes itens mencionados podem ser observados ao longo do trabalho, como forma de conduzir as etapas de pesquisa.

1.3 ESTRUTURA DO TRABALHO

Com o intuito de melhorar organização e o entendimento sobre os assuntos que cercam os experimentos deste trabalho, abaixo é possível observar uma breve explicação de como os capítulos desta pesquisa se dividem.

- **1 Introdução** - trata dos aspectos gerais do trabalho, objetivos, metodologia e estrutura;
- **2 Conceitos básicos sobre trilha de auditoria e *blockchain*** - traz conceitos iniciais para correto entendimento dos elementos utilizados nos experimentos;
- **3 Algoritmos para envio de transações** - demonstra a aplicação criada para execução dos experimentos e extração dos resultados com os métodos assíncrono, síncrono e assíncrono com fila;
- **4 Resultados de desempenho** - faz uma análise sobre o desempenho dos resultados obtidos durante os experimentos;
- **5 Conclusão** - destaca os objetivos alcançados e resultados obtidos durante a execução do trabalho.

Conforme dito, para que seja possível ter uma melhor compreensão sobre os experimentos feitos neste trabalho, antes é necessário contextualizar alguns conceitos sobre as tecnologias e métodos utilizados. O capítulo 2 explica os conceitos necessário para o correto entendimento dos experimentos.

2 CONCEITOS BÁSICOS SOBRE TRILHA DE AUDITORIA E *BLOCKCHAIN*

Este capítulo tem como objetivo elucidar conceitos básicos necessários para a correta compreensão dos experimentos aplicados e que são tratados ao longo deste trabalho.

A divisão das seções deste capítulo se dá da seguinte forma. A seção 2.1 trata das trilhas de auditorias convencionais, seu funcionamento e aplicação, seguida da seção 2.2 que trata justamente do *blockchain*, utilizado para complementar o uso convencional dos *logs*, as seções 2.3 e 2.4 tratam dos contratos inteligentes e da tecnologia Ethereum, respectivamente. Ao final, na seção 2.5 é possível ver as considerações finais do autor sobre este capítulo.

2.1 TRILHAS DE AUDITORIA (*LOGS*)

Simon, Santos e Hara (2008) explicam que a auditoria tem como objetivo evitar e identificar ações suspeitas e fraudulentas, extraindo dados sobre atividades dos usuários do banco de dados. Estes dados extraídos são analisados com o intuito de identificar problemas de segurança e sua origem.

Xu, Chadwick e Otenko (2005) completam dizendo que oferecer armazenamento íntegro e permanente dos registros de *log* é a principal funcionalidade de um serviço de auditoria, isso para que falhas possam ser identificadas sempre que ocorrerem.

Segundo Hawthorn et al. (2006), a identificação de padrões e ações suspeitas é um importante requisito para a segurança de um sistema. E mais do que isso, a auditoria deve ser feita de forma transparente e independente, assegurando que todas as informações relevantes sejam catalogadas.

Roratto e Dias (2014) também abordam o tema dizendo que, trilhas de auditoria, as quais também são chamadas de *logs*, são utilizadas com o objetivo de assegurar o fluxo preciso das ações executadas dentro de um sistema. Toda entrada, transação ou fonte de um determinado documento precisa ser feita e registrada em um relatório ou arquivo.

Peterson et al. (2007) explicam que os *logs* permitem a conferência de um sistema de arquivos em um dado período no passado. A auditoria é um protocolo de contestação/resposta entre o indivíduo que está auditando e o sistema de arquivos que está sendo auditado.

Para experimentos rápidos, Bosworth e Kabay (2002) falam que o rastreamento pode ser empregado em uma única transação. Porém, para assegurar que o controle funcione de forma consistente, o experimento deve abranger grandes volumes de dados e vários períodos de tempo.

Na perspectiva de Elmasri e Navathe (2008), a trilha de auditoria de sistema inclui um registro para cada operação aplicada ao banco de dados que possa necessitar de recuperação por falha no sistema ou falha de operação.

Elmasri e Navathe (2008) seguem dizendo que, dados pertinentes ao usuário que executou a ação também podem ser registrados, tais como: conta do usuário e terminal onde o procedimento foi executado. Estes dados facilitam na identificação de qualquer alteração feita no banco de dados que possa ser suspeita. Deste modo, pode ser executada uma auditoria no banco de dados, que consiste em analisar as trilhas de auditoria de todos os acessos e operações aplicadas durante um determinado período.

Por fim, Elmasri e Navathe (2008) concluem que, em qualquer caso de operação ilegal ou não autorizada encontrada, o administrador do banco de dados poderá determinar qual usuário foi utilizado para executar tal operação. *Logs* de banco de dados são muito importantes para bancos de dados sensíveis, que possuem muitas transações e usuários, como no caso de um banco de dados bancário.

2.1.1 *Logs* em SGBDs

Utilizar sistemas de arquivos para armazenamento de trilhas de auditoria não é recomendado, pois caso algum arquivo for excluído, não existirá registro dessa ação. Problema o qual não ocorre em um banco de dados, segundo McDowell (2007).

Segundo o que dizem Simon, Santos e Hara (2008), grande parte dos SGBDs permitem registrar ações na base de dados, gerando *logs* de auditoria. Entretanto, os métodos não são transparentes, e, por muitas vezes, requerem a criação de *triggers* para cada objeto a ser monitorado.

Sallach (1992) complementa dizendo que, a utilização de *triggers* não é uma prática recomendada, pois a cada ação realizada são executadas estas rotinas, o que onera o uso do banco de dados.

Uma *trigger*, ou gatilho em sua tradução, é um objeto de banco de dados associado a uma tabela e que é ativado quando um determinado evento ocorre para esta. Alguns usos de gatilhos são para realizar verificações de valores a serem inseridos em uma tabela, ou para realizar cálculos em valores envolvidos em uma atualização ou para geração de *logs*, conforme mencionado anteriormente (MYSQL, 2020).

Uma *trigger* é definida para ser ativada quando uma instrução de inserção, atualização ou deleção de um registro na tabela associada é executada. Essas operações de linha são eventos de *trigger*. Por exemplo, as linhas podem ser inseridas por instruções de inserção ou atualização, e um gatilho de inserção é ativado para cada linha inserida. Um gatilho pode ser definido para ativar antes ou depois do evento de gatilho. Por exemplo, é possível ter um acionador ativado antes de cada linha que é inserida em uma tabela ou

após cada linha que é atualizada (MYSQL, 2020).

McDowell (2007) afirma que, a criação de dados de auditoria pode ser feita utilizando funções genéricas ou através de políticas de uso do banco de dados e *logs* automáticos. Na geração de registros de *logs*, deve-se ter uma tabela separada para gravação das trilhas de auditoria associadas à criação, alteração e exclusão de dados e registros do banco de dados.

McDowell (2007) segue afirmando que, para que seja possível pesquisar registros de *logs* de forma mais específica e rápida, cada pacote no sistema deverá ter uma trilha de auditoria individual. A vantagem desta abordagem é que todas as entradas de auditoria estão intimamente associadas com os dados que representam.

Segundo Roratto e Dias (2014), diante da crescente dependência dos sistemas de armazenamento de dados críticos, novas soluções para proteção e monitoramento destes dados devem ser desenvolvidas. Atualmente existem alguns estudos[1] e implementações na área, mas nenhuma solução definitiva para segurança de *logs*.

Roratto e Dias (2014) ainda apresentam dois sistemas comerciais da Oracle e IBM, que prometem resolver os problemas de acessos indevidos a informações sigilosas e integridade dos registros de auditoria. Roratto e Dias (2014) concluem que a área de trilhas de auditoria é muito promissora, importante, e recomendam a realização de novas pesquisas sobre controle e segurança da informação.

Como alternativa aos *logs* armazenados em banco de dados, Kalis e Belloum (2018) sugerem a utilização do *blockchain* para gravação e autenticação de trilhas de auditoria. A seção 2.2, mostra maiores detalhes sobre o funcionamento desta tecnologia.

2.2 *BLOCKCHAIN*

Conforme a definição de Nakamoto (2008), o *blockchain* é uma estrutura de dados compartilhada que distribui seus dados por uma rede constituída por nós, de forma que não haja um ponto isolado de falha ou nenhum controle centralizado que possa ser comprometido.

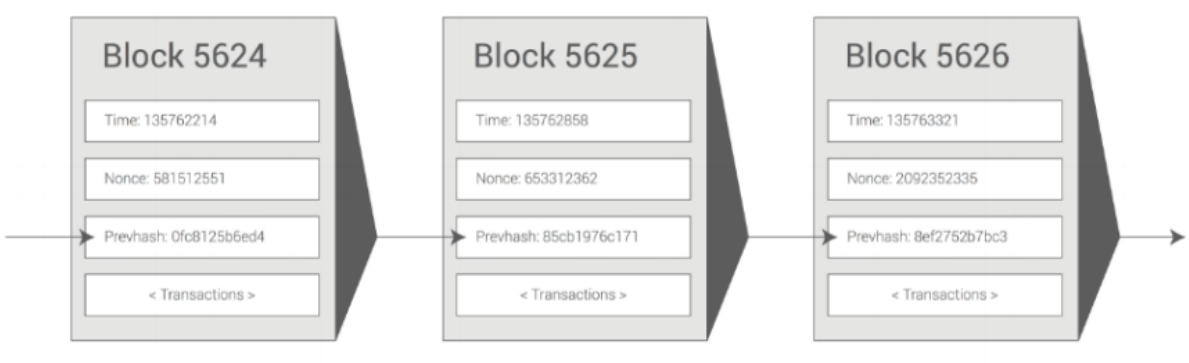
Esta rede utiliza um algoritmo de consenso que permite que estes nós distribuídos e independentes, aprovem transações legítimas e rejeitem transações maliciosas através de um mecanismo de votação, dizem Buterin et al. (2014).

Buterin et al. (2014) explicam que, no *blockchain*, como a tradução literal sugere "cadeia de blocos", os dados são armazenados em um bloco, e vários blocos em sequência formam uma cadeia. Além dos dados, cada bloco também possui a data / hora, o *nonce* e

[1] https://abre.ai/cH0H acessado em 13/05/2021

um *hash* do bloco anterior, a fim de torná-lo mais resistente à manipulação (mais detalhes sobre os atributos podem ser vistos na seção 2.2.1). Esta estrutura pode ser vista na Figura 1.

Figura 1 – Estrutura do *blockchain*



Fonte: Buterin et al. (2014)

Buterin et al. (2014) seguem explicando que, se um atacante adulterar o conteúdo de um bloco anterior, o *hash* deste bloco será alterado com ele, e, uma vez que o *hash* é alterado, ele deixa de ser referenciado por outro bloco, e não será aceito pelo resto da rede, e por consequência, uma bifurcação da rede será criada.

Em caso de bifurcações, a cadeia mais longa é a líder. Sendo assim, para que o bloco modificado seja aceito pelo resto da rede, o atacante precisaria aumentar sua cadeia mais rápido do que o resto da rede combinada, para só então passar a ser a cadeia mais longa. Entretanto os recursos de todo o resto da rede são logicamente maiores do que a de um único nó sozinho, o que inviabilizaria este processo e garante a integridade dos dados do *blockchain*, concluem Buterin et al. (2014).

As próximas seções irão trazer maiores informações sobre os atributos dos blocos, suas características e algoritmos.

2.2.1 Atributos

Na Figura 2 é possível observar a representação visual de um bloco que compõe uma cadeia de blocos. Nela é possível observar os atributos necessários para o correto funcionamento da rede, sendo que cada um deles funciona da seguinte maneira:

- **Índice** - é a posição do bloco na cadeia (HAN, 2020).
- ***Timestamp*** - é a data e hora do momento exato em que o bloco foi adicionado à rede.

- ***Nonce*** - utilizado como variável para satisfazer a regra de criação do *hash*.
- ***Previous Hash*** - trata-se do *hash* do bloco anterior, responsável por fazer a ligação entre os blocos, criando a cadeia.
- ***Data*** - o conteúdo do bloco, com todas as transações e dados armazenados.
- ***Hash*** - esta informação é gerada a partir de um algoritmo que leva em consideração todos os atributos anteriores, através do popularmente conhecido processo de mineração.

O processo de mineração, nada mais é que a geração de um *hash* da junção de todos os atributos do bloco. Han (2020) explica que um *hash* possui algumas características como:

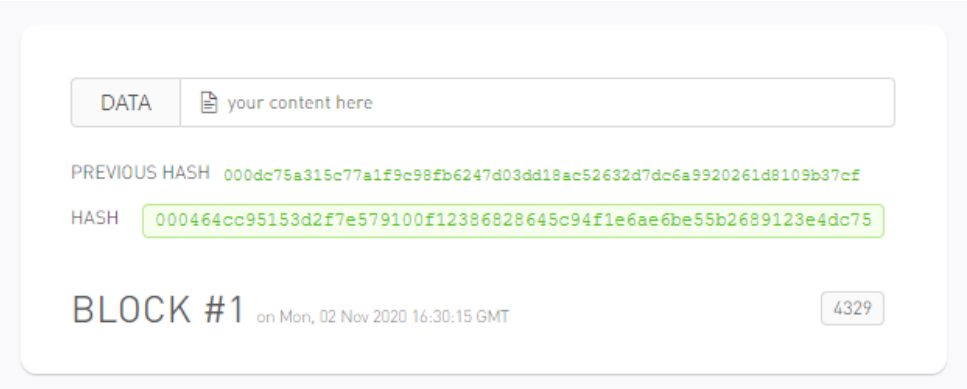
- tamanho fixo;
- o mesmo conteúdo sempre gera o mesmo *hash*;
- diferentes conteúdos possuem diferentes *hash's*;
- não é possível converter *hash* para seu conteúdo original;
- uma pequena alteração no conteúdo gera uma grande alteração no *hash*.

Mota et al. (2019) também explicam que uma função geradora de *hash* é um algoritmo capaz de mapear dados de comprimento dinâmico para dados de comprimento fixo, de forma que o mesmo conteúdo sempre gere o mesmo *hash*, e qualquer alteração no conteúdo gere um *hash* completamente diferente.

Han (2020) diz que apesar do esforço computacional para gerar um *hash* ser muito baixo, precisa atender certos requisitos, como é o caso visto na Figura 2, onde este só é válido quando iniciado com uma sequência de 3 zeros (000). Dessa forma, quanto maior o número de zeros necessários, maior é a dificuldade para encontrar um *hash* válido, pois, menores são as possibilidades.

Han (2020) segue dizendo que, para que seja possível gerar um *hash* diferente, é preciso modificar o conjunto de dados, pois como dito anteriormente, o mesmo conteúdo sempre gera o mesmo *hash*. Por este motivo existe o atributo *nonce*, que se trata de uma variável numérica que inicia em 0 e é acrescida de 1 a cada tentativa fracassada de gerar um *hash* que atenda aos requisitos.

Figura 2 – Atributos de um bloco



Fonte: adaptado de Han (2020)

A seguir é possível verificar as etapas necessárias para a geração de um *hash* válido, conforme Han (2020):

1. o atributo *nonce* inicializa com o valor 0.
2. o *hash* é gerado a partir do conteúdo do bloco.
3. o *hash* é testado para ver se atende aos requisitos, sendo 2 (duas) as possibilidades:

   A. o *hash* atende os requisitos, neste caso o processo está concluído.

   B. o *hash* não atende os requisitos, neste caso:

      I. o *nonce* é acrescido em 1.

      II. o processo retorna ao item 2.

Estas tentativas de gerar um *hash* válido, conforme citado no item 3, durante a mineração é o que onera o processo, o torna custoso e ao mesmo tempo seguro, pois uma alteração no conteúdo de um bloco exigiria que uma nova mineração fosse feita para gerar um novo *hash* para o bloco atual e todos os que o sucedem (HAN, 2020).

2.2.2 Algoritmo de Consenso

Para que um *blockchain* seja viável, apenas uma cadeia é necessária, e para eleger essa cadeia existe um algoritmo de consenso, onde todos ou a maioria dos integrantes da rede concordam em qual é o próximo bloco na cadeia, e por sua vez, qual é o estado de toda esta rede *blockchain*, explica Mitt (2018).

As redes *blockchains* podem ser divididas em privadas e públicas. Atzori (2015) explica que redes privadas possuem membros “semi-confiáveis”, onde somente membros credenciados podem fazer parte da rede. Neste caso a necessidade de algoritmos fortes

de consenso é menor, pois todos os participantes já foram previamente registrados e verificados. Além disso, este tipo de rede normalmente possui um número pequeno de integrantes, o que possibilita mecanismos de consenso alternativos.

Em *blockchains* públicos, Atzori (2015) explica que se espera que o número de integrantes da rede seja grande e confiável, pois qualquer integrante pode se juntar à rede. Sendo assim, mecanismos de consenso para este tipo de rede *blockchain* precisam tomar conta dos ataques de Sybil.

Mitt (2018) explica que ataques de Sybil em uma rede *blockchain* permitem que um único integrante utilize várias identidades para influenciar o processo de consenso. No caso da rede *blockchain* do Bitcoin, resolve-se esse problema projetando a rodada de consenso para ser computacionalmente difícil. Os nós precisam provar que utilizaram uma quantidade significativa de energia com o *Proof-to-Work (PoW)* para resolver o problema criptográfico.

Mitt (2018) segue dizendo que a criptomoeda Ethereum utiliza o algoritmo *Proof-of-Stake (PoS)*, que tenta suavizar o gasto energético do *PoW*, decidindo aleatoriamente qual é o próximo nodo que criará o próximo bloco.

Karamitsos, Papadaki e Barghuthi (2018) concluem a partir disso, que a segurança e a imutabilidade da tecnologia *blockchain* se dão graças aos algoritmos de consenso, em especial em redes públicas. Uma tentativa de fraude em um bloco necessita uma sequência de fraudes em seus blocos precedentes, o que torna o mecanismo protegido contra tentativas de adulteração.

Além das características citadas, o *blockchain* também permite a anexação de código computacional em seu interior através dos chamados *Smart Contracts*, que podem ser vistos na seção 2.3.

2.3 *SMART CONTRACT*

Smart Contract, ou em português, Contrato Inteligente, trata-se de um código computacional anexado ao *blockchain* identificado por um endereço único. Ele possui um conjunto de funções executáveis e que variam de estado. Sempre que alguma transação interage com este contrato, estas funções são executadas. Estas transações possuem parâmetros de entrada que são exigidos pelas funções do contrato. Imediatamente após a execução de uma função, as variáveis de estado contidas no contrato podem mudar de estado, dependendo da lógica utilizada na função, explicam Bahga e Madisetti (2016).

Bahga e Madisetti (2016) afirmam que os *smart contracts* podem ser escritos em linguagens de alto nível, como Solidity ou Python. Após serem compilados, os contratos são enviados à rede *blockchain*, que por sua vez atribui um endereço exclusivo aos contratos. Qualquer usuário da rede *blockchain* pode executar as funções de um contrato

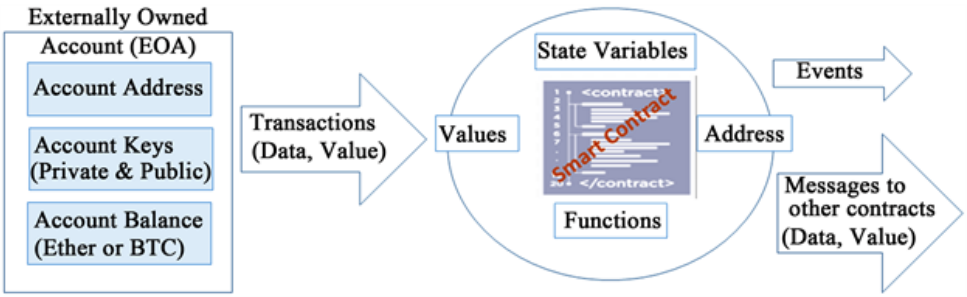
enviando transações para ele. O código do contrato é executado em cada nó da rede como parte da verificação de novos blocos.

Karamitsos, Papadaki e Barghuthi (2018) explicam que um contrato inteligente é um programa de código identificado por um endereço na rede *blockchain*, onde seus principais componentes são um conjunto de funções executáveis e variáveis de estado. Cada transação possui parâmetros de entrada que requerem uma função no contrato, que por consequência, podem alterar o *status* das variáveis de estado, dependendo da implementação lógica.

Karamitsos, Papadaki e Barghuthi (2018) concordam com Bahga e Madisetti (2016), onde dizem que o código do contrato inteligente é escrito em linguagens de alto nível. Karamitsos, Papadaki e Barghuthi (2018) ainda complementam que o código do contrato é compilado em *bytecode* usando compiladores como Solidity ou Serpent. Após isso, o código do contrato é carregado junto à rede *blockchain*, que receberá um endereço exclusivo, como dito anteriormente por Bahga e Madisetti (2016).

Contratos Inteligentes podem enviar mensagens para outros contratos. A mensagem é composta pelo endereço do remetente, endereço do destinatário, valor da transferência e um campo de dados que contém os dados de entrada para o contrato do destinatário. Existe diferença entre mensagem e transação, onde a transação é produzida por *External Owned Account (EOA)*, enquanto a mensagem é produzida por um contrato inteligente, conforme mostrado na Figura 3 (KARAMITSOS; PAPADAKI; BARGHUTHI, 2018).

Figura 3 – Estrutura do Contrato Inteligente



Fonte: Karamitsos, Papadaki e Barghuthi (2018)

Karamitsos, Papadaki e Barghuthi (2018) também explicam que *External Owned Account (EOA)* são considerados contas de usuários de propriedade externa. Essas contas são controladas por chaves privadas. Este ator pode criar transações para transferir valor, criar contratos inteligentes ou chamar funções de contrato.

Buterin (2015) demonstram que uma das redes que implementam *blockchain* e permitem o armazenamento de Contratos Inteligentes em seu interior é a Ethereum. Maiores esclarecimentos sobre esta tecnologia podem ser vistos na seção 2.4.

2.4 ETHEREUM

A tecnologia Ethereum está popularmente associada à criptomoeda Bitcoin, que também utiliza o *blockchain* como base. Entretanto, é importante entender que o Ethereum é muito mais do que uma criptomoeda. Ethereum é uma plataforma de *software* aberto baseado na tecnologia *blockchain* que permite aos desenvolvedores construir e implementar aplicações que utilizam o conceito da descentralização (DISTRICT0X, 2020).

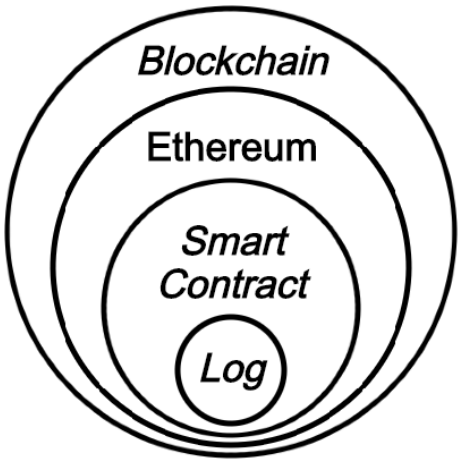
Como mencionado no parágrafo anterior, a Ethereum também possui uma criptomoeda em seu interior chamada Ether, a qual é utilizada para "alimentar" aplicativos construídos no *blockchain* Ethereum (DISTRICT0X, 2020).

Da mesma maneira, Buterin (2015) afirma que a tecnologia Ethereum é uma das preferidas para o desenvolvimento de Contratos Inteligentes. É uma plataforma de execução e processamento de contratos completa baseada em *blockchain*. A arquitetura e implementação do Ethereum são totalmente independentes da criptomoeda Bitcoin. O programador pode criar seus formatos de transações, transições de estado e funções de eventos e regras de propriedade. O código do *software* é executado em uma máquina virtual denominada *Ethereum Virtual Machine (EVM)*.

2.5 CONSIDERAÇÕES FINAIS

Os conceitos apresentados neste capítulo servem para demonstrar a capacidade e segurança das tecnologias que cercam o *blockchain*, e também como base para melhor compreensão dos capítulos seguintes. A Figura 4 elucida de forma visual a relação entre os conceitos citados, de forma a facilitar o entendimento e representar como estão inseridos uns nos outros.

Figura 4 – Relação entre os conceitos abordados



Fonte: Elaborada pelo autor

Além do Ethereum, outras tecnologias possuem o mesmo nível de segurança, e permitem criar Contratos Inteligentes em seu interior. Porém, conforme visto na seção 2.4, Ethereum atualmente é a mais utilizada, inclusive já em aplicações relacionadas às trilhas de auditoria. Portanto esta tecnologia foi a escolhida pelo autor para implementar uma aplicação de *log* utilizando *blockchain*. Os detalhes sobre esta implementação e seu desempenho estão disponíveis no próximo capítulo.

3 ALGORIMOS PARA ENVIO DE TRANSAÇÕES

Conforme visto no capítulo 2, os conceitos servem para melhor compreensão dos algoritmos e implementações feitas neste capítulo. Vale destacar que todas as citações de envio de *hash* ao *blockchain* tratam-se do envio do *log* à rede *blockchain* Ethereum, conforme representado na Figura 4.

Como parte do planejamento do experimento, este capítulo explica o funcionamento da aplicação desenvolvida por Kalis e Belloum (2018), bem como o algoritmo de envio das transações. E como alternativa a este algoritmo, duas propostas diferentes para análise dos resultados, com o intuito de avaliar o desempenho através da medição do tempo de execução das transações de cada um dos métodos utilizados.

Todos os códigos fonte citados nesse capítulo podem ser encontrados na integra no Apêndice B.

3.1 CONTACT APP

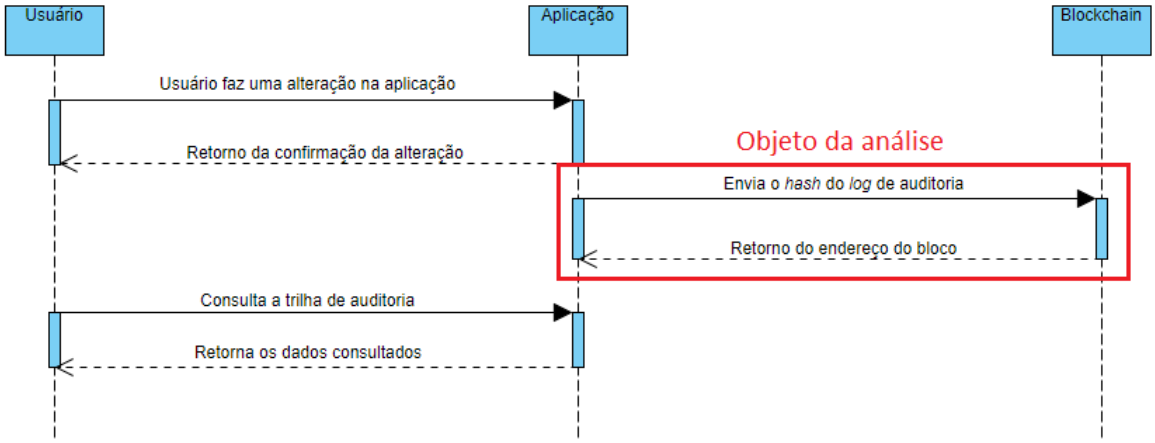
Conforme dito anteriormente na seção 1.2, como base para execução dos experimentos, foi utilizada a proposta de implementação feita por Kalis e Belloum (2018), que segundo eles, é usada para para gerenciamento de contatos internos nas empresas.

A aplicação consiste em cadastro e gerenciamento de contatos, de forma que qualquer alteração feita na base de dados gera uma entrada de auditoria representada em sua aplicação pela classe `"AuditEntry"`, com alguns atributos como usuário, data, hora e o *hash* de todo o conjunto desses dados.

Então, após a obtenção deste *hash*, a classe `"AuditerServiceUsingBlockchain"` é responsável por fazer seu envio ao *blockchain*, e, ao final, adicionar o endereço do bloco criado à entrada de auditoria. Isso de forma assíncrona à aplicação para que o usuário não perceba todo este processamento. A Figura 5 ilustra o fluxo das operações no diagrama de sequência, bem como o objeto de análise abordado neste trabalho.

O envio de somente o *hash* ao *blockchain* permite que os custos de mineração das transações sejam consideravelmente diminuídos, e também proporciona a auditoria de qualquer tipo de dado, uma vez que o *hash* tem seu comprimento fixo, conforme explica Kalis (2018). Contudo, esta forma de auditoria não substitui os *backup's* e cópia dos dados, pois o *hash* por si só não permite que os dados originais sejam recuperados em caso de fraude, permitindo apenas a identificação da mesma, como exemplificado nos 3 (três) cenários criados por Kalis e Belloum (2018), descritos a seguir.

Figura 5 – Diagrama de sequência com o fluxo das operações no método assíncrono



Fonte: Elaborada pelo autor

Kalis e Belloum (2018) exemplificam o primeiro cenário da seguinte forma: um administrador de sistemas em treinamento recebe uma solicitação de seu colega para restaurar alguns arquivos do banco de dados do sistema, porém sem muita experiência, o administrador restaura todo o banco de dados da aplicação sobrescrevendo os últimos registros criados após o *backup*, impossibilitando a recuperação destas informações.

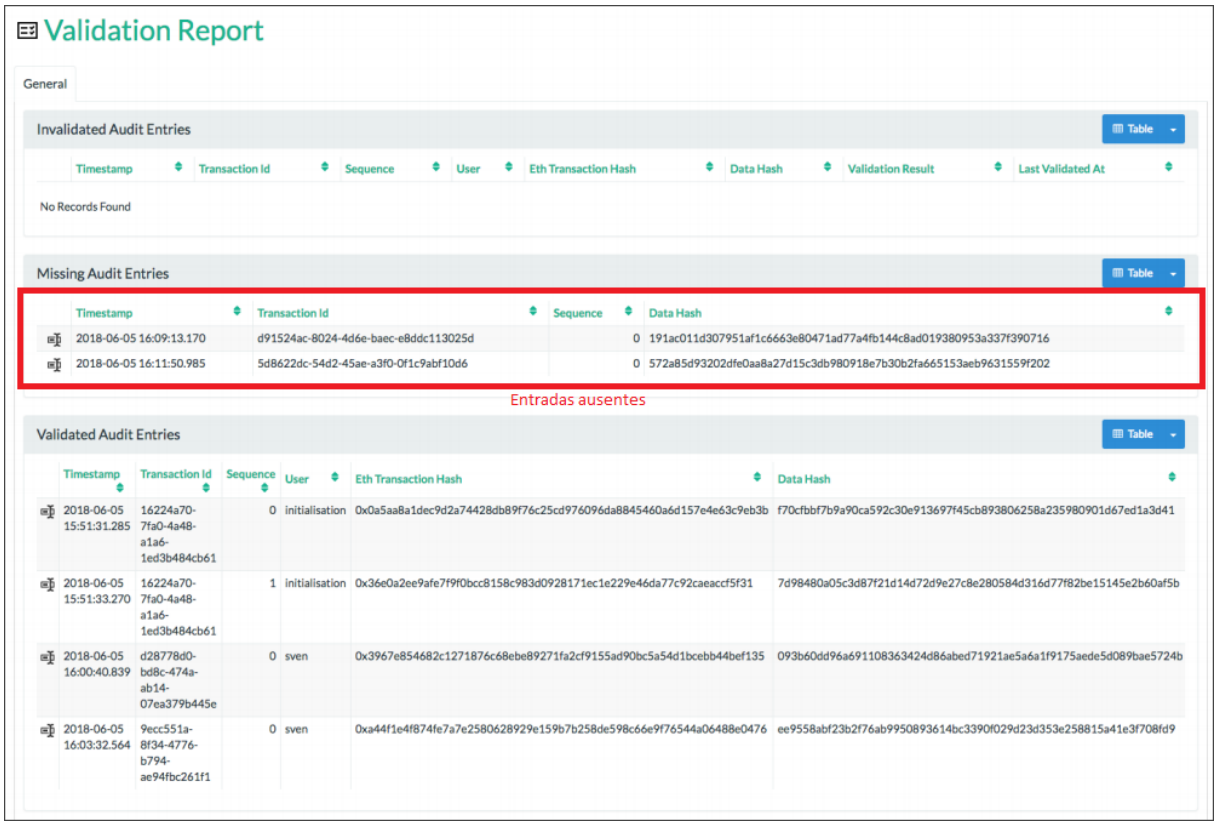
Kalis e Belloum (2018) seguem dizendo que este ponto destaca uma fraqueza na aplicação, mas também permite visualizar como este tipo de erro pode ser identificado muito cedo.

Para simular este cenário um *backup* do banco de dados da aplicação foi feito e restaurado depois de algumas alterações terem sido feitas, substituindo estas novas alterações no banco de dados. Após isso, uma validação da trilha de auditoria foi feita. A Figura 6 trata-se de um recorte feito na página da aplicação *Contact App*, responsável por fornecer dados pertinentes as trilhas de auditoria da aplicação, neste caso ela demonstra que existem entradas de auditoria ausentes, pois estas foram excluídas durante a restauração do *backup* do banco de dados. Um vídeo demonstrando este cenário está disponível no YouTube[1] (KALIS; BELLOUM, 2018).

Kalis e Belloum (2018) demonstram no segundo cenário que um determinado funcionário de uma empresa decide largar seu emprego e aposentar-se. Para complementar sua renda, ele decide mudar o beneficiário de uma das maiores faturas da sua antiga companhia para a sua própria conta bancária, e, após o pagamento, desfez a alteração, pois a empresa utiliza trilhas de auditoria, e ele deseja cobrir seus rastros.

[1] https://youtu.be/nfekoK6pUqU acessado em 17/04/2021

Figura 6 – Entradas de auditoria ausentes



Fonte: Kalis e Belloum (2018) adaptada pelo autor

Kalis e Belloum (2018) seguem narrando que como ele ainda possui suas credenciais antigas da corporação, decide conectar-se ao banco de dados da empresa e remover todas as entradas de auditoria que registraram suas ações.

Contudo, a instituição em questão possuia *backup* rotineiro e trilha de auditoria em *blockchain*, que puderam ser recuperados, identificando facilmente as entradas de auditorias faltantes, trazendo o antigo funcionário e suas ações maliciosas à tona (KALIS; BELLOUM, 2018).

Por último, em seu terceiro cenário, Kalis e Belloum (2018) descrevem um funcionário de uma empresa que altera maliciosamente o endereço de e-mail de um contato, e em seguida altera a entrada de auditoria diretamente no banco de dados, fazendo com que o usuário auditado seja seu colega ao invés dele próprio, transferindo a culpa.

Após isso, denuncia seu colega ao superior da empresa, e, como a empresa utiliza trilhas de auditoria em *blockchain*, foi possível identificar rapidamente que a entrada de auditoria foi adulterada. Porém como a auditoria ainda não havia sido incluída no *backup* da empresa, não há provas concretas das intenções do funcionário. Isso porque a implementação mostra as entradas de auditoria alteradas, mas não mostra os dados reais

dentro da entrada de auditoria, uma vez que apenas o *hash* é enviado ao *blockchain*, e este não pode ser decodificado para a informação original (KALIS; BELLOUM, 2018).

Estes cenários apresentados por Kalis e Belloum (2018), reforçam a importância da utilização de trilhas de auditoria em aplicações, e a principal vantagem de enviá-las ao *blockchain*, uma vez que este não pode ser alterado nem mesmo por usuários com os mais altos privilégios.

3.2 APLICAÇÃO PARA EXECUÇÃO DE EXPERIMENTOS E EXTRAÇÃO DE RESULTADOS (APEEER)

Considerando os cenários descritos na seção anterior, é possível observar algo em comum entre os 3 (três) casos: todos possuíam alguma informação associada ao *blockchain* para determinar a integridade da informação original.

Em outras palavras, toda alteração feita no banco de dados gera automaticamente uma entrada de auditoria, a qual possui um *hash* que precisa ser enviado ao *blockchain* o mais breve possível.

Contudo, para que a aplicação funcione sem onerar tempo aos usuários, Kalis (2018) explica, em seu trabalho *"Using blockchain to validate audit trail data in private business applications"*, que as transações são enviadas ao *blockchain* de forma assíncrona, para que o resto do aplicativo possa continuar executando enquanto a transação é enviada ao *blockchain*.

Kalis (2018) completa dizendo que ao final desta transação, uma função de retorno é chamada para adicionar o *hash* da transação Ethereum à entrada de auditoria da aplicação, para que possa ser consultada futuramente.

Ao fim, Kalis (2018) relata, no capítulo *"Limits of the implementation"* de seu trabalho, que quando mais de cinco transações simultâneas são enviadas ao *blockchain*, novas transações não são aceitas.

Para contornar este problema, Kalis (2018) sugere a criação de um protocolo para enfileirar automaticamente novas transações quando já houver alguma em processamento.

Como forma estudar o problema destacado por Kalis (2018), uma aplicação denominada "Aplicação Para Execução de Experimentos e Extração de Resultados" (APEEER) foi desenvolvida como projeto do experimento, para a criação e envio de *hash's* ao *blockchain*. Logo, o método utilizado por Kalis (2018) pode ser replicado permitindo a execução de experimentos em maior escala, uma vez que esta nova aplicação desenvolvida gera dados aleatórios automaticamente, sem precisar simular manualmente a criação e edição de contatos.

Além de simular o método utilizado por Kalis (2018), duas formas alternativas

de envio de *hash* foram desenvolvidas com o intuito de executar experimentos e ver os diferentes comportamentos e resultados gerados por cada uma delas, para avaliar e poder decidir qual o melhor método a se utilizar em diferentes situações.

A primeira delas trata-se do envio síncrono das transações, justamente para medir o desempenho sem qualquer interferência de transações paralelas. E a segunda, trata-se do mesmo método assíncrono, porém com um limitante de transações máximas em paralelo, criando uma lista de espera das demais transações através de uma fila, conforme sugerido pelo próprio Kalis (2018).

A principal vantagem de extrair o método de envio para outra aplicação é a possibilidade de fazer com que a aplicação gere dados em massa para executar os experimentos com um volume de dados muito maior, o que seria inviável utilizando a aplicação *"Contact App"* para gerar trilhas de auditoria uma a uma.

Para gerar os dados e simular trilhas de auditorias, um método para criação de registros foi adicionado à APEEER, de forma que fosse possível determinar o número de registros a serem gerados, e sua frequência, que poderia variar de 1 a 60 registros por minuto. Com isso, este método cria de forma aleatória *hash's* para serem enviados ao *blockchain*.

Abaixo é possível analisar o código fonte do método responsável pela criação dos registros a serem enviados ao *blockchain*.

Código Fonte – Método responsável pela criação dos registros

```
1  public static void insert(Connection db) {
2   try {
3
4    Contrato contrato = null;
5    // instancia a classe para geração de valores aleatórios
6    Random rand = new Random();
7    MessageDigest m = null;
8    String val = null;
9
10   int i = 0;
11   int n = 0;
12
13   // inicia a iteração para a geração de registros
14   while (i < App.ROWS) {
15
16    // gera um número aleatório baseado a frequência de registros
17    n = rand.nextInt(60 / (App.ROWS_PER_MINUTE > 0 ? App.
        ROWS_PER_MINUTE : 1));
18
19    if (n == 0 || App.ROWS_PER_MINUTE == 0) {
20
```

```
21      // gera um valor aleatório para representar o log
22      val = String.valueOf(rand.nextInt());
23      // inicia a geração do hash do valor gerado
24      m = MessageDigest.getInstance("MD5");
25      m.update(val.getBytes(), 0, val.length());
26
27      val = String.valueOf(new BigInteger(1, m.digest()).toString
            (16));
28
29      while (val.length() < 32) {
30       val = "0" + val;
31      }
32      // conclui a geração do hash
33
34      // gera o log de resultado
35      File.log("insert          ", val);
36
37      // cria o objeto do contrato
38      contrato = new Contrato();
39      contrato.setSituacao(0);
40      contrato.setHashConteudo(val);
41
42      // insere o objeto no banco de dados
43      DBContrato.insert(db, contrato);
44      i++;
45     }
46     if (App.ROWS_PER_MINUTE > 0) {
47      Thread.sleep(1000);
48     }
49    }
50
51   } catch (Exception e1) {
52    e1.printStackTrace();
53   }
54  }
```

Para os experimentos aplicados, optou-se por utilizar a geração de 30 registros por minuto, definido na constante *"App.ROWS_PER_MINUTE"*. Em outras palavras, é como se um usuário qualquer de um sistema gerasse 1 (uma) entrada de auditoria a cada 2 (dois) segundos.

À medida que estas entradas de auditorias são geradas, 2 (dois) procedimentos são executados: o primeiro deles é a inclusão do registro em um banco de dados, a fim de gravar estas informações para posterior consulta, e o segundo é o envio do *hash* ao *blockchain*.

Para o envio destes dados ao *blockchain* em qualquer aplicação que utilize dessa

tecnologia, três elementos básicos são necessários. Seguindo exemplos e orientações de outros autores é possível ver, conforme indicado a seguir, estes elementos e quais ferramentas foram utilizadas para fornecer cada um:

- Uma rede Ethereum, que irá receber as transações. Obtida através do Infura, conforme citado por Hu et al. (2018), o qual funciona como um portal que fornece acesso à rede;
- Uma carteira Ethereum, que servirá para identificar a origem das transações e arcar com os custos de mineração. Criada utilizando o MetaMask, também visto no trabalho de Danilin, Lukin e Reshetova (2017);
- Um modelo de contrato para armazenamento das transações, codificado na liguagem Solidity e compilado pelo Remix como no exemplo de Dika (2017).

O Infura, citado no primeiro item da lista, permite acesso à diversas redes diferentes de Ethereum, além é claro, da rede principal. Contudo a rede utilizada no experimentos foi a denominada Ropsten, que serve exclusivamente para testes, e não possui custo monetário envolvido para efetivar as transações, diferentemente da rede principal que exige investimento real para compra de Ethereum, utilizado para mineração das transações.

Para desenvolvimento da aplicação, a linguagem de programação escolhida foi Java, pois trata-se da mesma linguagem utilizada por Kalis (2018) em seu projeto, o que permite que os experimentos funcionem com os mesmos métodos, e por consequência permite que a análise de desempenho seja executada com o mesmo critério e rigor, uma vez que os métodos são os mesmos.

Após providenciar todos os itens necessários para utilização do *blockchain*, foi possível criar uma aplicação para comunicação com a rede Ethereum[2].

Feito vários testes e ajustes com a aplicação operando, o método de envio utilizado por Kalis (2018) foi adicionado à aplicação para que fosse possível analisar o desempenho do método, observando os problemas apontados por Kalis (2018).

Como métrica para análise do desempenho, utilizou-se do tempo de duração das transações com o *blockchain* para que seja possível quantificar e comparar com os outros métodos.

Nas próximas subseções é possível ver mais detalhes sobre sobre o método de envio assíncrono, utilizado por Kalis (2018) e as alternativas propostas neste trabalho, sendo elas, o envio síncrono e o assíncrono com fila.

[2] Um exemplo completo das ferramentas com tutorial para criação encontra-se disponível em https://github.com/DiovaneGabriel/SmartContract

3.2.1 Envio assíncrono

Kalis (2018) utilizou em sua aplicação o envio assíncrono dos dados ao *blockchain*, para que o usuário possa seguir utilizando a aplicação sem precisar aguardar o envio da entrada de auditoria. Porém, como já mencionado anteriormente neste capítulo, esta abordagem apresentou falhas em situações onde mais de 5 transações ocorriam ao mesmo tempo.

Na sequência é possível observar um trecho de código escrito por Kalis (2018), com algumas adaptações feitas pelo autor, onde o envio da transação é feita de forma assíncrona.

Código Fonte – Método responsável pelo envio das transações

```
1  public void commitAuditEntry(Boolean sync) {
2   // objeto que representa a entrada de auditoria
3   AuditEntry auditEntry = currentAuditEntry.get();
4
5   if (auditEntry == null) {
6    return;
7   }
8
9   try {
10   // obtém a data atual para geração dos logs de resultados
11   LocalDateTime dateStart = LocalDateTime.now();
12
13   // instancia a transação
14   RemoteCall<TransactionReceipt> transaction = web3Service.
        getAuditTrailContract()
15     .audit(auditEntry.getIdentifier(), auditEntry);
16
17   if (sync) {
18    // faz o envio síncrono
19    transaction.send();
20    // chama a função para gravação dos logs de resultados
21    new File(currentAuditEntry.get().getHash(), dateStart).run();
22   } else {
23    // faz o envio assíncrono
24    transaction.sendAsync().thenAccept(new
         TransactionReceiptConsumer(this.contrato, this.connection))
25      .exceptionally(e -> {
26       // em caso de erro, exibe o erro
27       e.printStackTrace();
28       return null;
29      }).thenRun(
30        // callback para gravação dos logs de resultados
31        new File(currentAuditEntry.get().getHash(), dateStart));
32   }
```

```
33   } catch (Exception e) {
34    e.printStackTrace();
35   }
36   // limpa a variável com a entrada de auditoria
37   currentAuditEntry.set(null);
38  }
```

Na linha 1 é possível observar o parâmetro `"sync"`, utilizado na linha 17, onde é feita a distinção no método de envio, sendo assíncrono, ou síncrono, que é descrito na subseção 3.2.2, e, sendo esta a única diferença no código fonte entre os dois métodos.

Com o objetivo de tentar identificar uma melhor alternativa para agilizar o envio e evitar falhas, duas outras abordagens foram implementadas, conforme justificado anteriormente, com a intenção de avaliar a melhor aplicação em diferentes cenários.

3.2.2 Envio síncrono

A primeira alternativa trata-se do envio síncrono, de forma com que o serviço de envio de *hash's* ao *blockchain* envie apenas um destes por vez, respeitando a ordem de geração, e somente após o retorno desta transação o próximo *hash* será enviado.

Da mesma forma que no envio assíncrono, os usuários da aplicação não precisam aguardar o envio da transação, pois a entrada de auditoria é guardada em uma fila, e o usuário pode seguir utilizando a aplicação normalmente.

Entretanto, como o envio da entrada só ocorre após a conclusão de todas as outras anteriores a ela, pode haver um tempo de espera indesejado na fila, o que pode ser um problema em potencial dependendo do tipo da aplicação. Isso porque quanto maior for o tempo de espera do *hash* na fila, maior é o tempo que a informação fica vulnerável dentro do sistema, permitindo que ela seja fraudada antes mesmo de ser enviada ao *blockchain*.

O código fonte utilizado para tratar este método pode ser visto na subseção 3.2.1, com a diferença de ter o parâmetro `"sync"` setado como `"true"`.

3.2.3 Envio assíncrono com fila

O segundo método alternativo trata-se do mesmo envio assíncrono utilizado por Kalis (2018), porém com um limitador de transações simultâneas, fazendo com que novas transações sejam enfileiradas até que alguma transação pendente seja concluída.

Isso evita problemas relacionados a excesso de transações paralelas, conforme menciona Kalis (2018). Desta forma, qualquer novo *hash*, gerado durante o envio máximo de transações paralelas, fica aguardando em uma fila até que alguma destas seja concluída.

Este método não resolve o problema do tempo de espera na fila, mas pode diminuí-lo, uma vez que mais envios podem ser executados ao mesmo tempo.

Abaixo é possível conferir o código fonte do trecho responsável pelo gerenciamento da fila de transações.

Código Fonte – Método responsável pelo gerenciamento da fila de transações

```
1  private static void limitAsync(Connection connection,
       AuditerServiceUsingBlockchain blockchain) {
2
3    try {
4     //obtém o número de transações pendentes
5     Integer running = DBContrato.getContratosProcessando(connection
          );
6     //calcula a quantidade de transações vagas
7     Integer limit = App.LIMIT - running;
8     limit = limit < 0 ? 0 : limit;
9
10    //consulta no banco de dados as próximas transações a serem
          enviadas com base no número de vagas
11    ArrayList<Contrato> contratos = DBContrato.
          getContratosPendentesLimit(connection, limit);
12    //se houver algum contrato
13    if (contratos != null) {
14     //itera cada um dos contratos
15     for (Contrato contrato : contratos) {
16      //seta a situação do contrato para "1"
17      contrato.setSituacao(1);
18      //atualiza no banco de dados
19      DBContrato.updateById(connection, contrato, contrato.getId())
            ;
20
21      //chama o método responsável pelo envio da transação
22      blockchain.auditAsync(contrato);
23      //aguarda 1 segundo
24      Thread.sleep(1000);
25     }
26    }
27   } catch (Exception e) {
28    e.printStackTrace();
29   }
30
31 }
```

Este método é chamado em *loop* durante a execução da aplicação através de uma interação `"while(true)"`, para que fique consultando o banco de dados e consumindo a fila de transações.

O controle da fila se dá através do atributo `"situacao"` do objeto `"contrato"`, o qual é utilizado internamente no método `"DBContrato.getContratosProcessando"`, a

fim de contabilizar o número de transações pendentes que estão sendo processadas, e assim poder determinar quantas ainda podem ser feitas. A constante `"App.LIMIT"` utilizada na linha 7, é explicada com maiores detalhes na seção 3.3.

Após determinar o número de transações que ainda podem ser enviadas, uma nova consulta é feita no banco de dados. Com isso, uma iteração deste resultado é feita para enviar o contrato para o *blockchain* utilizando o método assíncrono, descrito na subseção 3.2.1.

Além de fazer o envio da transação, o contrato tem sua situação alterada para "1", conforme é possível ver na linha 17. Isso garante que o contrato não será enviado novamente e indica que ele está sendo processado, o que servirá como referência na consulta da linha 5.

3.3 EXECUÇÃO DOS EXPERIMENTOS

Conforme mencionado anteriormente, para a execução de todos os experimentos foi estabelecido que a geração dos registros para envio ao *blockchain* fosse de 30 registros por minuto. Desta forma todos os métodos foram submetidos ao mesmo cenário de utilização, que seria equivalente a aplicações que gerem aproximadamente 1 entrada de auditoria a cada 2 segundos.

Outro ponto a se considerar durante os experimentos é que apenas o desempenho do envio dos *hash's* foi medido. Desta forma, a única preocupação do método responsável por gerar os registro foi gerar um *hash* de 32 *bits* de uma informação aleatória, que tem o mesmo tamanho dos *hash's* gerados pela aplicação "Contact App".

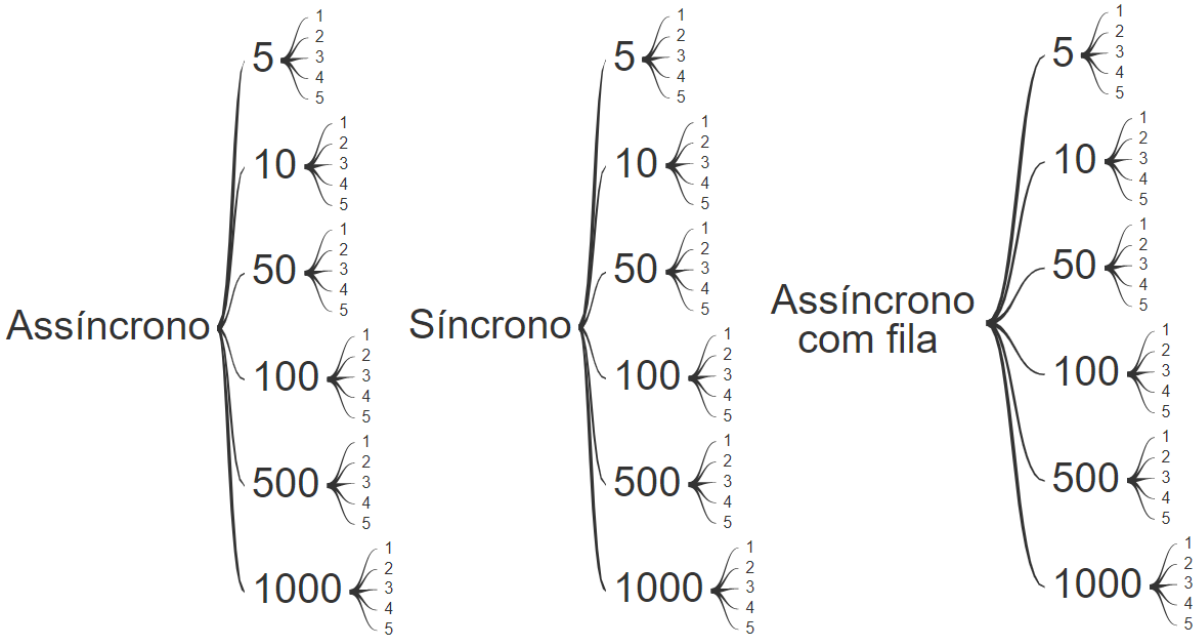
Para execução dos experimentos 6 (seis) cenários diferentes foram criados para cada um dos 3 (três) métodos de envio, cada um com diferentes volumes de dados, sendo eles 5 (cinco), 10 (dez), 50 (cinquenta), 100 (cem), 500 (quinhentos) e 1000 (mil) registros.

Cada um dos 6 (seis) cenários dos 3 (três) métodos foram submetidos a 5 (cinco) rodadas de experimentos, ou seja, cada cenário foi executado 5 (cinco) vezes para cada método. Na Figura 7 é possível visualizar o esquema elaborado para execuções dos experimentos e montagem do banco de ensaios.

No caso do método assíncrono com fila, foi estabelecido que o número máximo de transações simultâneas em execução seria de 5 (cinco), pois, segundo Kalis (2018), a partir disso ocorreram problemas durante as transações.

Inicialmente seriam feitas 10 (dez) rodadas para cada cenário, contudo a medida que os experimentos foram sendo executados observou-se que haveria uma duração maior do que o tempo oferecido pelo cronograma.

Figura 7 – Planejamento dos Experimentos



Fonte: Elaborada pelo autor

Este ponto serviu como um limitador na especificação do tempo para condução do experimento. Na seção 4.2 é possível ver o tempo médio necessário para a execução de cada rodada, sem considerar o tempo de intervalo entre uma e outra.

Apesar da ressalva feita por Kalis (2018) sobre o problema com mais de 5 (cinco) execuções simultâneas, nada semelhante a isso foi presenciado durante os experimentos do método assíncrono.

Em algumas situações, o tempo de execução da transação excedeu 10 (dez) minutos, gerando o cancelamento automático da transação por *timeout*, que por padrão é 600 (seiscentos) segundos. Mas isso não está relacionado a transações simultâneas, pois ocorreram alguns episódios de *timeout* com o método síncrono, onde não há transações paralelas.

Para cada experimento, seja ele apenas uma nova rodada de um mesmo cenário ou um cenário diferente, algumas constantes são definidas na classe principal, e a partir delas a aplicação executa o cenário correspondente e gera o *log* de execução. Na sequência, é possível ver um trecho de código com as constantes responsáveis pelos diferentes comportamentos da APEEER.

Código Fonte – Trecho de código com as definições das constantes

```
1 // número da rodada do experimento
2 public static final Integer TEST = 1;
3
```

```
4  // método utilizado
5  public static final Integer METHOD = 0;
6
7  // número de registros a serem gerados
8  public static final Integer ROWS = 10;
9
10 // quantidade de registros por minutos a serem gerados
11 public static final Integer ROWS_PER_MINUTE = 30;
12
13 // número máximo de transações paralelas
14 public static final Integer LIMIT = 5;
15
16 // nome do arquivo de log a ser gerado
17 public static final String FILE_PATH = "log_#" + TEST + "_" +
       METHOD + "_" + ROWS + "_" + ROWS_PER_MINUTE + "_" + LIMIT + ".
       txt";
```

Dentro do escopo da aplicação, estas constantes se mantêm invariáveis, porém, no ponto de vista dos experimentos são elas que permitirão que a aplicação tenha diferentes comportamentos através da operacionalização destas variáveis.

Para poder determinar o desempenho de cada método, foi desenvolvido um método para extração de resultados, através da geração de *log's* de execução, com o registro de alguns comandos executados pela APEEER.

Para isso algumas funções foram feitas para padronizar e facilitar a geração dos *log's* dos resultados. Abaixo é possível observar duas funções responsáveis pela criação destes arquivos com nome predefinido pela constante `"App.FILE_PATH"` estabelecida na classe principal da aplicação.

Código Fonte – Métodos responsáveis pela geração de *log's* das execuções

```
1  public static void log(String what, String content) {
2   // @param what    procedimento que está sendo executado
3   // @param content hash do conteúdo da transação ou hash e tempo
        total separados por " "
4
5   // obtém a data e hora atual
6   LocalDateTime now = LocalDateTime.now();
7   // cria padrão de formatação da data e hora
8   DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd
        HH:mm:ss");
9   // monta a string do log a ser gerado
10  String log = dtf.format(now) + " " + what + (content != null ? "
        " + content : "");
11  // chama a função responsável por gerar o log
12  generate(log);
13 }
```

```
14
15 public static void generate(String str) {
16  // exibe o log no console
17  System.out.println(str);
18  try {
19   // adiciona a string de log ao final do arquivo de log da execuç
       ão
20   Files.write(Paths.get(App.FILE_PATH), ("\n" + str).getBytes(),
       StandardOpenOption.APPEND);
21  } catch (IOException e) {
22   e.printStackTrace();
23  }
24 }
```

Deste modo, alguns procedimentos chaves foram selecionados para serem incluídos no arquivo de *log*, para auxiliar nos experimentos e servir das informações de tempo deccorido para análise futura. É possível ver na lista abaixo os diferentes procedimentos inseridos no *log* e sua finalidade.

- `METHOD` - informação de cabeçalho sucedida pelo sinal "=" e "0", "1" ou "2", onde "0" representa o método assíncrono, "1" representa o método síncrono, e "2" representa o método assíncrono com fila;
- `ROWS` - informação de cabeçalho sucedida pelo sinal "=" e pelo número de registros gerados durante os experimentos, sendo ele 5 (cinco), 10 (dez), 50 (cinquenta), 100 (cem), 500 (quinhentos) ou 1000 (mil);
- `ROWS_PER_MINUTE` - informação de cabeçalho sucedida pelo sinal "=" e pelo número de registros gerados por minuto, sendo fixo "30", conforme explicado no capítulo 3;
- `LIMIT` - informação de cabeçalho sucedida pelo sinal "=" e pelo número máximo de transações símultâneas, utilizado no método assíncrono com fila, sendo fixo "5", conforme explicado no capítulo 3;
- `start` - antecedido pela data e hora de execução, separados por " ", indica o início da execução da aplicação;
- `startDeploy` - antecedido pela data e hora de execução, separados por " ", indica o início da abertura de comunicação com a rede *blockchain*;
- `endDeploy` - antecedido pela data e hora de execução, separados por " ", indica a conclusão da abertura de comunicação com a rede *blockchain*;
- `insert` - antecedido pela data e hora de execução e sucedido por um *hash* - separados por " ", indica a geração e inserção deste *hash* no banco de dados;

- `startTrasaction` - antecedido pela data e hora de execução e sucedido por um *hash*, separados por " ", indica o início da transação do *hash* gerado anteriormente com o *blockchain*;

- `endTransaction` - antecedido pela data e hora de execução, sucedido por um *hash* e por um número "x", separados por " ", indica conclusão da transação do *hash* iniciado anteriormente com o *blockchain*, onde "x" representa em segundos o tempo total decorrido desde o início da transação correspondente.

Desta forma, ao final da execução da aplicação, um arquivo com informações sobre a execução do experimento é gerado, permitindo analisar as variações de tempo nas transações, bem como o cenário em questão, através das informações de cabeçalho. Na Figura 8 é possível visualizar um exemplo de arquivo *log* gerado.

Figura 8 – Exemplo de *log* com os resultados da execução

```
METHOD=0
ROWS=5
ROWS_PER_MINUTE=30
LIMIT=5
2021/04/08 20:52:57 start
2021/04/08 20:52:57 startDeploy
2021/04/08 20:53:14 endDeploy
2021/04/08 20:53:14 insert           d6513b44ee5ac613f736fbf90816b267
2021/04/08 20:53:14 startTrasaction d6513b44ee5ac613f736fbf90816b267
2021/04/08 20:53:15 insert           a47aacbdfc335a662d426a5df0162fba
2021/04/08 20:53:15 startTrasaction a47aacbdfc335a662d426a5df0162fba
2021/04/08 20:53:17 insert           436e3b9222d39af0fbd718c9738e9186
2021/04/08 20:53:17 startTrasaction 436e3b9222d39af0fbd718c9738e9186
2021/04/08 20:53:18 insert           90c1b28e67d3fc0b8f9f2d69a960fd0a
2021/04/08 20:53:18 startTrasaction 90c1b28e67d3fc0b8f9f2d69a960fd0a
2021/04/08 20:53:19 insert           c3548d54c224714b2292816ae8414220
2021/04/08 20:53:20 startTrasaction c3548d54c224714b2292816ae8414220
2021/04/08 20:54:17 endTransaction   a47aacbdfc335a662d426a5df0162fba 61
2021/04/08 20:54:18 endTransaction   436e3b9222d39af0fbd718c9738e9186 61
2021/04/08 20:54:20 endTransaction   90c1b28e67d3fc0b8f9f2d69a960fd0a 61
2021/04/08 20:54:21 endTransaction   c3548d54c224714b2292816ae8414220 61
2021/04/08 20:54:31 endTransaction   d6513b44ee5ac613f736fbf90816b267 76
```

Fonte: Elaborada pelo autor

Ao final da realização dos experimentos, foi possível fazer a coleta de dados através da geração de *log's* de execução, tornando possível o acesso aos dados referente ao tempo de execução de cada transação, bem como seu horário de início e fim. Portanto, a partir disso foi possível a execução análises sobre estes dados, e gerar resultados de desempenho, como demonstrado no capitulo 4.

4 RESULTADOS DE DESEMPENHO

Com a execução de todos os cenários e rodadas estabelecidas, e extração de todos os arquivos de *logs* de resultados gerados, foi possível iniciar a leitura e o tratamento destes dados.

Este capítulo trata dos resultados obtidos através dos arquivos de *log* produzidos em cada uma das execuções, bem como as tratativas sobre as informações a fim de eliminar comportamentos anormais, e, as análises estatísticas dos resultados através de tabelas e gráficos gerados a partir dos dados.

4.1 TRATAMENTO DOS DADOS

Conforme mencionado no capítulo anterior, um dos dados contidos no arquivo de *log* gerado é o tempo total decorrido no envio da transação, que servirá como métrica para análise de desempenho, portanto, a partir desta informação é possível estruturar em forma de tabela os resultados de todas as rodadas de um determinado cenário e algoritmo.

Na Tabela 1 é possível visualizar os resultados obtidos a partir das 5 (cinco) rodadas executadas utilizando o método assíncrono com fila, bem como alguns dos indicadores resultantes de seu conteúdo.

Tabela 1 – Resultado obtido em 5 (cinco) rodadas utilizando o método assíncrono com fila com 10 (dez) registros

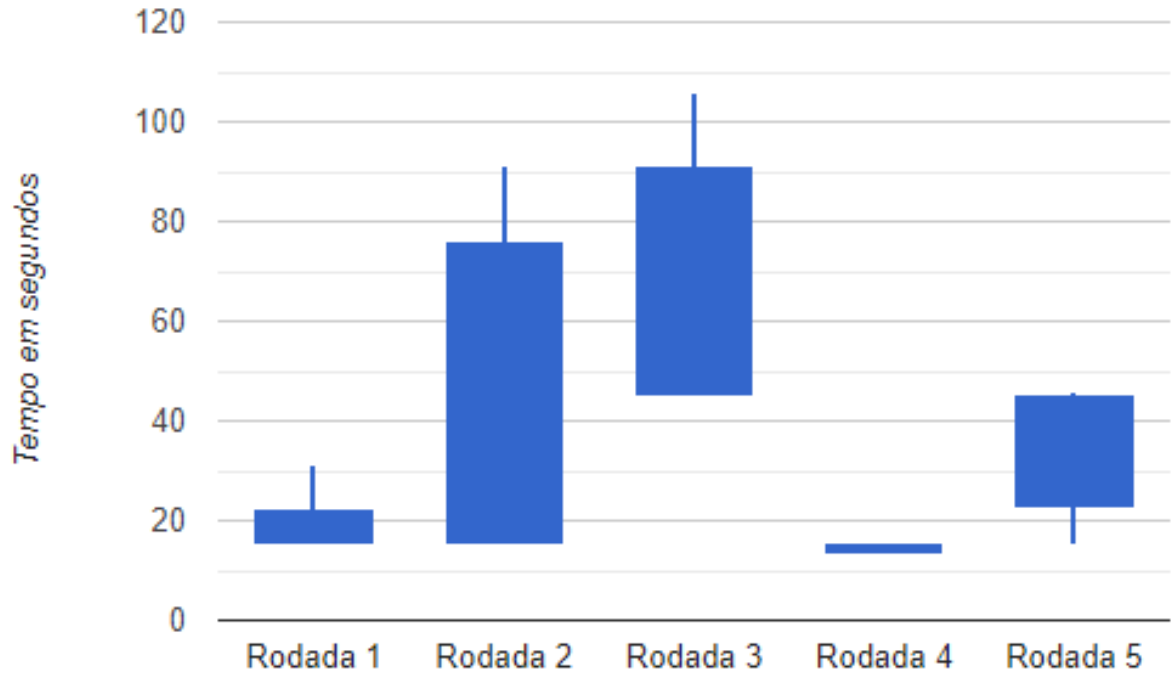
Registros	1	2	3	4	5
1	15	15	61	15	46
2	15	15	45	15	46
3	15	91	46	15	45
4	15	91	45	15	46
5	15	76	45	15	30
6	31	76	106	15	30
7	15	76	91	15	30
8	15	15	91	15	31
9	30	15	91	15	15
10	31	46	91	15	15
Média	19,7	51,6	71,2	15	33,4
Desvio Padrão	7,57	33,8	24,89	0	12,15
Mínimo	15	15	45	15	15
Máximo	31	91	106	15	46

Fonte: Elaborada pelo autor

Além dos dados, alguns outros indicadores, como 1º e 3º quartis , são calculados

para auxiliar na representação visual através dos gráficos. A Figura 9 representa os dados contidos na Tabela 1, juntamente com o 1º e 3º quartil.

Figura 9 – Gráfico *candlestick* dos resultados obtidos em 5 (cinco) rodadas utilizando o método assíncrono com fila com 10 (dez) registros



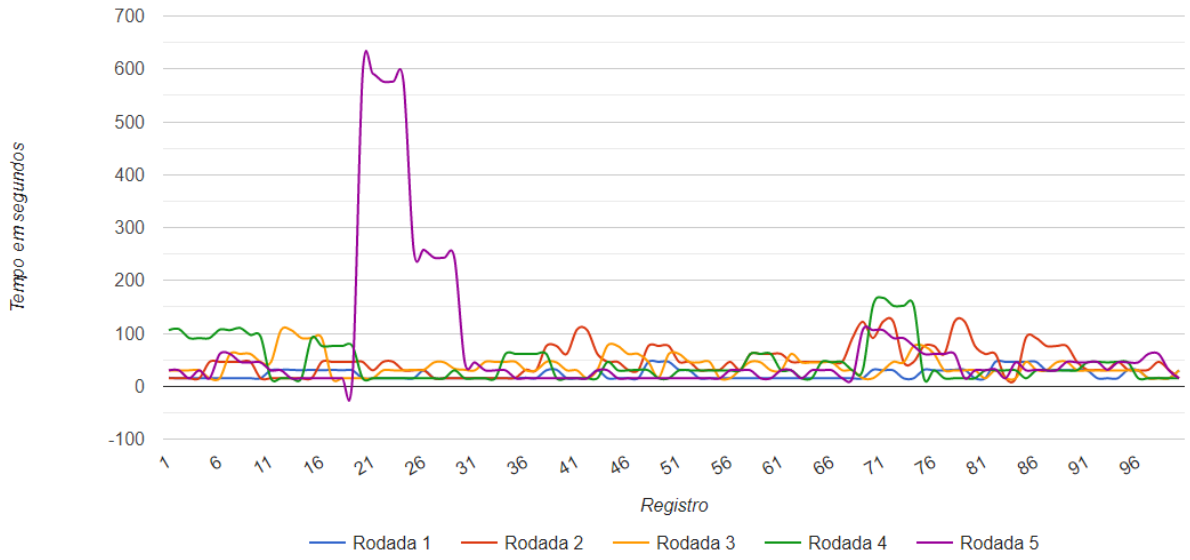
Fonte: Elaborada pelo autor

Neste gráfico é possível perceber que o tempo de duração das transações se mantém constante durante toda a rodada 4, comportamento este que não ocorre nas demais rodadas. Tal diferença sugere que as transações estão sujeitas à interferências externas à aplicação, como por exemplo, o fluxo de acesso a rede Ethereum, pois trata-se de um recurso compartilhado alheio à aplicação. Este comportamento é abordado com maiores detalhes na seção 4.2.

Outra representação visual utilizada para análise dos resultados é o gráfico de linha. Este modelo de gráfico permite visualizar uma linha para cada rodada, e nela um ponto para cada registro, o que no caso da Figura 10 evidencia alguns pontos completamente fora de padrão para a rodada 5.

Tal comportamento ocorreu em casos eventuais, com pouca recorrência, mas que influenciam nos resultados das médias. Portanto, optou-se por avaliar estes resultados através do cálculo de *outlier* que através de uma fórmula convencionada estabelece uma barreira inferior e outra inferior a partir do resultado de um conjunto de informações, que neste caso são os resultados de cada rodada.

Figura 10 – Gráfico de linha dos resultados obtidos em 5 (cinco) rodadas utilizando o método assíncrono com fila com 100 (cem) registros



Fonte: Elaborada pelo autor

Para avaliar os *outliers* duas fórmulas foram feitas, sendo a primeira delas o *outlier* extremo, que, como o nome sugere, desconsidera somente os seus valores extremos. Na Tabela 2 é possível ver os valores resultantes a partir desta fórmula.

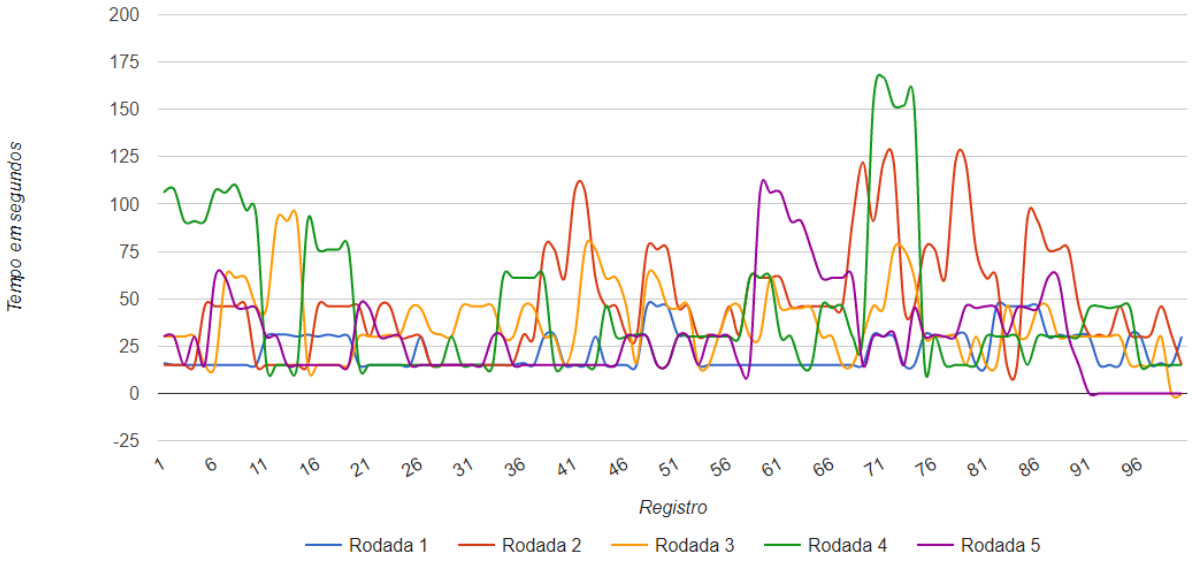
Tabela 2 – Valores de *outlier* e barreira extremos gerados sobre os resultados das transações do método assíncrono com fila com 100 (cem) registros

Registros	1	2	3	4	5
Outlier Extremo	45	93	48	138	93
Barreira Inferior Extrema	-30	-63	-18	-123	-78
Barreira Superior Extrema	75	154	94	199	139

Fonte: Elaborada pelo autor

Contudo, após a visualização de um novo gráfico, desconsiderando somente os valores fora das barreiras extremas, não houve um resultado satisfatório pois ainda existiam valores pontuais (Figura 11). A partir desta conclusão a segunda fórmula foi aplicada, agora desconsiderando todos os valores *outliers*, e não somente os extremos, gerando uma nova tabela e gráfico, desconsiderando os valores fora das barreiras (Tabela 3 e Figura 12 respectivamente).

Figura 11 – Gráfico de linha dos resultados obtidos em 5 (cinco) rodadas utilizando o método assíncrono com fila com 100 (cem) registros desconsiderando os registros fora das barreiras extremas



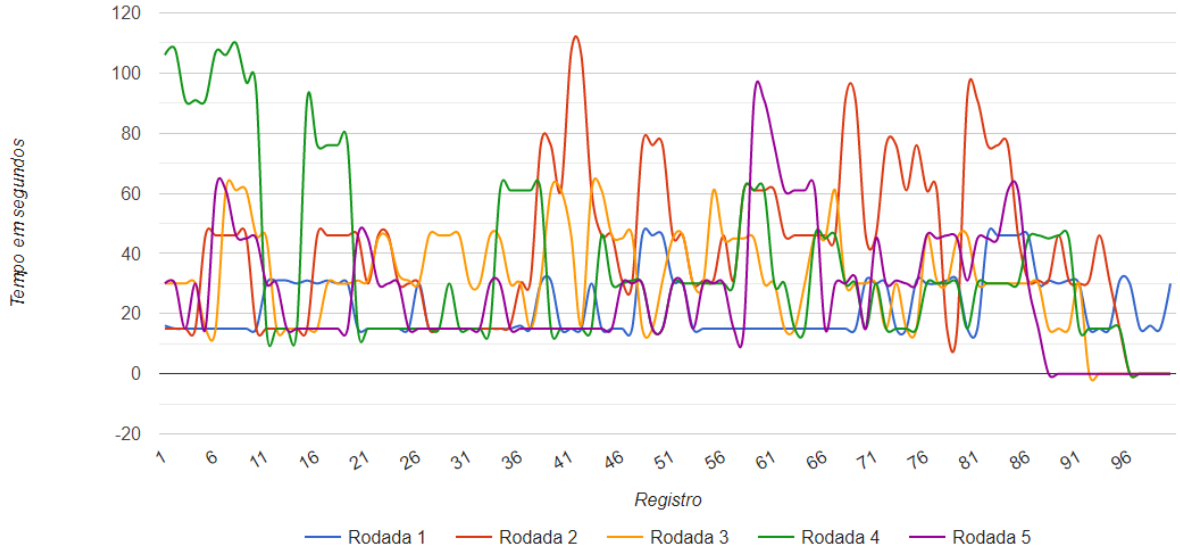
Fonte: Elaborada pelo autor

Tabela 3 – Valores de *outlier* e barreira gerados sobre os resultados das transações do método assíncrono com fila com 100 (cem) registros

Registros	1	2	3	4	5
Outlier	22,5	46,5	24	69	46,5
Barreira Inferior	-7,5	-16,5	6	-54	-31,5
Barreira Superior	52,5	107,5	70	130	92,5

Fonte: Elaborada pelo autor

Figura 12 – Gráfico de linha dos resultados obtidos em 5 (cinco) rodadas utilizando o método assíncrono com fila com 100 (cem) registros desconsiderando os registros fora das barreiras



Fonte: Elaborada pelo autor

A partir a visualização deste gráfico, foi possível concluir que os dados entre as barreiras superior e inferior representam de forma mais realista os dados gerados, além de que a maioria dos demais cenários sequer tiveram valores fora de suas barreiras. Os resultados podem ser vistos na íntegra, juntamente com o código fonte no Apêndice B.

Desta forma, todas as demais análises feitas na seção seguinte deste capítulo foram feitas com base nos valores entre as barreiras superior e inferior de sua respectiva rodada.

4.2 ANÁLISE DOS RESULTADOS

Esta seção traz em forma de 4 (quatro) subseções as diferentes técnicas de análise dos dados do experimento definidas para avaliação dos resultados.

4.2.1 Variação entre rodadas

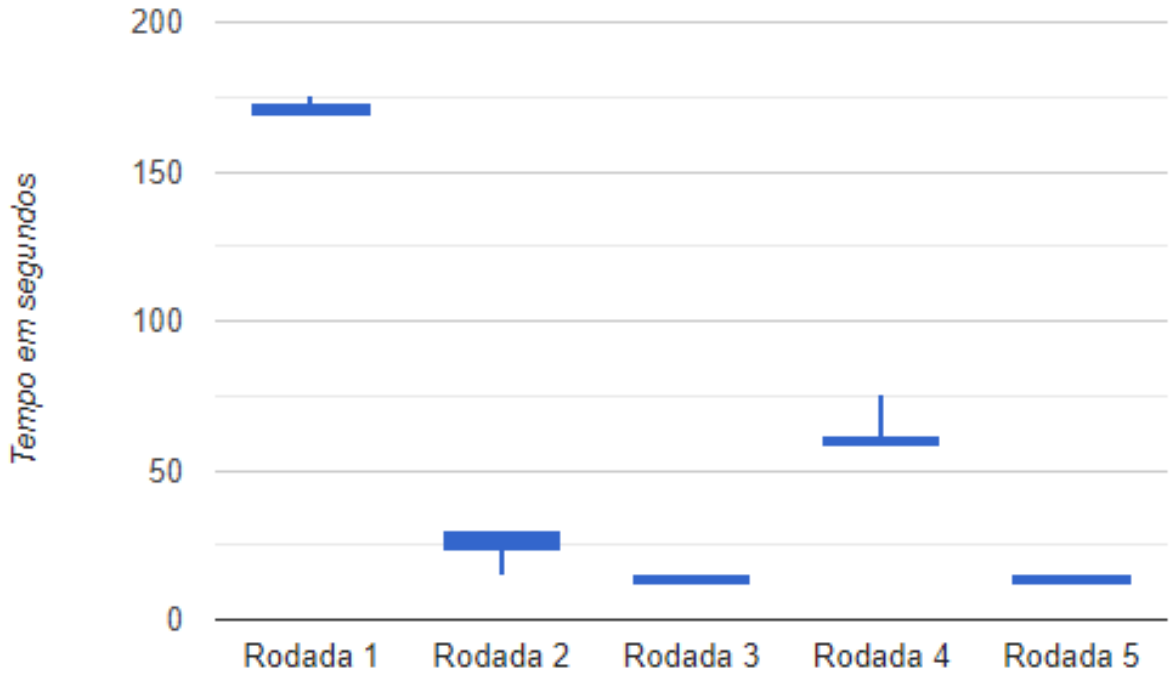
Como já citado na seção anterior, uma das características observadas nos gráficos é a variação no comportamento entre uma rodada e outra, uma vez que todas as rodadas possuem os mesmos parâmetros de algoritmo, número de registros e quantidade de registros gerados por minuto.

Vale reforçar que todas as 5 (cinco) rodadas tiveram as mesmas características e o mesmo volume de dados, que, como visto anteriormente na seção 3.3, trata-se de um *hash* de 32 *bits* de comprimento. Portanto, sabendo que todas as transações são homogêneas com relação ao seu conteúdo, esperava-se que o tempo de duração das transações fosse

muito semelhante entre as rodadas, se não idêntico, o que não ocorreu. Sendo assim, não há nada que explique este comportamento heterogêneo dentro da mesma aplicação, com os mesmos cenários e parâmetros.

Este comportamento variável entre as rodadas é possível ser observado em todos os cenários e algoritmos, tendo maior evidência nos cenários com menor número de registros, como é o caso da Figura 13.

Figura 13 – Gráfico *candlestick* dos resultados obtidos em 5 (cinco) rodadas utilizando o método assíncrono com 5 (cinco) registros



Fonte: Elaborada pelo autor

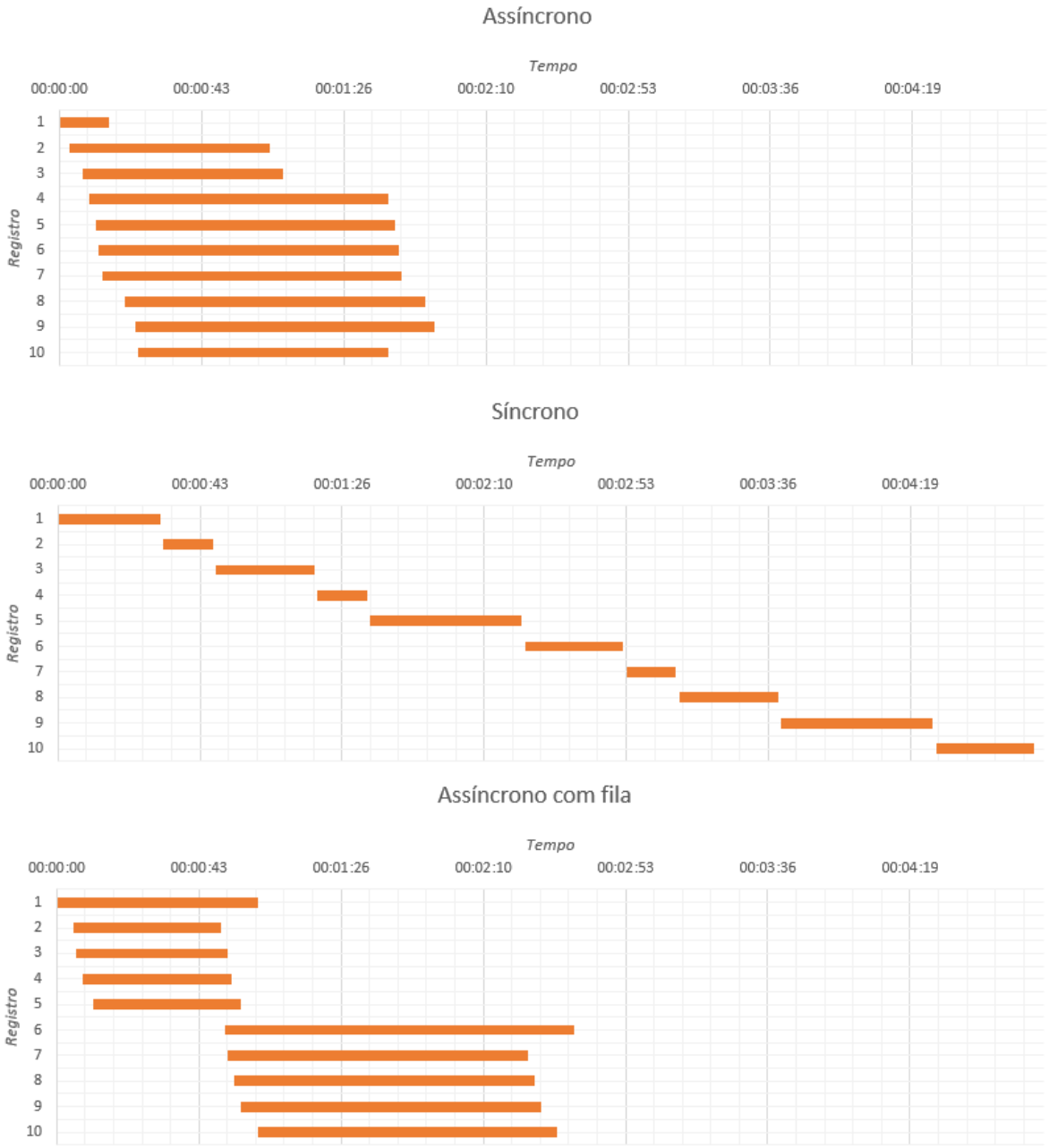
Esta figura reforça a sugestão dada anteriormente, onde as diferenças de tempo podem estar relacionadas à demandas externas sobre a rede Ethereum, pois, os 5 (cinco) registros executados em cada rodada, foram executados em um curto período de tempo, tornando-os menos expostos à variações externas, o que explica um baixo desvio padrão dentro de cada rodada, diferentemente de cenários com 50 (cinquenta) ou mais registros, que por sua vez possuem uma maior variação entre seus resultados. Os demais gráficos podem ser visualizados no Apêndice A.

Em outras palavras, quanto maior o número de registros a ser enviado ao *block-chain*, maior o tempo necessário, e por consequência, sujeitos a maiores intempéries da rede. Contudo neste caso, as 5 (cinco) rodadas foram executadas em diferentes momentos, o que justifica a diferença entre as médias em comparação umas com as outras.

4.2.2 Tempo total de execução dos métodos

Outro aspecto observado nos experimentos, é o tempo total de execução das transações das rodadas, que mostrou significativa diferença na execução do método síncrono, o qual justamente executa apenas uma transação por vez, o que por consequência ocasiona um consumo superior de tempo em comparação aos métodos assíncronos.

Figura 14 – Gráficos *gantt* comparativo dos resultados obtidos em 1 (uma) rodada de 10 (dez) registros de cada um dos 3 (três) métodos



Fonte: Elaborada pelo autor

A Figura 14 ajuda a compreender melhor o comportamento de cada algoritmo, o

tempo necessário para execução de cada transação, e o tempo total de execução de toda a rodada.

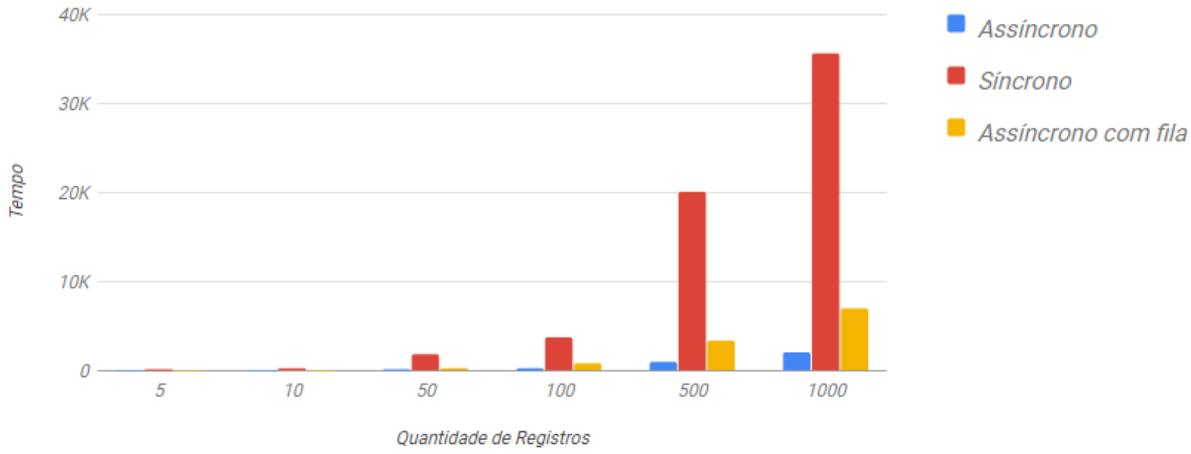
Tendo ciência do fator de tempo total de execução, uma analise destes resultados foi feita, a fim de identificar o tempo necessário para o envio das transações de cada cenário e algoritmo. A Tabela 4 traz os dados com os tempos totais obtidos durante os experimentos, bem como a representação gráfica, visível na Figural 15.

Tabela 4 – Média de tempo total das transações por método e quantidade de registros

Método	5	10	50	100	500	1000
Assíncrono	65,80	84,60	171,80	289,20	1093,40	2163,60
Síncrono	167,20	365,80	1942,40	3850,40	20174,40	35720,80
Assíncrono com fila	46,60	94,00	373,00	928,00	3471,60	7094,80

Fonte: Elaborada pelo autor

Figura 15 – Gráfico de coluna comparativo da média dos resultados obtidos em todas as rodadas e cenários de cada um dos 3 (três) métodos

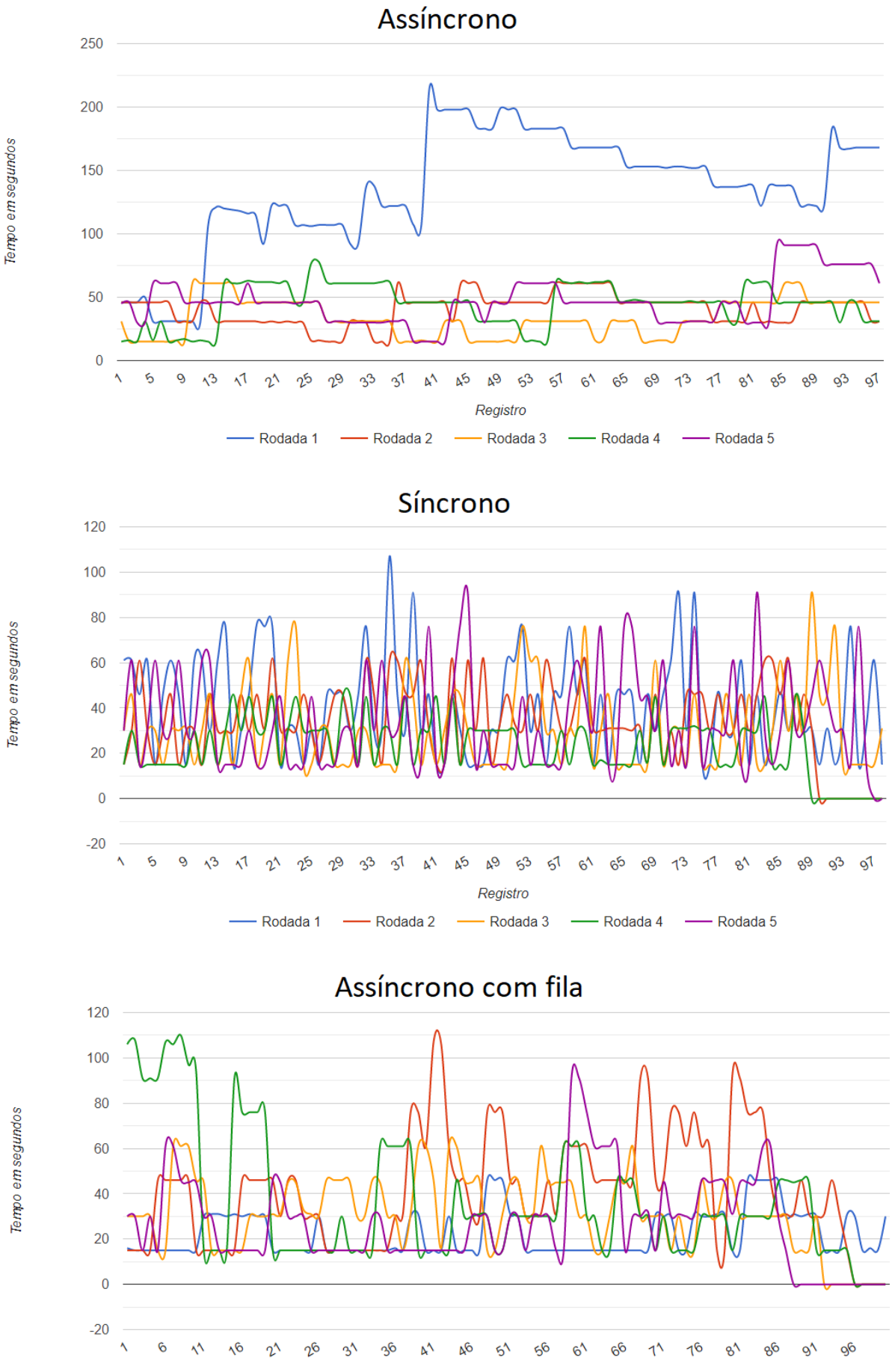


Fonte: Elaborada pelo autor

É evidente que o método síncrono se mostra a frente dos demais em todos os cenários, pois executa apenas uma transação por vez. Portanto tal circunstância precisa ser levada em consideração no momento de utilizar este método em uma aplicação. A seção 4.3 aborda estes aspectos para fazer as sugestões.

Sabendo então, que o método síncrono demanda o maior tempo total de execução, e, que quanto maior o tempo de execução, maior é a exposição à interferências da rede, e, que interferências na rede causam inconstâncias no tempo de execução de cada transação, é possível supor que o método síncrono possui maiores variações no tempo de cada transação.

Figura 16 – Gráficos de linha comparativo dos resultados obtidos das 5 (cinco) rodadas de 100 (cem) registros de cada um dos 3 (três) métodos



A Figura 16 confirma exatamente esta suposição, pois nela fica clara a diferença nos tempos de cada registro, sendo o método assíncrono o mais uniforme, pois todas as 5 (cinco) rodadas estão representadas no gráfico com linhas segmentadas que tendem a ser paralelas ao eixo x, se comparado com os demais métodos. O oposto disso ocorre com o método síncrono, onde em todas as suas rodadas possui valores variados, gerando linhas mais sinuosas e inconstantes.

Por sua vez, o método assíncrono com fila pode ser caracterizado como um meio termo, um tanto quanto lógico, pois utiliza o método assíncrono para envio de transações simultâneas, porém com uma limitação de apenas 5 (cinco) por vez. Por conta disso sua representação no gráfico possui linhas menos sinuosas comparadas ao método síncrono, porém não tão paralelas ao eixo x em comparação ao método assíncrono.

Este comportamento se repete em qualquer cenário, com qualquer volume de registros, porém torna-se mais visível a partir de 100 (cem) transações. Contudo devido ao grande número de pontos gerados nos gráficos nos cenários de 500 (quinhentos) e 1000 (mil) registros, a opção com 100 (cem) se torna a com melhor nitidez.

4.2.3 Média de desempenho geral dos cenários e métodos

O que talvez seja a principal análise dentre todas citadas nesta seção, é a de desempenho geral dos diferentes métodos, pois, dentro de um contexto convencional, onde quanto mais rápido a trilha de auditoria é enviada ao *blockchain*, menor é o tempo de vulnerabilidade da aplicação.

Logo, considerando todos os aspectos citados anteriormente, e levando em conta a exclusão dos *outliers*, conforme explicado na seção 4.1, é possível resumir os resultados de duas formas.

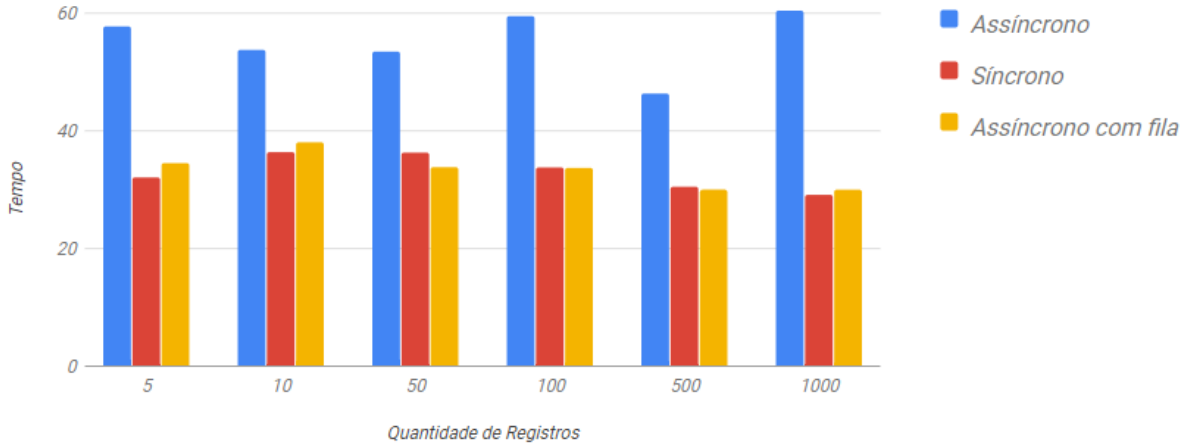
A primeira delas trata de avaliar o desempenho médio por método e volume de dados, conforme pode ser visto na Tabela 5 e Figura 17, permitindo observar o comportamento de cada algoritmo com diferentes volumes de transações, onde é possível observar a evidente demasia no tempo médio do algoritmo assíncrono.

Tabela 5 – Média de tempo das transações por método e quantidade de registros

Método	5	10	50	100	500	1000
Assíncrono	57,88	53,89	53,60	59,63	46,47	60,56
Síncrono	32,21	36,51	36,43	33,91	30,63	29,25
Assíncrono com fila	34,64	38,18	33,95	33,83	30,17	30,13

Fonte: Elaborada pelo autor

Figura 17 – Gráfico de coluna comparativo da média dos resultados obtidos em todas as rodadas e cenários de cada um dos 3 (três) métodos



Fonte: Elaborada pelo autor

Com base nas análises anteriores, que sugerem variações no desempenho das transações por decorrência de ações e demandas externas à rede, é possível especular que as próprias transações paralelas da aplicação influenciam na demanda da rede e afetam o seu desempenho, isto provavelmente a partir de um número maior do que 5 (cinco) transações simultânea, pois, o método assíncrono com fila também possui transações paralelas, e não teve seu comportamento afetado por este fator.

Contundo, tecnicamente o cenário com 5 (cinco) registros não possui diferença alguma em sua execução entre os métodos assíncrono e assíncrono com fila, pois em ambos os casos todas as transações são feitas em paralelo, uma vez que o assíncrono com fila permite que até 5 registros sejam executados simultaneamente. Neste caso não foi possível identificar nenhuma sugestão lógica que explicasse tal comportamento que gerou médias tão distintas entre os dois métodos.

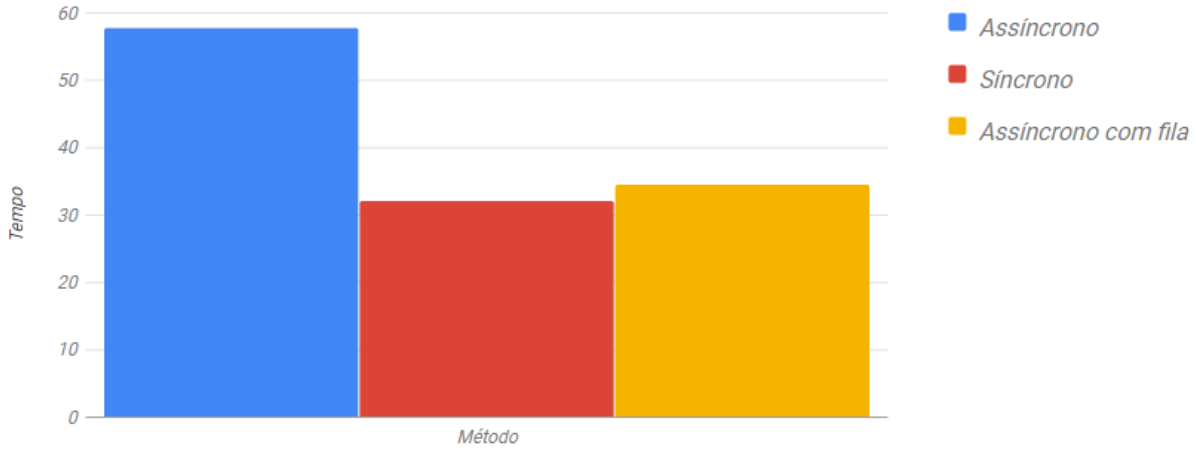
Como iniciado anteriormente, a segunda forma de analise dos dados, demonstra uma visão geral do desempenho de cada algoritmo, reunindo os resultados de todos os cenários experimentados. A Tabela 6 exibe os resultados obtidos através deste contexto, e a Figura 18 permite observar de forma visual e comparativa.

Tabela 6 – Média de tempo das transações por método

Método	Média
Assíncrono	55,99
Síncrono	30,24
Assíncrono com fila	30,54

Fonte: Elaborada pelo autor

Figura 18 – Gráfico de coluna comparativo da média dos resultados totais de cada um dos 3 (três) métodos



Fonte: Elaborada pelo autor

Neste gráfico é possível ver o resultado geral das médias já previstas na Figura 17, onde o método assíncrono com média é maior que os demais métodos, conforme visto anteriormente.

4.2.4 Padronização no tempo de duração das transações

Da mesma forma que na subseção 4.2.3, uma avaliação sobre a média geral dos cenários e algoritmos foi feita, considerando a padronização no tempo de execução nas transações.

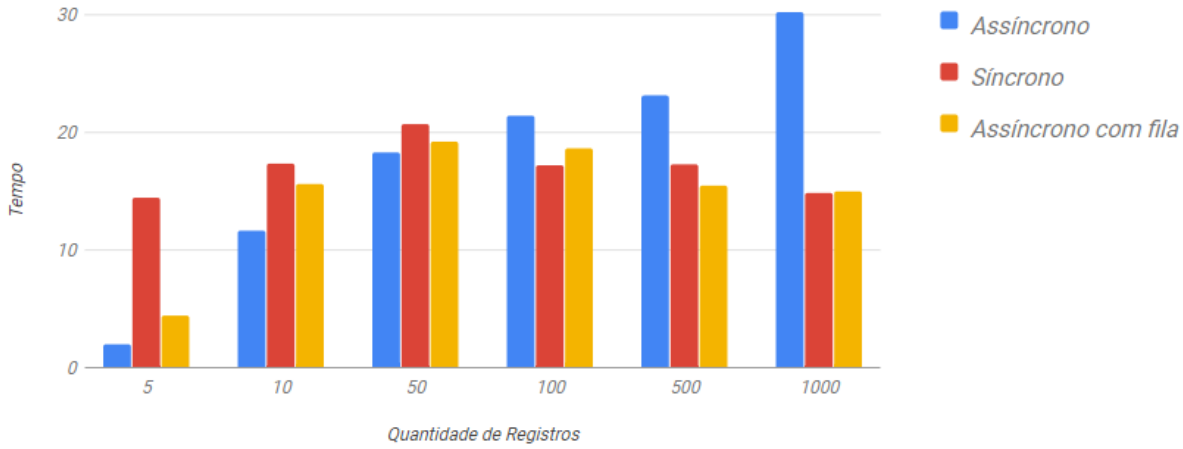
Esta análise foi feita levando em conta o desvio padrão de cada conjunto de dados, logo, quanto menor for o desvio padrão, mais uniforme e padronizado é o tempo de duração das transações. A Tabela 7 exibe os resultados obtidos através destes cálculos, e a Figura 19 mostra de forma visual os dados da tabela.

Tabela 7 – Média de desvio padrão das transações por método e quantidade de registros

Método	5	10	50	100	500	1000
Assíncrono	2,05	11,71	18,36	21,48	23,22	30,29
Síncrono	14,51	17,41	20,77	17,27	17,35	14,92
Assíncrono com fila	4,49	15,68	19,28	18,71	15,54	15,05

Fonte: Elaborada pelo autor

Figura 19 – Gráfico de coluna comparativo da média de desvios padrões dos resultados obtidos em todas as rodadas e cenários de cada um dos 3 (três) métodos



Fonte: Elaborada pelo autor

A crescente variação no desvio padrão com o aumento no volume de dados fica explícita no algoritmo assíncrono, o que pode ser interpretado novamente como uma instabilidade na rede, causada pelo aumento de transações simultâneas causada pelo próprio método.

Já no método assíncrono com fila, esta situação ocorre com no máximo 5 (cinco) transações, o que torna seu desvio padrão mais semelhante ao método síncrono, onde isso não acontece.

Um ponto importante a ser observado neste caso, é que o método assíncrono só se mostra inferior em termos de variação de tempo, nos cenários com mais de 100 registros.

Da mesma forma que na análise média, citada na subseção anterior, houve uma diferença evidente entre os métodos assíncrono e assíncrono com fila no cenário com 5 (cinco) registros, onde tecnicamente não há diferença. Isto pode ser um indicativo que 5 (cinco) rodadas para este cenário traz um número baixo de amostras para serem analisadas, o que talvez justificaria este resultado tão diferente entre os dois algoritmos.

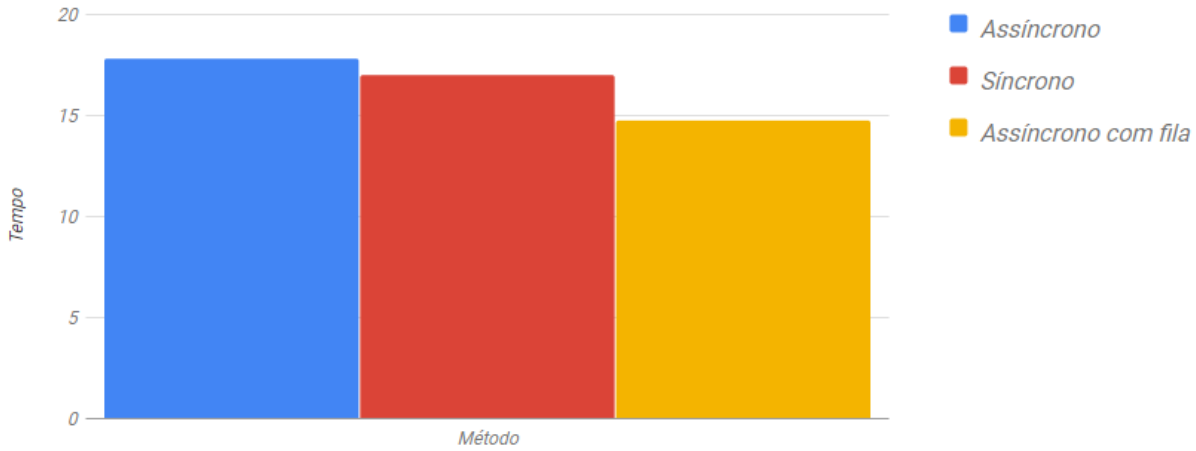
Ao final, para que seja possível analisar a padronização de cada método independendo ao volume de dados, a Tabela 8 mostra o desvio padrão geral de cada algoritmo, e a Figura 20 o gráfico correspondente.

Tabela 8 – Média de tempo das transações por método

Método	Desvio Padrão
Assíncrono	17,85
Síncrono	17,04
Assíncrono com fila	14,79

Fonte: Elaborada pelo autor

Figura 20 – Gráfico de coluna comparativo da média de desvios padrões dos resultados totais de cada um dos 3 (três) métodos



Fonte: Elaborada pelo autor

Este gráfico com o resultado total dos desvios padrões mostra que, apesar o método assíncrono ter tido um resultado inferior nos cenários com maior número de transações, os cenários com menor volume de registros acabaram compensando, o que gerou uma média muito semelhante aos demais métodos.

Porém, o método assíncrono com fila ainda ficou com o menor desvio padrão de todos, mostrando-se então o método com a menor variação no tempo médio de execução dentre os demais, o que pode ser um ponto importante a ser considerado dependendo da aplicação em que o algoritmo estará atuando.

4.3 SUGESTÕES

A partir das análises feitas na seção 4.2, é possível identificar aspectos positivos e negativos nos métodos e cenários experimentados, os quais foram utilizados como base para recomendações que podem ser vistas nessa seção.

Das quatro análises feitas, três podem ser utilizadas para tomada de decisão e colocadas em forma de sugestão, estas são:

- Tempo total de execução dos métodos;
- Média de desempenho geral dos cenários e métodos;
- Padronização no tempo das transações.

Considerando a combinação entre a média de desempenho geral e padronização nas transações, o algoritmo assíncrono com fila mostrou um desempenho superior aos demais métodos. Desta forma, aplicações que necessitam de uma combinação entre velocidade e padronização, este tende ser o melhor algoritmo.

Esta sugestão também se mostra a melhor opção para aplicações onde o aumento de transações simultâneas torna-se um problema, como o citado por Kalis (2018), embora tal situação não tenha ocorrido durante os experimentos deste trabalho.

Contudo, em aplicações onde transações simultâneas não geram problemas e considerando apenas o tempo total de execução, a melhor alternativa se torna o método assíncrono, principalmente em aplicações onde o objetivo é enviar os *logs* ao *blockchain* o mais breve possível.

Quanto ao método síncrono, nenhuma análise apontou vantagem ao utilizá-lo, devido à maior demanda de tempo utilizada e por consequência, maior exposição aos intempéries da rede. Por conta disso sua única aplicação logicamente viável seria em situações ondem não são possíveis transações simultâneas.

Além de tudo, vale reforçar que todos os experimentos foram feitos utilizando a rede Ropsten, citada na seção 3.2, com isso existem variações que podem ocorrer nos resultados obtidos mediante a alteração da rede, devido ao número de integrantes e demandas envolvidas.

5 CONCLUSÃO

O presente trabalho se propunha a utilizar a tecnologia *blockchain* para armazenamento de informações que sirvam como trilha de auditoria para autenticar a integridade de dados. Além disso, elaborar métodos alternativos para envio destas informações à rede *blockchain*, criar experimentos para avaliar métodos já existentes, e, ao final, evidenciar vantagens e desvantagens de cada algoritmo, propondo recomendações sobre a utilização destes.

Tais objetivos foram atendidos, através da criação da APEEER, com diferentes algoritmos de envio de transações, execução dos experimentos e extração dos resultados, como podem ser vistos nos capítulos 3 e 4. A seção 4.3, sugere o uso do algoritmo assíncrono com fila como a melhor opção na maioria das aplicações, pois teve média de 30,54 segundos de duração por transação, combinada com o menor desvio padrão dentre os 3 (três) algoritmos, sendo ele 14,79, além de não possuir restrições para o seu emprego.

Além dos objetivos propostos, espera-se contribuir com uma visão geral sobre a utilização do *blockchain* em trilhas de auditoria, e com exemplos práticos de aplicações de envio de dados à rede Ethereum. De forma que, todos os algoritmos e resultados serão disponibilizados de forma gratuita a qualquer interessado no assunto, conforme pode ser visto no Apêndice B.

Como sugestões a trabalhos futuros, algumas alternativas podem ser exploradas a fim de completar os resultados obtidos neste trabalho. A primeira delas seria experimentos relacionados a variação da fila do método assíncrono com fila, com o objetivo de identificar um possível ponto onde o número de transações simultâneas começam a afetar o desempenho da rede.

Além disso, experimentos relacionados ao aumento de números de rodadas para gerar um número maior de amostras, principalmente nos cenários com menor volume de dados. E por último, utilizar outras redes diferentes da Ropsten, a fim de identificar possíveis diferenças que possam haver no desempenho de cada uma.

REFERÊNCIAS

ATZORI, M. Blockchain technology and decentralized governance: Is the state still necessary? *Available at SSRN 2709713*, 2015. Citado 2 vezes nas páginas 21 e 22.

BAHGA, A.; MADISETTI, V. K. Blockchain platform for industrial internet of things. *Journal of Software Engineering and Applications*, Scientific Research Publishing, v. 9, n. 10, p. 533–546, 2016. Citado 2 vezes nas páginas 22 e 23.

BELLARE, M.; YEE, B. *Forward integrity for secure audit logs*. [S.l.], 1997. Citado na página 12.

BOSWORTH, S.; KABAY, M. E. *Computer security handbook*. [S.l.]: John Wiley & Sons, 2002. Citado 2 vezes nas páginas 11 e 16.

BUTERIN, V. *On Public and Private Blockchains*. 2015. Disponível em: <https://blog-.ethereum.org/2015/08/07/on-public-and-private-blockchains/>. Acesso em: 09 nov. 2020. Citado 2 vezes nas páginas 23 e 24.

BUTERIN, V. et al. A next-generation smart contract and decentralized application platform. *white paper*, v. 3, n. 37, 2014. Citado 2 vezes nas páginas 18 e 19.

CROSBY, M. et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, v. 2, n. 6-10, p. 71, 2016. Citado na página 12.

DANILIN, P. I.; LUKIN, A. A.; RESHETOVA, E. N. Assessment organization service based on ethereum platform. In: *Proceedings of the 5th International Conference on Actual Problems of System and Software Engineering*. [S.l.: s.n.], 2017. Citado na página 32.

DIKA, A. *Ethereum smart contracts: Security vulnerabilities and security tools*. Dissertação (Mestrado) — NTNU, 2017. Citado na página 32.

DISTRICT0X. *What Is Ethereum?* 2020. Disponível em: <https://education.district0x-.io/general-topics/understanding-ethereum/what-is-ethereum/>. Acesso em: 11 nov. 2020. Citado na página 24.

ELMASRI, R.; NAVATHE, S. *Fundamentals of database systems*. [S.l.]: Pearson Education India, 2008. Citado na página 17.

GALEN, D. et al. Blockchain for social impact: moving beyond the hype. *Center for Social Innovation, RippleWorks. https://www. gsb. stanford. edu/sites/gsb/files/publication-pdf/study-blockchain-impact-moving-beyond-hype_0. pdf*, 2018. Citado na página 12.

GIL, A. C. *Métodos e técnicas de pesquisa social*. [S.l.]: 6. ed. Editora Atlas SA, 2008. Citado na página 14.

HAN, S. J. *Blockchain Demo*. 2020. Disponível em: <https://blockchaindemo.io/>. Acesso em: 2 nov. 2020. Citado 3 vezes nas páginas 19, 20 e 21.

HAWTHORN, P. et al. *Statewide databases of registered voters*. [S.l.]: ASSOC COMPUTING MACHINERY 1515 BROADWAY, NEW YORK, NY 10036 USA, 2006. Citado na página 16.

HU, Y.-C. et al. Hierarchical interactions between ethereum smart contracts across testnets. In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. [S.l.: s.n.], 2018. Citado na página 32.

KALIS, R. Using blockchain to validate audit trail data in private business applications. *University of Amsterdam, Jun*, 2018. Citado 9 vezes nas páginas 26, 29, 30, 32, 33, 34, 36, 37 e 55.

KALIS, R.; BELLOUM, A. Validating data integrity with blockchain. In: IEEE. *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.], 2018. p. 272–277. Citado 10 vezes nas páginas 4, 5, 12, 13, 14, 18, 26, 27, 28 e 29.

KARAMITSOS, I.; PAPADAKI, M.; BARGHUTHI, N. B. A. Design of the blockchain smart contract: A use case for real estate. *Journal of Information Security*, Scientific Research Publishing, v. 9, n. 3, p. 177–190, 2018. Citado 2 vezes nas páginas 22 e 23.

MCDOWELL, R. Focus on quality-validation of spectrometry software–audit trails for spectrometer software-columnist bob mcdowall discusses the use of audit trails in the software applications used to. *Spectroscopy-Eugene*, [Springfield, OR: Aster Pub. Corp., c1986-, v. 22, n. 4, p. 14–21, 2007. Citado 2 vezes nas páginas 17 e 18.

MITT, S. *Plokiahela Rakendus–Hyperledger Fabric uuring*. 2018. Citado 2 vezes nas páginas 21 e 22.

MOTA, J. S. et al. Tdchain: uso de blockchain na cadeia de distribuição de medicamentos. In: *21º CONGRESSO DE COMPUTAÇÃO E TECNOLOGIAS DA INFORMAÇÃO*. [S.l.: s.n.], 2019. Citado na página 20.

MYSQL. *MySQL 8.0 Reference Manual, Using Triggers*. 2020. Disponível em: <https://dev.mysql.com/doc/refman/8.0/en/triggers.html>. Acesso em: 26 out. 2020. Citado 2 vezes nas páginas 17 e 18.

NAKAMOTO, S. *Bitcoin: A peer-to-peer electronic cash system*. [S.l.], 2008. Citado na página 18.

PETERSON, Z. N. et al. Design and implementation of verifiable audit trails for a versioning file system. In: *FAST*. [S.l.: s.n.], 2007. v. 7, p. 20–20. Citado na página 16.

RORATTO, R.; DIAS, E. D. Segurança da informação de produção e operações: Um estudo sobre trilhas de auditoria em sistemas de banco de dados. *JISTEM-Journal of Information Systems and Technology Management*, SciELO Brasil, v. 11, n. 3, p. 717–734, 2014. Citado 2 vezes nas páginas 16 e 18.

SALLACH, D. L. A deductive database audit trail. In: *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's*. [S.l.: s.n.], 1992. p. 314–319. Citado na página 17.

SANT'ANA, R. R. P. d. RevoluÇÃo blockchain: Os contratos inteligentes. In: PONTIFíCIA UNIVERSIDADE CATóLICA DE MINAS GERAIS (PUC). [S.l.], 2018. Citado na página 12.

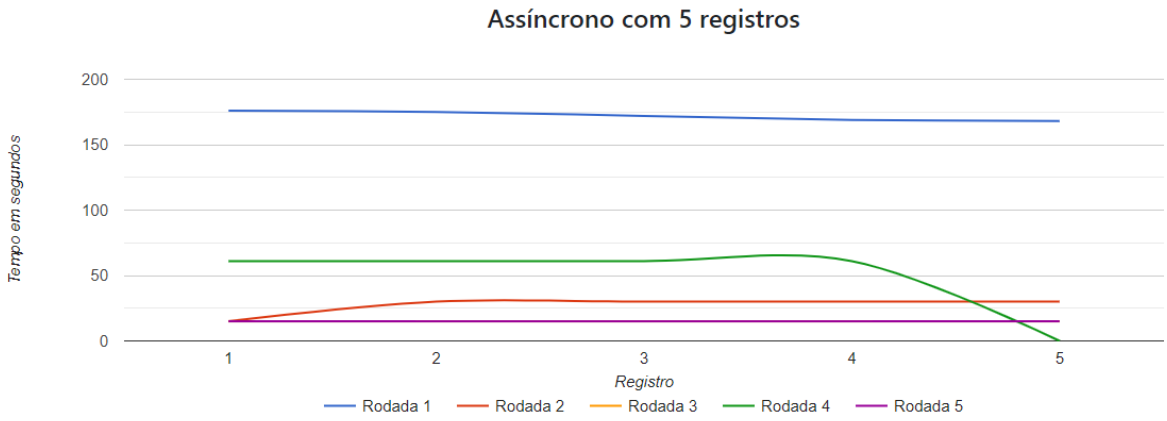
SCHNEIER, B.; KELSEY, J. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security (TISSEC)*, ACM New York, NY, USA, v. 2, n. 2, p. 159–176, 1999. Citado na página 11.

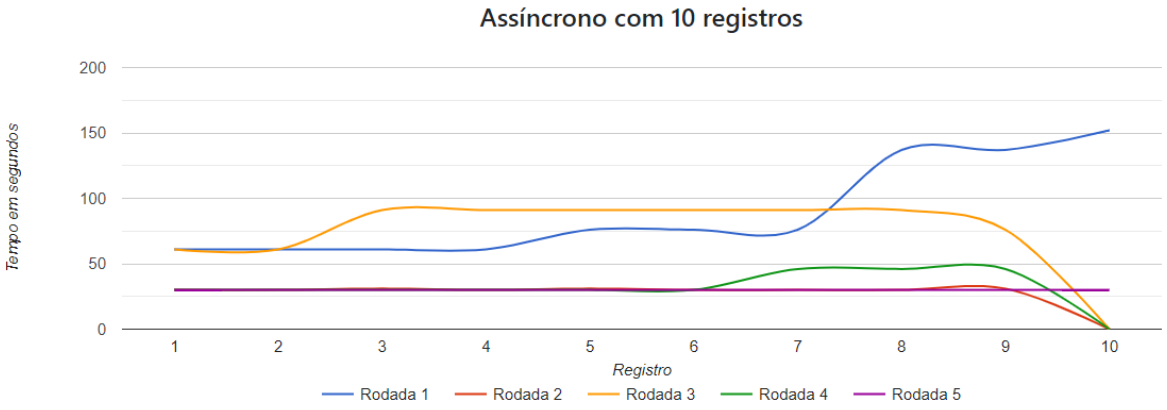
SIMON, F.; SANTOS, A. L. dos; HARA, C. S. Um sistema de auditoria baseado na análise de registros de log. *Escola Regional de Banco de Dados (ERBD'2008)*, 2008. Citado 2 vezes nas páginas 16 e 17.

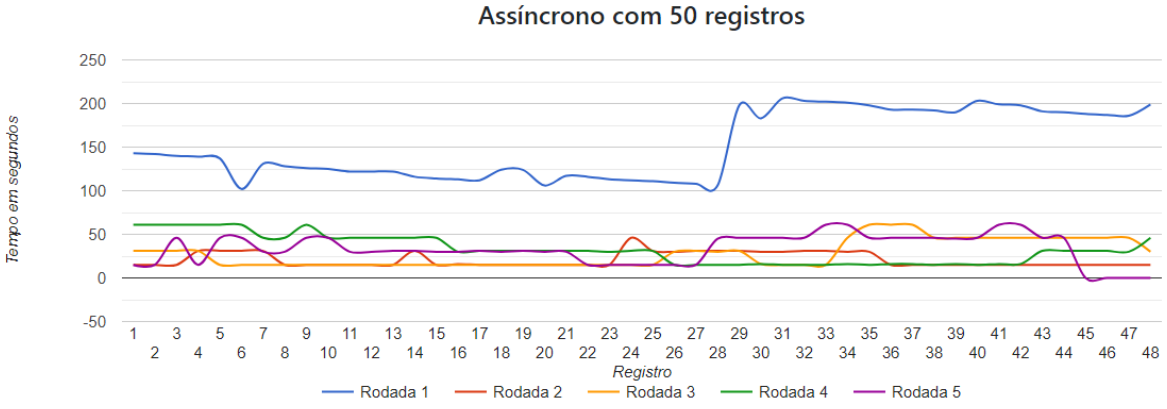
TURRIONI, J. B.; MELLO, C. H. P. Metodologia de pesquisa em engenharia de produção. *Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Itajubá. Itajubá: UNIFEI*, 2012. Citado na página 14.

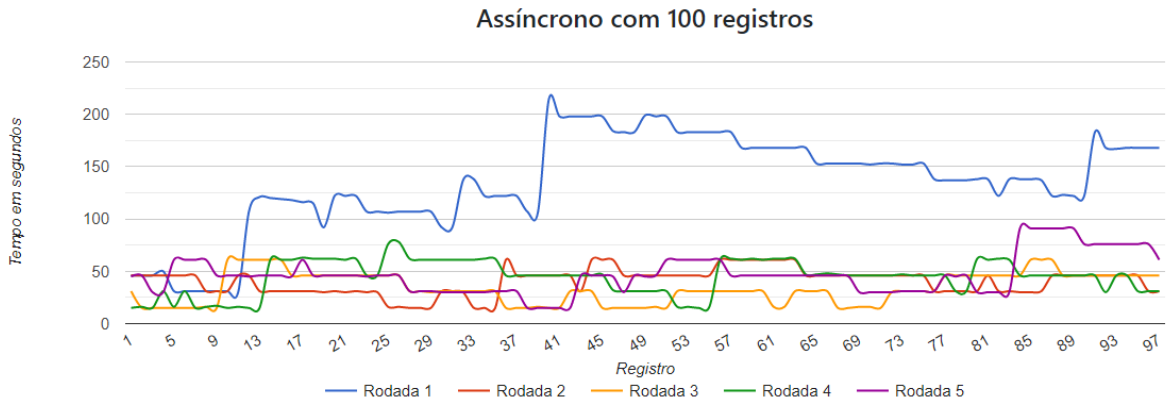
XU, W.; CHADWICK, D. W.; OTENKO, S. A pki based secure audit web server. *IASTED Communications, Network and Information and CNIS*, 2005. Citado 2 vezes nas páginas 11 e 16.

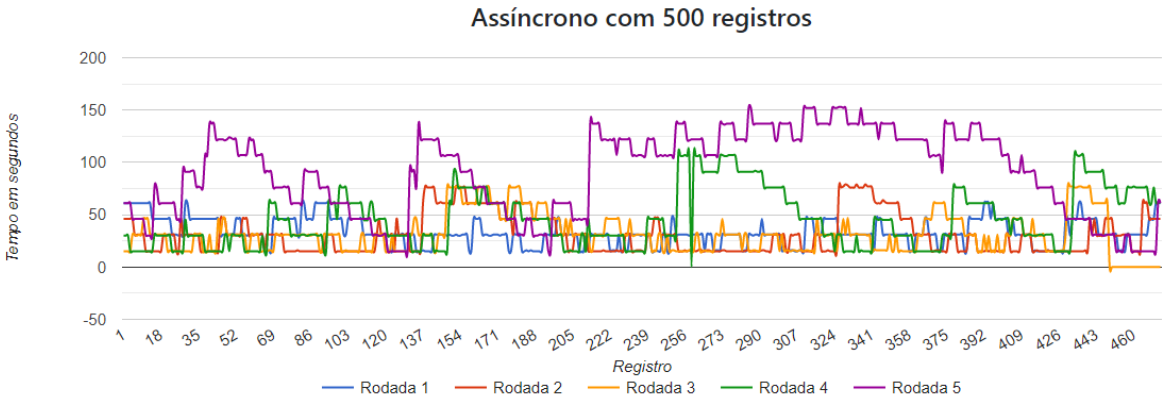
APÊNDICE A – GRÁFICOS DOS RESULTADOS

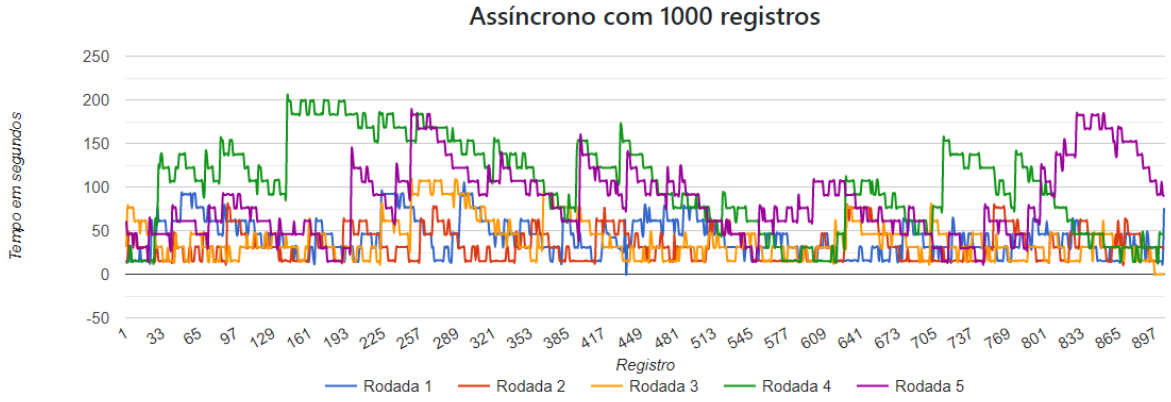


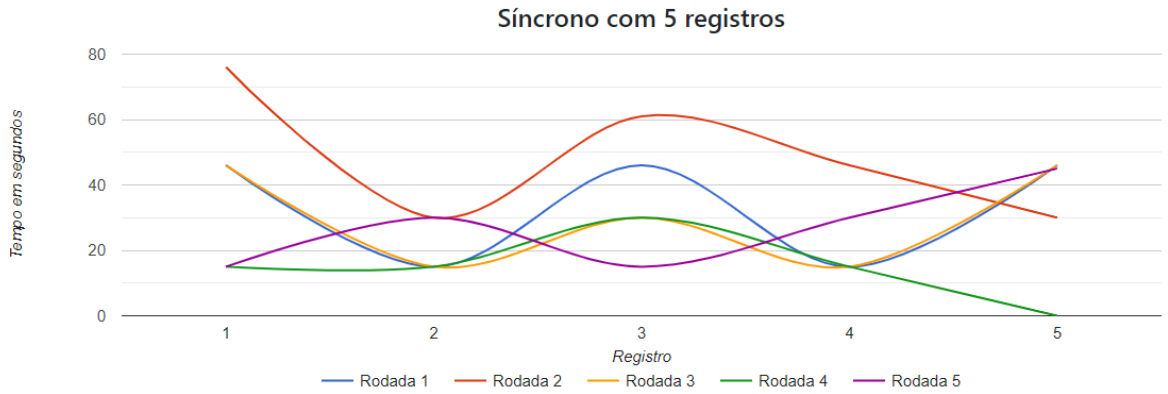


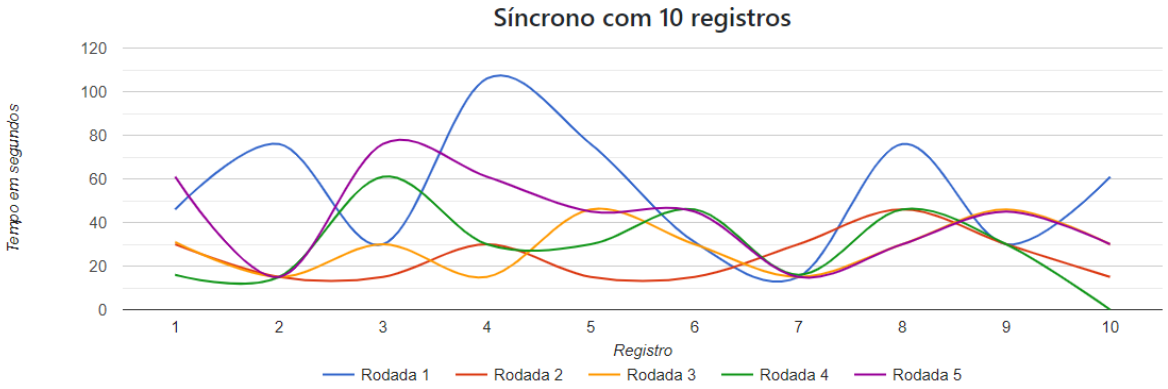


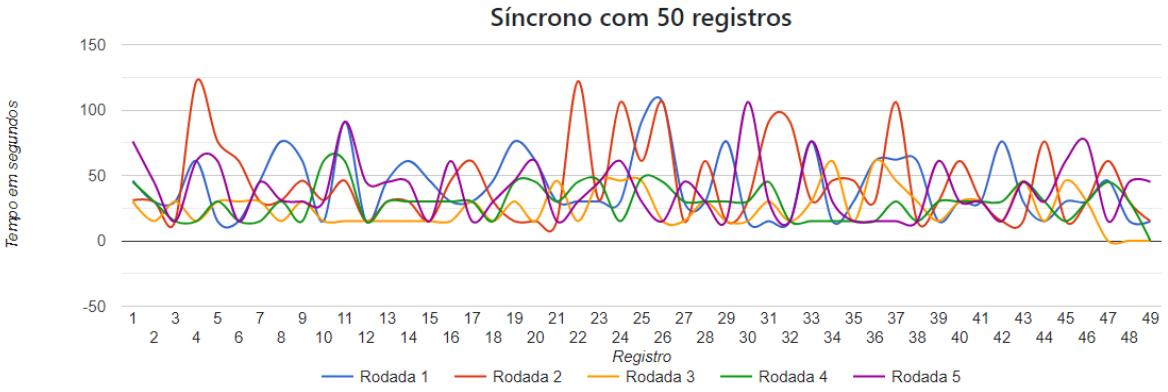

Assíncrono com 100 registros
Tempo em segundos
250
200
150
100
50
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

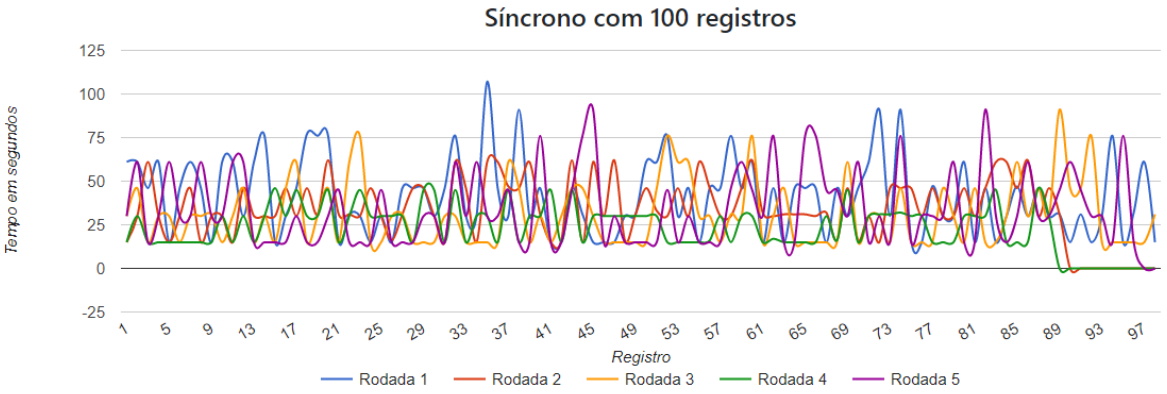

Assíncrono com 500 registros
Tempo em segundos
200
150
100
50
0
-50
1 18 35 52 69 86 103 120 137 154 171 188 205 222 239 256 273 290 307 324 341 358 375 392 409 426 443 460
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

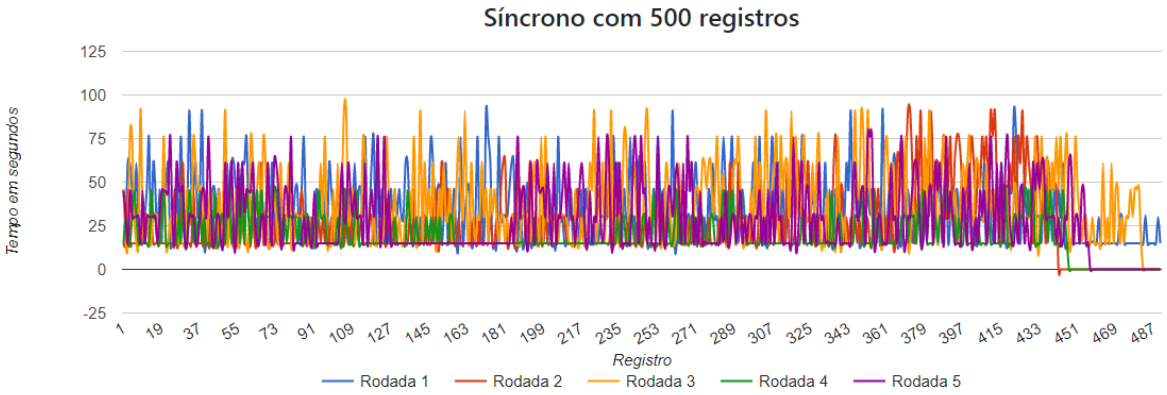

Assíncrono com 1000 registros
Tempo em segundos
250
200
150
100
50
0
-50
1 33 65 97 129 161 193 225 257 289 321 353 385 417 449 481 513 545 577 609 641 673 705 737 769 801 833 865 897
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

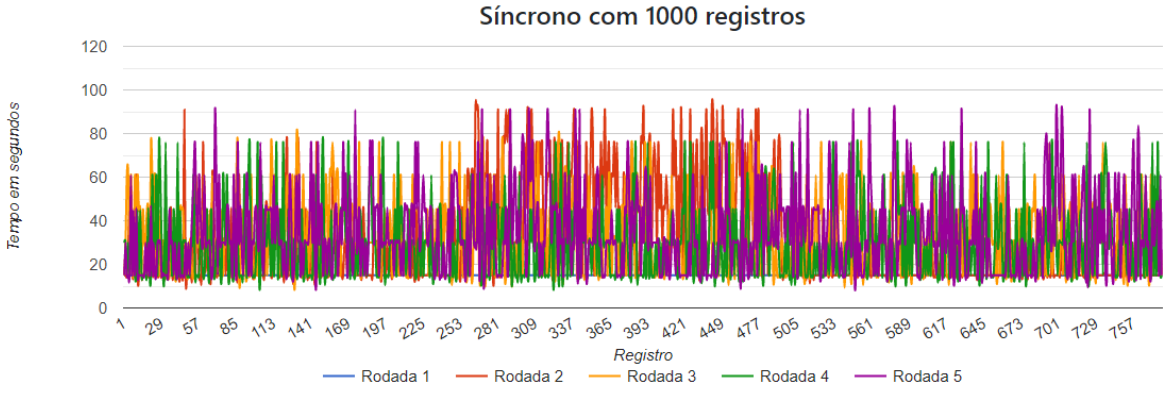

Síncrono com 5 registros
Tempo em segundos
80
60
40
20
0
1
2
3
4
5
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

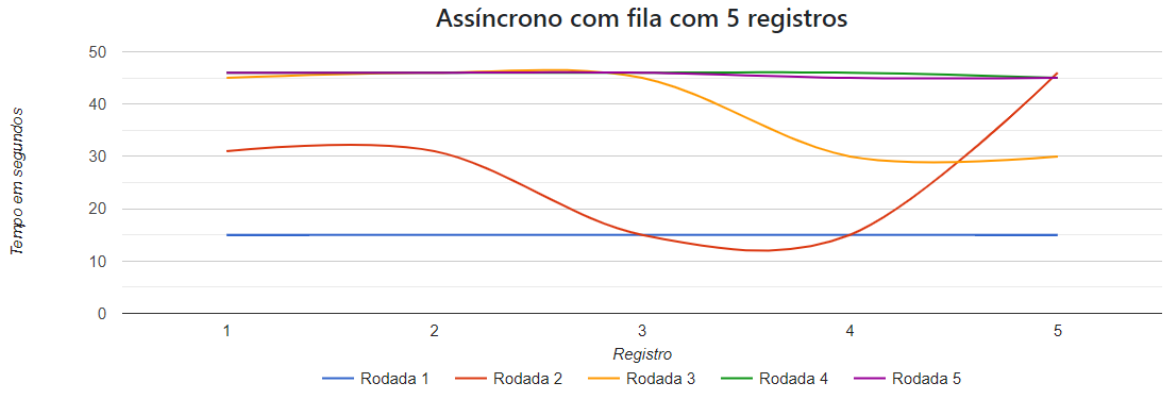

Síncrono com 10 registros
Tempo em segundos
120
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9
10
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

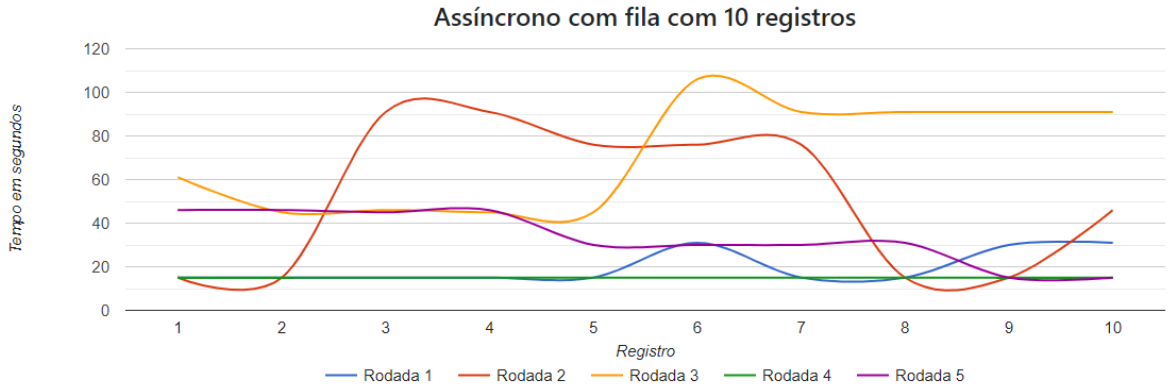

Síncrono com 50 registros
Tempo em segundos
150
100
50
0
-50
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

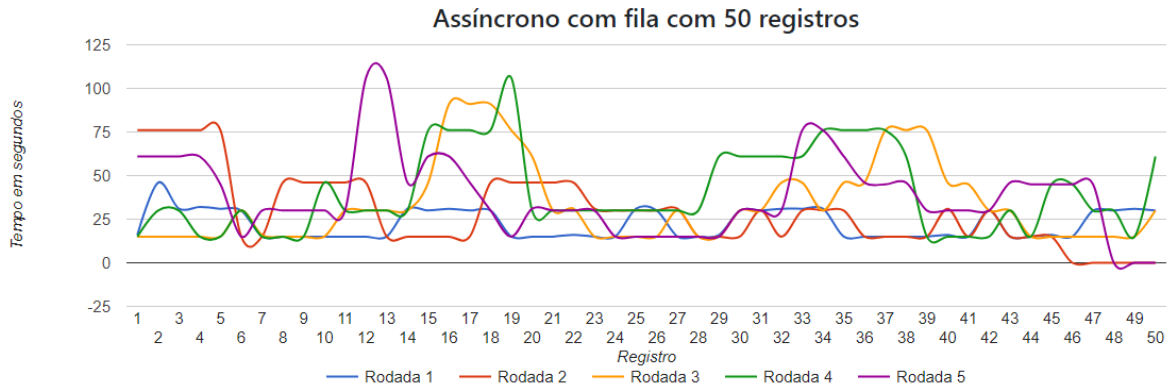

Síncrono com 100 registros
Tempo em segundos
125
100
75
50
25
0
-25
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

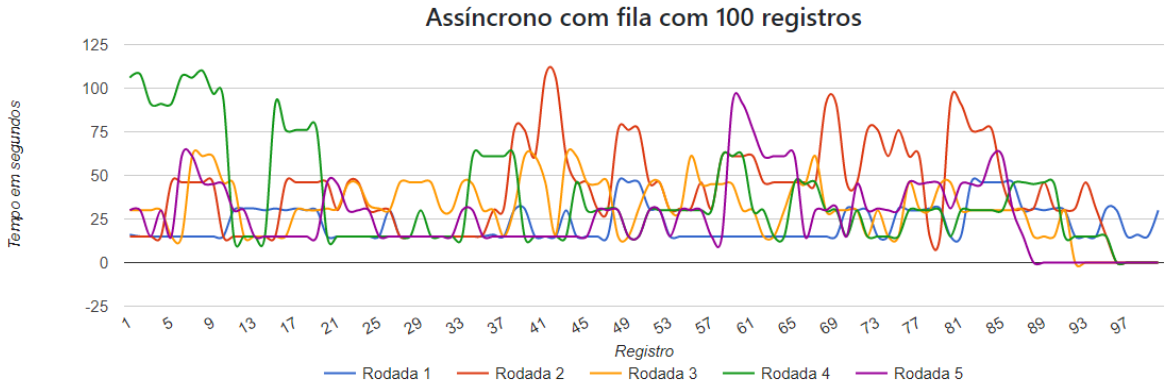

Síncrono com 500 registros
Tempo em segundos
125
100
75
50
25
0
-25
1 19 37 55 73 91 109 127 145 163 181 199 217 235 253 271 289 307 325 343 361 379 397 415 433 451 469 487
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

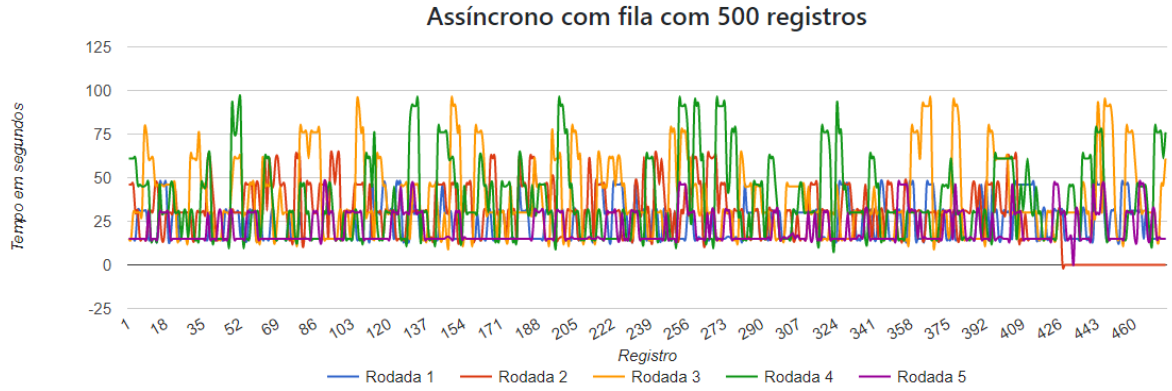

Síncrono com 1000 registros
Tempo em segundos
120
100
80
60
40
20
0
1 29 57 85 113 141 169 197 225 253 281 309 337 365 393 421 449 477 505 533 561 589 617 645 673 701 729 757
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

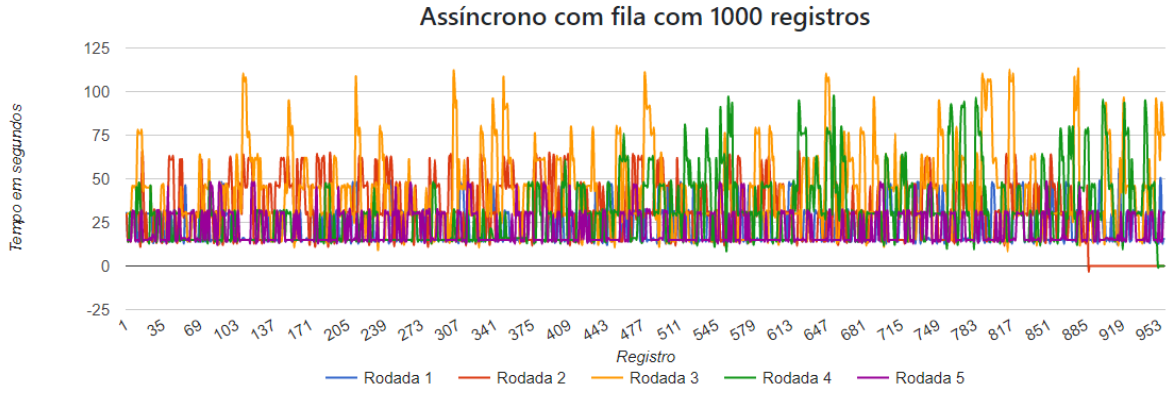

Assíncrono com fila com 5 registros
Tempo em segundos
50
40
30
20
10
0
1
2
3
4
5
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

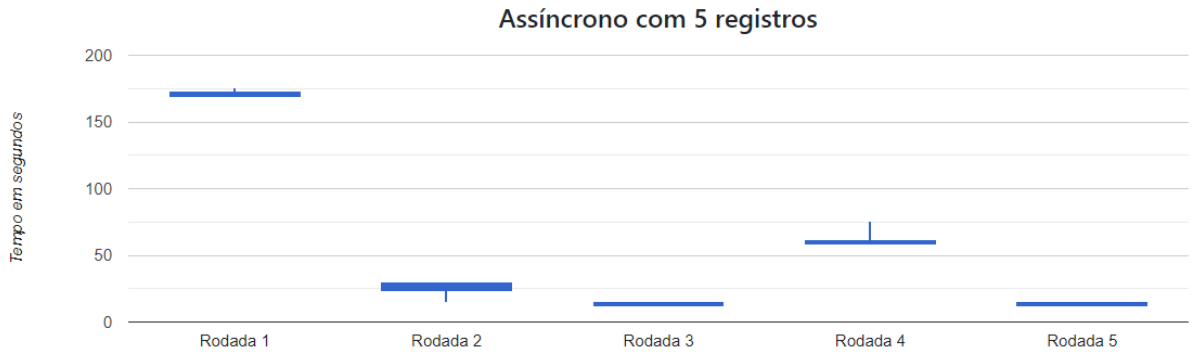

Assíncrono com fila com 10 registros
Tempo em segundos
120
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9
10
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

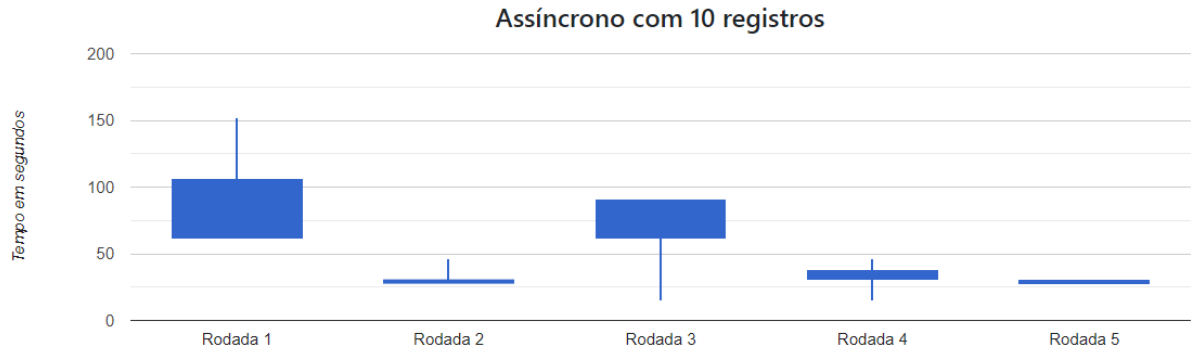

Assíncrono com fila com 50 registros
Tempo em segundos
125
100
75
50
25
0
-25
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

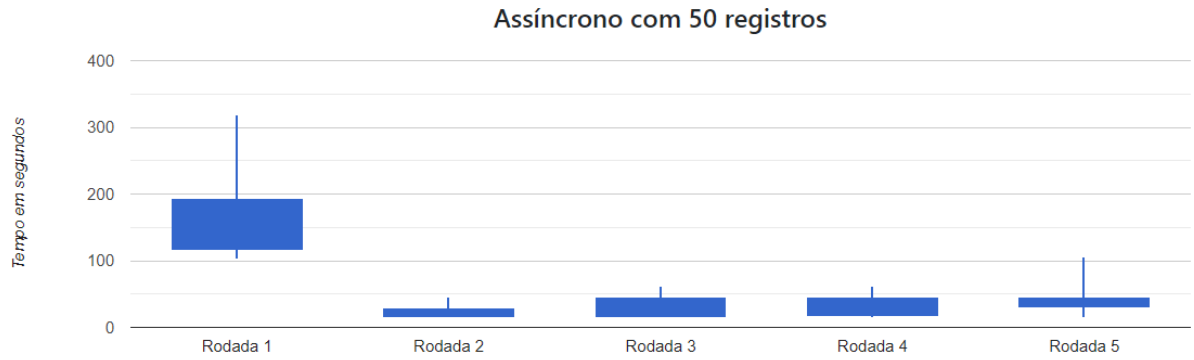

Assíncrono com fila com 100 registros
Tempo em segundos
125
100
75
50
25
0
-25
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

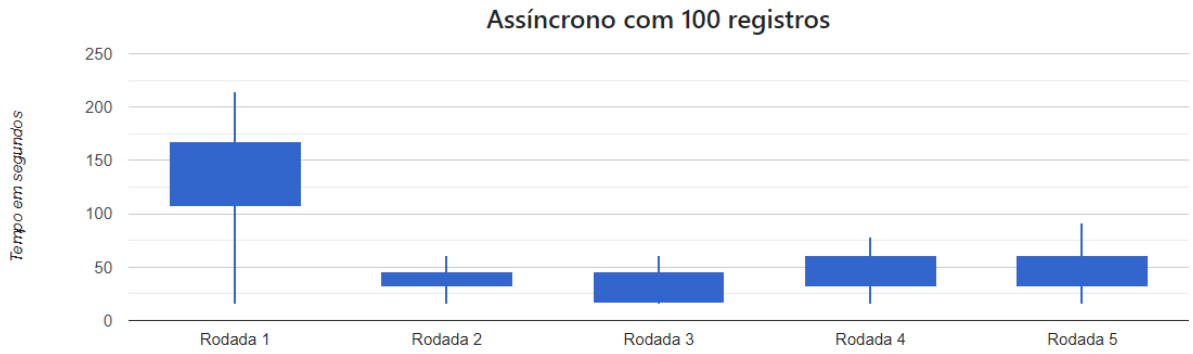

Assíncrono com fila com 500 registros
Tempo em segundos
125
100
75
50
25
0
-25
1 18 35 52 69 86 103 120 137 154 171 188 205 222 239 256 273 290 307 324 341 358 375 392 409 426 443 460
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

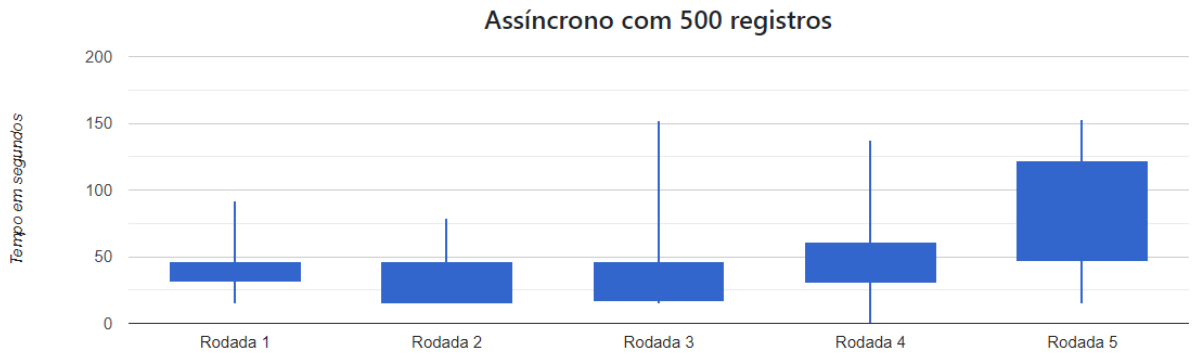

Assíncrono com fila com 1000 registros
Tempo em segundos
125
100
75
50
25
0
-25
1 35 69 103 137 171 205 239 273 307 341 375 409 443 477 511 545 579 613 647 681 715 749 783 817 851 885 919 953
Registro
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

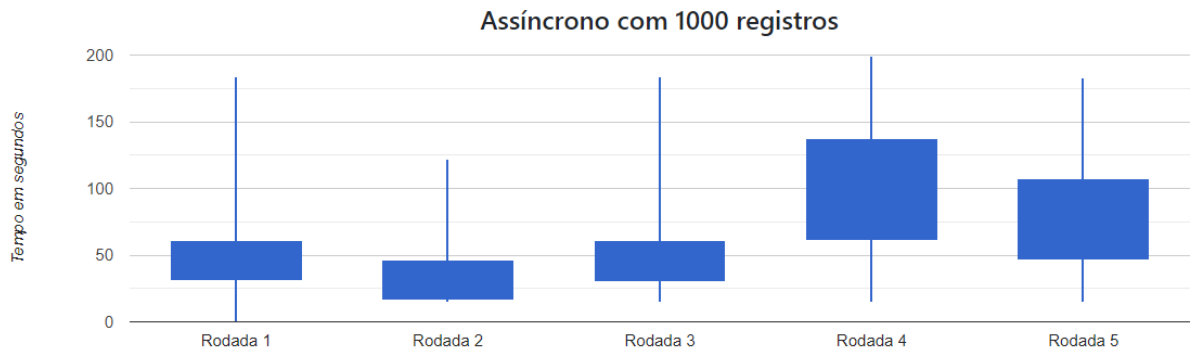

Assíncrono com 5 registros
Tempo em segundos
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

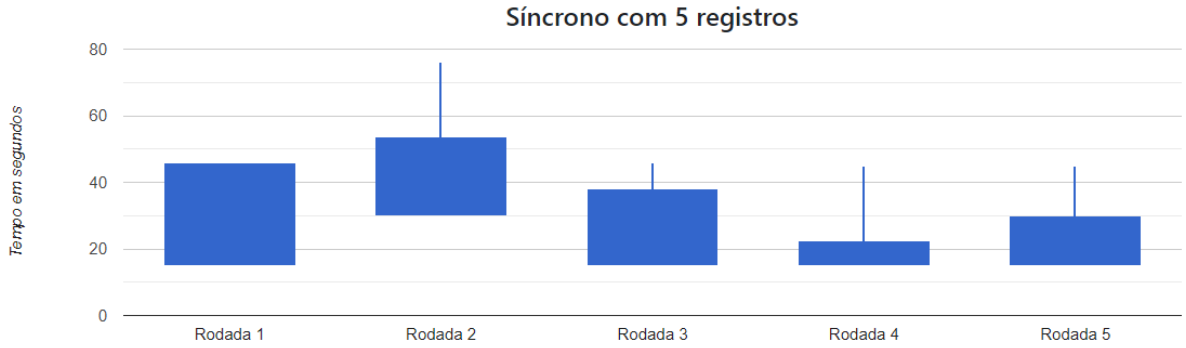

Assíncrono com 10 registros
Tempo em segundos
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

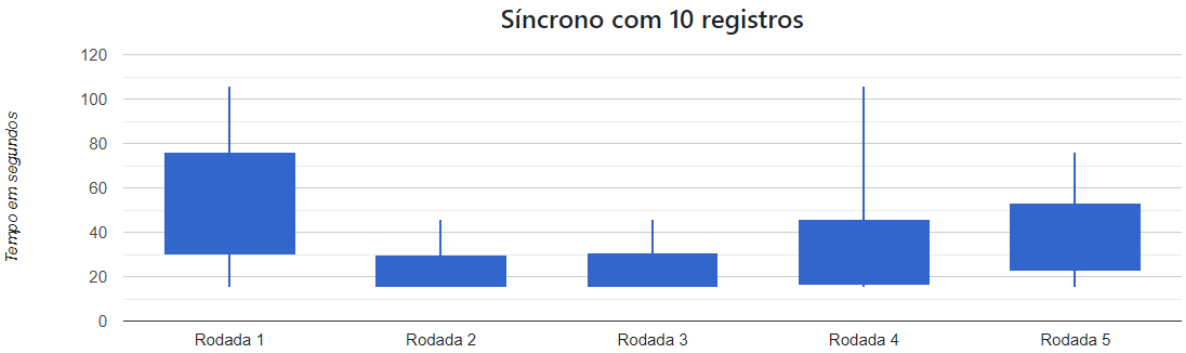

Assíncrono com 50 registros
Tempo em segundos
400
300
200
100
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

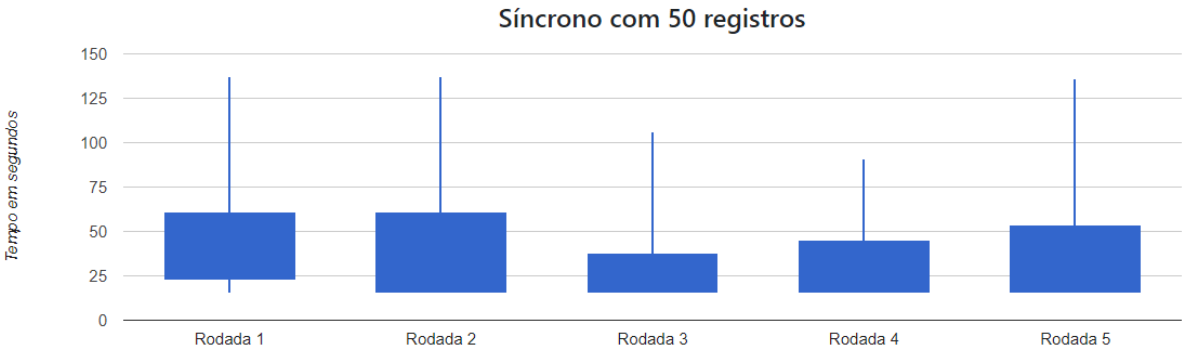

Assíncrono com 100 registros
Tempo em segundos
250
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

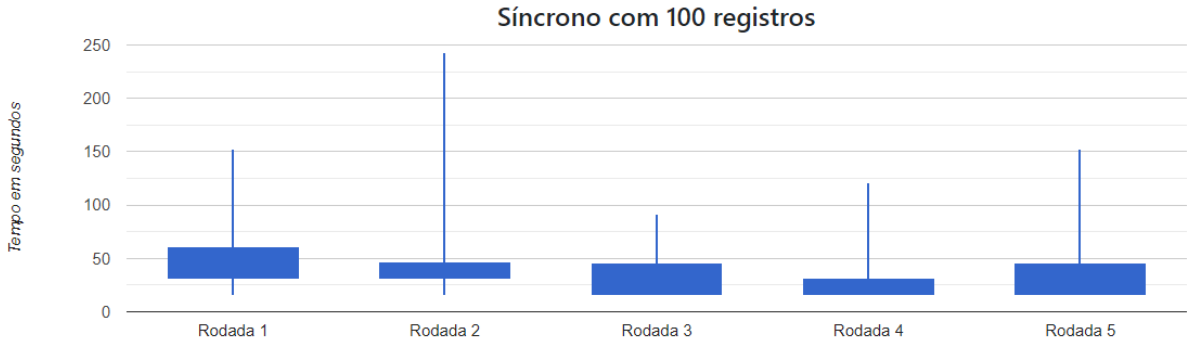

Assíncrono com 500 registros
Tempo em segundos
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

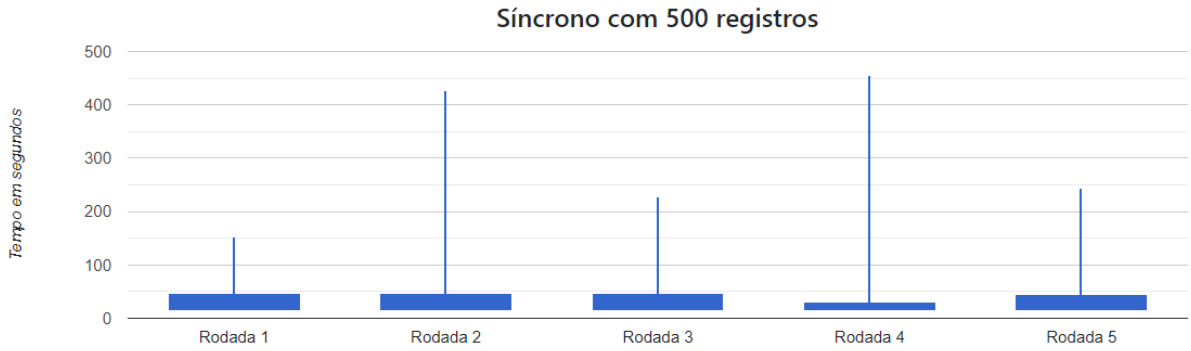

Assíncrono com 1000 registros
Tempo em segundos
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

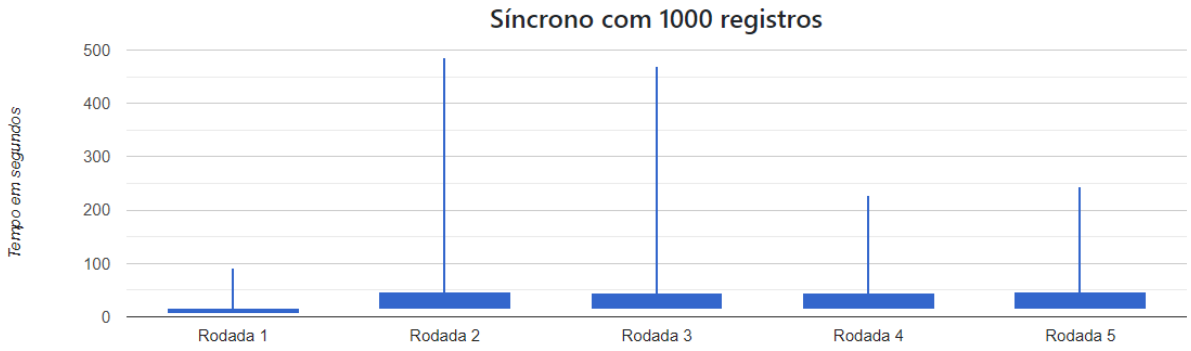

Síncrono com 5 registros
Tempo em segundos
80
60
40
20
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

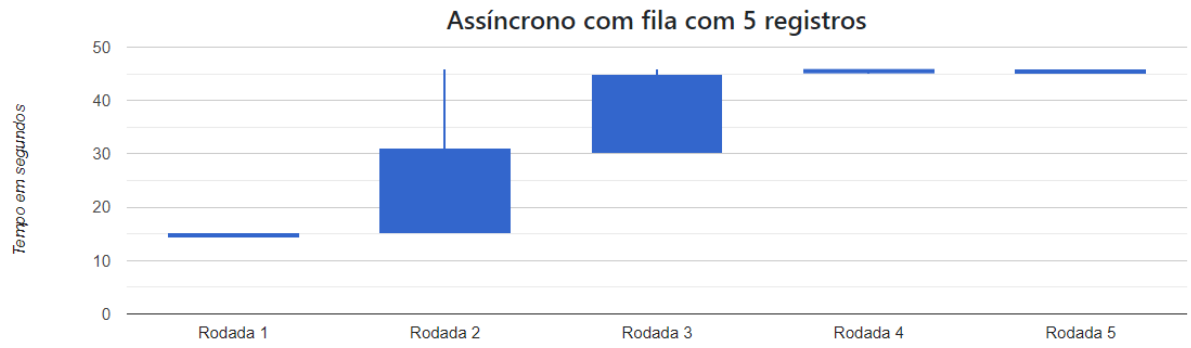

Síncrono com 10 registros
Tempo em segundos
120
100
80
60
40
20
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

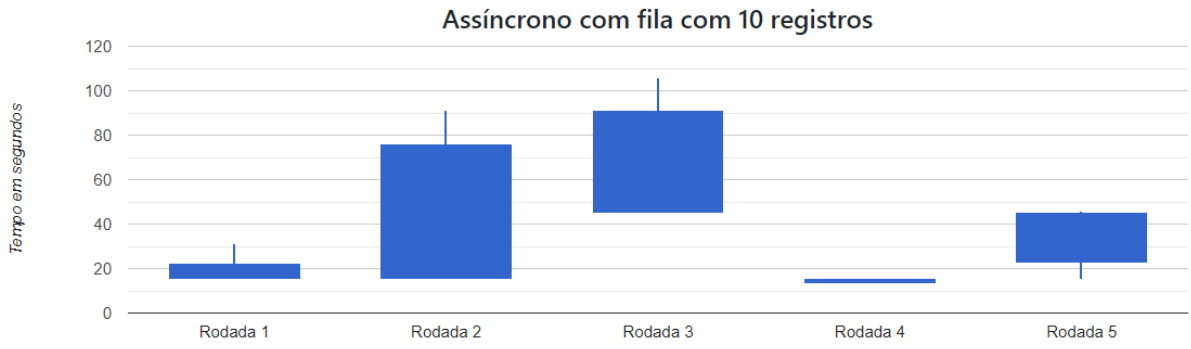

Síncrono com 50 registros
Tempo em segundos
150
125
100
75
50
25
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

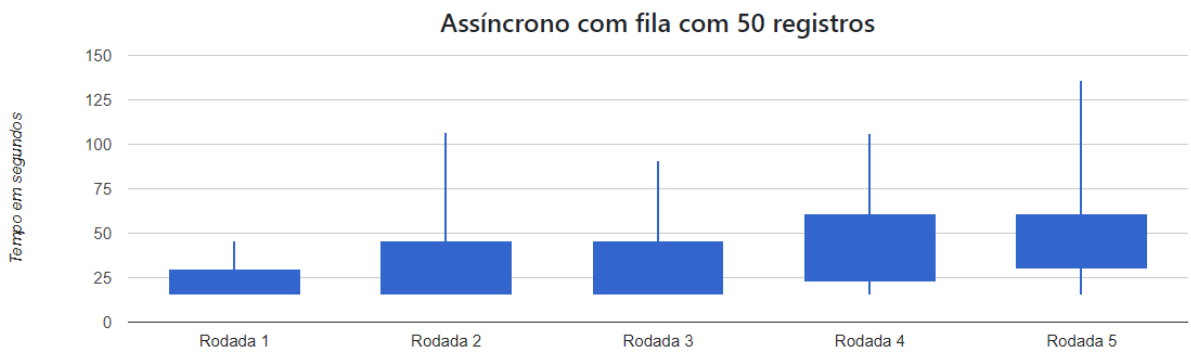

Síncrono com 100 registros
Tempo em segundos
250
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

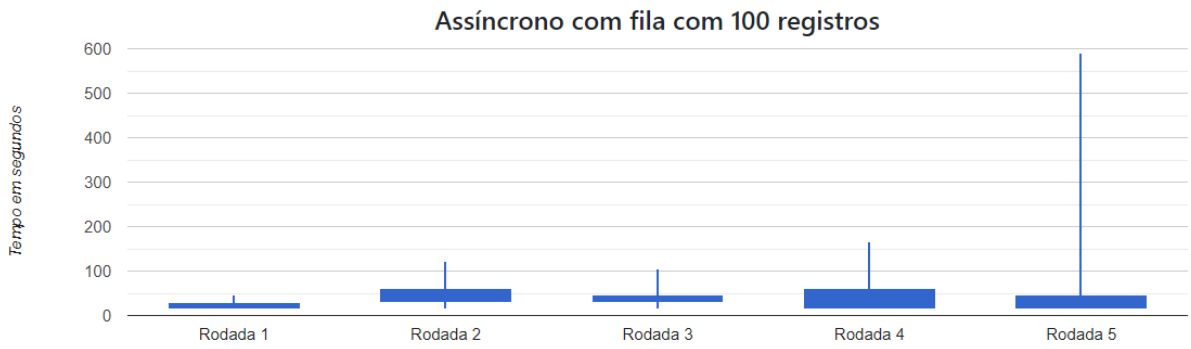

Síncrono com 500 registros
Tempo em segundos
500
400
300
200
100
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

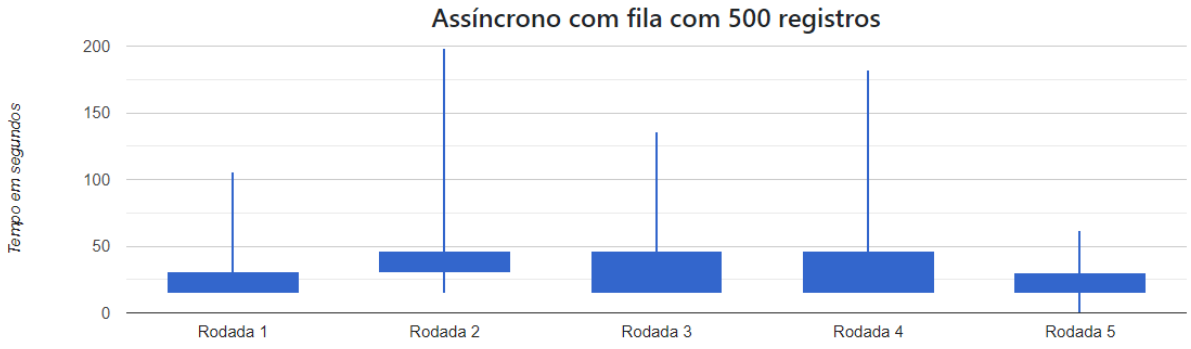

Síncrono com 1000 registros
Tempo em segundos
500
400
300
200
100
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

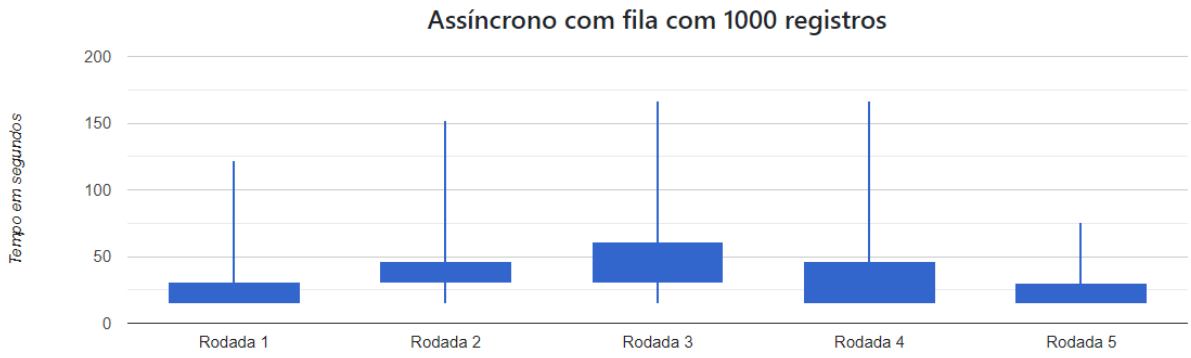

Assíncrono com fila com 5 registros
Tempo em segundos
50
40
30
20
10
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5


Assíncrono com fila com 10 registros
Tempo em segundos
120
100
80
60
40
20
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5


Assíncrono com fila com 50 registros
Tempo em segundos
150
125
100
75
50
25
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5


Assíncrono com fila com 100 registros
Tempo em segundos
600
500
400
300
200
100
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5


Assíncrono com fila com 500 registros
Tempo em segundos
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5


Assíncrono com fila com 1000 registros
Tempo em segundos
200
150
100
50
0
Rodada 1
Rodada 2
Rodada 3
Rodada 4
Rodada 5

APÊNDICE B – CÓDIGO FONTE

Fonte: Todo o código fonte da APPPER encontra-se disponível em https://abre.ai/cNpU acessado em 30/05/2021.